ORI: CAPTURING GRADIENT SUBSPACE DRIFT VIA STREAMING k-PCA FOR VERY LOW-RANK LLM TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

A prominent family of low-rank LLM optimizers, pioneered by GaLore, performs gradient-subspace projection: at each step, they project the gradient into a low-dimensional subspace and map the update back to the full model. This type of training reduces memory, but its effectiveness depends on how the algorithm adapts to a changing gradient subspace as the model is being optimized. Existing work either recomputes the gradient subspace at certain intervals, or performs post-hoc adaption to catch up to the moving subspace. In this work, we explicitly identify that in practice, the LLM gradient subspace changes slowly and stably—a phenomenon we call *gradient subspace drift*. We argue that explicitly leveraging this slow drift is key to ensuring stable learning in extremely low-rank scenarios. To this end, we propose Ori, a novel low-rank optimizer for training LLMs that achieves state-of-the-art accuracy under stringent low-rank constraints. To capture the shifting principal gradient directions over time, Ori employs a streaming k-PCA algorithm inspired by Oja++, whose low per-step cost enables high-frequency, on-the-fly subspace tracking. In practice, Ori reduces the performance gap of the state-of-the-art low-rank optimizers to full-rank optimizers by up to 69.3% on a 1B LLaMA pretraining task, and achieves better performance on 15 out of 16 task scenarios in Roberta-Base fine-tuning. We further propose a novel theoretical analysis establishing dynamic regret bounds for subspace tracking under drift, providing formal guarantees that yield sublinear regret in the noiseless setting. Together, these contributions allow Ori to further push the performance and efficiency frontiers of gradient-projecting optimizers, narrowing the gap to full-rank training for LLMs. Code can be found at https://anonymous.4open.science/r/Ori-Submission-2132.

1 INTRODUCTION

Training large language models (LLMs) under strict memory budgets has motivated a variety of low-rank optimization methods that restrict either the weights or the gradients to a low-dimensional subspace (Hu et al., 2022; Zhao et al., 2024; Mo et al., 2025). This broad field is primarily divided into two families. *Weight-subspace methods*, such as LoRA, adapt the model weights directly (Hu et al., 2022; Lialin et al., 2023). More recently, *gradient-subspace methods*, pioneered by GaLore (Zhao et al., 2024), project the gradients into a low-rank subspace before the optimizer step. The latter approach and its subsequent work have proven particularly effective, demonstrating superior memory savings and performance by focusing on the dynamics of the optimizer itself (Zhao et al., 2024; Zhang et al., 2024; Lialin et al., 2023; Han et al., 2024; Dettmers et al., 2023).

Yet all gradient-subspace methods in this promising family face a core technical challenge: that the principal gradient directions, instead of being static, evolve as the model weights are updated throughout training. Consequently, the success of any such method hinges on how effectively it adapts to this changing subspace. GaLore tackles this by periodically recomputing the entire subspace from scratch via an expensive SVD (Zhao et al., 2024), a process that acts as a "cold restart" and discards prior subspace information. More recent methods, such as Online Subspace Descent (OSD) (Liang et al., 2024), move towards more continuous adaptation by using a secondary, non-convex optimization loop to "catch up" to the moving subspace.

However, existing adaptation strategies may overlook a crucial empirical observation that we make explicit in this work: the gradient subspace does not jump erratically but, rather, drifts smoothly after warm-up. In other words, the projector at step t is predictably close to that at $t - \Delta t$ for small Δt (e.g. 10). This slow, structured drift suggests a different perspective. Rather than recomputing the subspace (as in GaLore (Zhao et al., 2024)) or optimizing it with non-convex gradient descent (as in OSD (Liang et al., 2024)), we can track it with a streaming update that makes small, frequent rotations aligned with the most recent gradients, while also preserving and transferring optimizer statistics across these rotations. In light of these observations, we arrive at a different design problem:

How might we continuously maintain a cheap, stable projector that follows a drifting gradient?

We thus introduce **Ori**, a streaming k-PCA via an **O**ja-style updated low-**r**ank **i**terative optimizer built precisely for this slow-drift regime, which replaces periodic SVD with streaming k-PCA via Oja++ (a robust variant of Oja's algorithm (Oja, 1982; Allen-Zhu & Li, 2017)). Why Oja-style updates? They align naturally with slow subspace drift: each step uses the current gradient to make a small, incremental rotation of a $d \times k$ orthonormal basis at cost $\mathcal{O}(dk^2)$ with only a lightweight re-orthogonalization, enabling high-frequency projector refreshes with negligible overhead so the basis stays synchronized with the gradient geometry (Allen-Zhu & Li, 2017). Unlike periodic SVDs or a separate manifold-descent loop, the continuous Oja++ update produces smooth basis changes, which lets Ori transfer Adam-style first and second moments seamlessly between successive bases instead of "cold-restarting" statistics. This combination of streaming efficiency, optimizer-state continuity, and theoretical robustness (gap-free tracking under clustered spectra) makes Oja-style updates a principled fit for the slow-drift regime.

Beyond computational economy and continuity, Ori has two properties that make it particularly apt for LLM training. First, the Oja++ algorithm enjoys gap-free convergence in streaming PCA: its rate does not degrade when leading singular values are clustered (Allen-Zhu & Li, 2017). This matters because gradient covariance spectra in deep networks often exhibit small eigengaps. Second, we show how to analyze methods of this type through the lens of dynamic regret under drift, providing guarantees that the tracked subspace—and thus the projected optimization—stays close to an ideal, stepwise comparator that always uses the instantaneous top-k directions. Together, these properties ground Ori's design in a theory that reflects observed phenomena: slow, persistent drift with potentially small eigengaps.

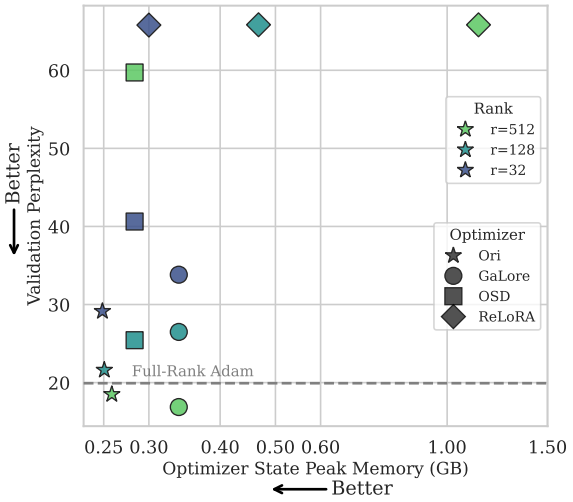


Figure 1: Trade-off between evaluation perplexity (Y-axis, lower is better) and peak optimizer state memory (X-axis, lower is better) for various low-rank optimizers in different ranks on the 1B LLaMA pre-training task. The ideal region is the bottom-left corner. **Ori ($\star$) consistently establishes a better Pareto frontier, achieving a superior performance-to-memory trade-off compared to state-of-the-art baselines like GaLore ($\circ$) and OSD ($\square$).**

Empirically, Ori narrowly tracks the evolving gradient subspace and converts this tracking fidelity into end-to-end wins where it matters most: very low ranks. Under tight rank budgets, each dimension in the projector is precious, and stale misalignment is especially costly. Empirical results show that Ori reduces the performance gap of state-of-the-art low-rank baselines to the full-rank Adam optimizer by up to 69.3% on the 1B LLaMA pre-training task, where the optimizer state rank is 128 in contrast to the model embedding dimension of 2048. And on Roberta-Base fine-tuning tasks, where the optimizer ranks are even lower (8 or 4), Ori achieves superior performance on 15 out of 16 scenarios. Moreover, Ori achieved this superior performance with one of the lowest computation budgets. See Figure 1 for a performance–memory overview on LLaMA-1B pre-training, where Ori consistently sits on the top-left frontier. Through ablation, we observe that (i) increasing the projector-update period degrades performance—quantifying the cost of staleness, and (ii) injecting a small carry-over (via a drift coefficient $\rho \approx 0.99$) improves accuracy—confirming the value of continuity and statistic transfer. Across pre-training and fine-tuning settings, Ori consistently improves the performance–memory trade-off over LoRA, GaLore, and OSD in the low-rank regime, thereby reducing the gap to full-rank Adam. The primary contributions of this paper are:

- Identify and quantify slow gradient subspace drift during LLM training and argue that exploiting its continuity—not merely reacting to it—matters for low-rank optimization.
- Propose Ori, a drift-aware low-rank optimizer that combines streaming k-PCA (Oja++) with moment transfer to achieve continuous, high-frequency projector updates at negligible cost.
- Provide theoretical support via dynamic-regret bounds under drift, yielding sublinear regret in the noiseless setting with mild dependence on the consecutive drift parameter.
- Present empirical results showing Ori's superior performance under stringent rank constraints, with ablations isolating the roles of update frequency and drift-aware continuity.

2 Background and Related Work

We refer the reader to Appendix B for a more detailed discussion of related work.

The Landscape: Low-Rank Optimization. Low-rank optimization in LLMs can be broadly divided into two categories. **Weight-subspace methods** adapt the model weights by constraining them to a low-rank manifold,

a paradigm popularized by LoRA and its many variants (Hu et al., 2022; Lialin et al., 2023; Zhang et al., 2023; Mo et al., 2025; Han et al., 2024). In contrast, Ori belongs to the second category, **gradient-subspace methods**, which operate on a low-rank projection of the gradients. This line of work was pioneered by GaLore (Zhao et al., 2024), which uses periodic Singular Value Decomposition (SVD) to identify and project gradients onto a low-rank basis. Many subsequent works have built upon this foundation by incorporating diverse strategies such as quantization (Zhang et al., 2024), structured sparsity (Muhamed et al., 2024), random projections (He et al., 2024; Wang et al., 2024), and importance sampling (Zhang et al., 2025; Liao et al., 2024). However, a common limitation underlines these approaches: by relying on static or infrequently updated projectors, they are inherently reactive to the evolving gradient landscape. This can lead to a "staleness penalty," where gradients are projected onto an outdated basis, losing critical information for optimization. Ori is designed to directly counteract this penalty by replacing infrequent recomputation with a continuous, online tracking mechanism.

The Mechanism: Online Subspace Tracking. The challenge of staleness in subspace projection has motivated online methods that update projectors frequently without relying on full SVD. Recent work explores **auxiliary optimization techniques** for dynamic subspace learning (Liang et al., 2024; Nguyen & Nguyen, 2024; Robert et al., 2024; Rajabi et al., 2025). Nguyen & Nguyen (2024) propose Subset-Norm and Subspace-Momentum, which decouples optimization by applying momentum in a low-dimensional subspace while performing SGD in the orthogonal complement. Robert et al. (2024) use block power iteration on a weighted gradient-EMA combination to adapt optimizer states to evolving subspaces. Rajabi et al. (2025) track subspaces on the Grassmannian manifold via geodesic updates, leveraging past estimates and error feedback for accuracy. Online Subspace Descent (OSD) (Liang et al., 2024) proposes a robust procedure that uses a secondary optimizer to perform gradient descent directly on the subspace manifold, initialized from the previous projector.

Ori adopts a fundamentally distinct strategy. Rather than relying on a secondary optimizer, it leverages the streaming k-PCA algorithm, **Oja++** (Allen-Zhu & Li, 2017), to update the projector. This distinction carries significant implications. First, gradient descent on the subspace manifold is inherently non-convex and complex, but Oja-style multiplicative updates come with strong theoretical assurances. Specifically, Oja++ achieves a provably "gap-free" convergence rate, offering greater robustness in static settings compared to gradient-based methods. Second, as formalized in our theoretical contributions, our Oja-based update exhibits desirable properties even in non-stationary, drifting environments, yielding strong dynamic regret bounds. Finally, the Oja++ update is computationally cheaper than gradient descent on the manifold, enabling Ori to update the projector more frequently. This higher frequency allows for improved tracking of the evolving gradient landscape.

The Theory: Connecting to Concept Drift. We frame the phenomenon of gradient subspace drift within the broader theoretical context of **concept drift** in online learning. Foundational work in this area establishes regret guarantees for learners adapting to a slowly shifting target distribution (Crammer et al., 2010). This theoretical lens is highly relevant to LLM training, where empirical evidence shows that gradients concentrate in a low-dimensional, slowly evolving subspace (Gur-Ari et al., 2018; Larsen et al., 2021). While the idea of a shifting target is well-established, our work is the first to formally analyze a low-rank LLM optimizer through the framework of dynamic regret under concept drift. This provides a principled theoretical foundation for Ori's performance and differentiates it from prior methods that focused primarily on algorithmic or empirical contributions.

3 Ori Optimizer

In this section, we further discuss the technical foundation of Ori. First, we establish and quantify the problem of gradient subspace drift, highlighting the limitations of existing approaches. Next, we introduce the core motivation of bootstrapping subspace updates, which forms the conceptual basis of our method. We then detail the specific online algorithm, a modified Oja++, used to implement this solution. Finally, we present the complete Ori algorithm, synthesizing these components into a cohesive and efficient optimization procedure.

3.1 The Phenomenon of Gradient Subspace Drift

Formalizing and Quantifying Drift. Low-rank optimizers that rely on periodic updates, such as GaLore (Zhao et al., 2024), implicitly assume that the principal gradient subspace remains relatively stable between updates. In this work, we propose to explicitly characterize this evolution. To do so, we adopt a formal metric from prior work on the changing geometry of optimization trajectories: the *canonical distance measure* (Gur-Ari et al., 2018). At any training step t, the top-k principal gradient directions $U_k^{(t)}$, which can be found via SVD of the gradient matrix $G_t = U^{(t)}\Sigma V^{(t)\top}$, span a subspace represented by the orthogonal projector $P_t = U_k^{(t)} U_k^{(t)\top}$. Our notion of gradient subspace overlap, based on the canonical distance between the gradient subspace at step t and a later step $t + \Delta t$, is then defined as

$$\text{overlap}\left(U_k^{(t)}, U_k^{(t+\Delta t)}\right) \equiv \frac{\text{Tr}(P_t P_{t+\Delta t})}{\sqrt{\text{Tr}(P_t)\text{Tr}(P_{t+\Delta t})}} = 1 - \frac{\|P_t - P_{t+\Delta t}\|_F^2}{2k} \propto \|P_t - P_{t+\Delta t}\|_F^2 . \tag{1}$$

This metric,[1] which ranges from 0 (orthogonal) to 1 (identical), is directly related to the geometric overlap between two subspaces. Specifically, $\text{Tr}(P_t P_{t+\Delta t})$ captures the k principal angles, $\{\theta_1, \dots \theta_k\}$, between subspaces P_t and $P_{t+\Delta t}$, in that

$$\text{Tr}(P_t P_{t+\Delta t}) = \sum_{i=1}^{k} \cos^2 \theta_i \leq k.$$

Thus, it provides a principled tool for empirically measuring the rate and nature of subspace evolution.

Empirical Evidence of Slow Drift. With the subspace overlap metric established, we empirically tested the stability assumption by measuring the overlap (with $\Delta t = 10$) throughout the pre-training of a 60M LLaMA model via the Adam optimizer on the C4 dataset (further experiment details can be found in Appendix D). The results, shown in Figure 2, provide clear evidence for a "slow drift" hypothesis. After an initial transient phase, the average gradient subspace overlap across all tested ranks ($k \in \{8, 32, 128\}$) stabilizes at a high value between 0.55 and 0.75. Crucially, this is significantly higher than the overlap observed for random subspaces of the same rank, confirming the existence of meaningful temporal structure. This indicates: (i) **The static subspace assumption is false**, as the overlap is consistently below 1.0, and the change over just 10 steps is substantial. (ii) **The evolution is not erratic**, as the overlap remains remarkably stable, suggesting the subspace moves with a predictable "velocity" rather than teleporting randomly. (iii) **The phenomenon is fundamental**, as the stable drift is present across a range of ranks, suggesting it is an intrinsic property of the optimization trajectory. This temporal coherence is precisely the property that an online tracking algorithm can be designed to exploit.

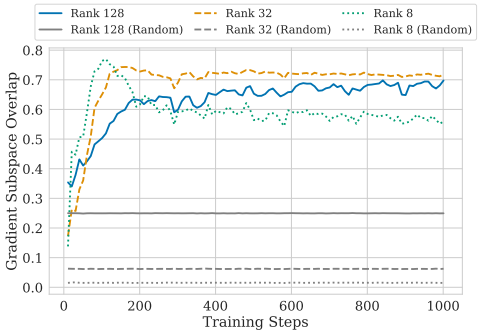


Figure 2: **Average gradient subspace overlap during pre-training.** We plot the overlap (Equation (1)) between subspaces at step t and $t + \Delta t$ (with $\Delta t = 10$), averaged over a 60M LLaMA model. The high, stable overlap, significantly above a random baseline, confirms that the subspace evolves gradually.

3.2 Online Subspace Tracking with a Bootstrapped Oja++

The empirical validation of slow, structured subspace drift motivates a paradigm shift from periodic, expensive *recomputation or adaption*, to continuous, lightweight *online tracking*, leading naturally to the use of streaming k-PCA algorithms, for which Oja's method provides a classic foundation (Oja, 1982). Its core is a simple multiplicative update that incrementally rotates the current basis Q_{t-1} with a new data sample x_t (in our case, derived from the gradient matrix G_t) to better align with new information in x_t, followed by a re-orthogonalization step (typically a QR decomposition) to maintain the basis properties. The fundamental update equation is:

$$Q_t \leftarrow \text{QR}\left((I + \eta_t x_t x_t^\top) Q_{t-1}\right), \tag{2}$$

where η_t is a learning rate (typically controlled by a cosine scheduler) and QR denotes the QR decomposition used for re-orthogonalization. While standard Oja's algorithm initializes the process with a random matrix, our key innovation is to explicitly leverage the temporal coherence of the subspace drift. We introduce a bootstrapping mechanism that uses the final projector from the previous update step, Q_{t-1}, as a warm start for the current one. This is controlled by a drift coefficient ρ, which interpolates between the previous subspace and a new random Gaussian matrix $R \sim \mathcal{N}(0, 1)$:

$$Q_{\text{init}} = \rho Q_{t-1} + (1 - \rho) R. \tag{3}$$

This bootstrapped matrix Q_{init} then serves as the next starting point for the update rule in Equation 2.

While Equation (2) and Equation (3) forms the conceptual basis of our approach, we specifically employ a more robust variant, Oja++ (Allen-Zhu & Li, 2017), for its crucial theoretical advantage of **gap-free convergence**. This means its convergence rate is independent of the eigengap (the difference between consecutive singular values of the gradient covariance matrix), a property that is vital for LLM training where such gaps can be small. The bootstrapping technique applies directly to Oja++ as well: the previous basis Q_{t-1} serves as the warm start for the multi-stage Oja++ update procedure. This combination of a theoretically sound, gap-free algorithm with a drift-aware bootstrapping strategy gives rise to the full form of our proposed optimizer, Ori, whose core projector update routine is detailed in Algorithm 1. For the algorithm pseudocode and its integration with common optimizers like Adam, we refer the readers to Appendix C.

[1] Equation (1) is well-defined because the projector P is idempotent ($P^2 = P$) and its trace equals its rank $\text{Tr}(P) = k$; we have $\|P_t - P_{t+\Delta t}\|_F^2 = \text{Tr}(P_t) + \text{Tr}(P_{t+\Delta t}) - 2\text{Tr}(P_t P_{t+\Delta t}) = 2k - 2\text{Tr}(P_t P_{t+\Delta t})$.

Algorithm 1 Ori Projector Update (Oja++)

Require: Current projector state $Q \in \mathbb{R}^{d\times r}$, full-rank gradient $G \in \mathbb{R}^{d\times n}$
Require: Oja++ hyperparameters H_{oja}: N_{updates}, ρ, N_{vecs}, cosine LR schedule for η

function UPDATE-PROJECTOR(Q, G, H_{oja})
  $Q_{\text{init}} \leftarrow \rho \cdot Q + (1-\rho)\cdot \text{Randn}(d, r)$ ▷ Bootstrap from last step's projector, Equation (3)
  $Q_{\text{new}} \leftarrow \text{zeros}(d, r)$
  $r_{\text{curr}} \leftarrow 0$
  for $k = 0 \to \log_2(r) - 1$ **do** ▷ Oja++ main update loop (Allen-Zhu & Li, 2017)
    $b \leftarrow r/2^{k+1}$ ▷ Oja++ updates one block at a time
    $r_{\text{next}} \leftarrow r_{\text{curr}} + b$
    $Q_{\text{new}}[:, r_{\text{curr}} : r_{\text{next}}] \leftarrow Q_{\text{init}}[:, r_{\text{curr}} : r_{\text{next}}]$
    $Q_{\text{slice}} \leftarrow Q_{\text{new}}[:, : r_{\text{next}}]$ ▷ Get the current active slice of the matrix
    $Q_{\text{slice}} \leftarrow \text{OJA-STEP}(Q_{\text{slice}}, G, N_{\text{updates}})$ ▷ Update the slice via Oja (Oja, 1982)
    $Q_{\text{new}}[:, : r_{\text{next}}] \leftarrow Q_{\text{slice}}$ ▷ Put the updated slice back
    $r_{\text{curr}} \leftarrow r_{\text{next}}$
  return Q_{new}

function OJA-STEP($Q_{\text{in}}, G, N_{\text{updates}}$)
  for $j = 1 \to N_{\text{updates}}$ **do** ▷ Oja main update loop (Oja, 1982)
    $\eta \leftarrow \text{GetCosineAnnealingLR}(j)$
    $G_{\text{sub}} \leftarrow \text{SampleColumns}(G, N_{\text{vecs}})$ ▷ Sample a subset of columns for efficiency
    $Q_{\text{in}} \leftarrow Q_{\text{in}} + \eta \cdot G_{\text{sub}}(G_{\text{sub}}^T Q_{\text{in}})$ ▷ Memory-efficient Oja update, Equation (2)
  return Q_{in}

4 Dynamic Regret Analysis with Oja's Algorithm

To gain further insight into the performance of these techniques for dynamic settings, we analyze streaming k-PCA algorithms in a nonstationary environment segmented into S steps, each comprising T sequential updates. Let $x_\zeta \in \mathbb{R}^d$ denote the streaming data at unified time index $\zeta \in [ST]$, where $\zeta = (s-1)T + t$ refers to update $t \in [T]$ inside step $s \in [S]$. We assume each x_ζ is drawn i.i.d from an unknown underlying distribution $\mathcal{D}$. Let the step-s optimal k-subspace be spanned by $V_s \in \mathbb{R}^{d\times k}$ with orthonormal columns. The dynamic comparator is a sequence of rank-k orthogonal projectors $\{P_s\}_{s=1}^S$ which is $P_s = V_s V_s^\top$ with *bounded drift* across consecutive steps:

$$\|P_s - P_{s+1}\|_F \;\le\; \lambda, \qquad s \in [S-1].$$

We now instantiate the analysis for multiplicative Oja updates on an explicit basis. The learner maintains an orthonormal basis $Q^{(\zeta)} \in \mathbb{R}^{d\times k}$ and updates with a noisy perturbation followed by reorthonormalization:

$$Q^{(\zeta+1)} \;=\; \text{QR}\Big(\big(I + \eta\,\hat{g}_\zeta\big)\,Q^{(\zeta)}\Big), \qquad \hat{g}_\zeta \;=\; \hat{x}_\zeta \hat{x}_\zeta^\top, \qquad \hat{x}_\zeta \;=\; x_\zeta + y_\zeta, \tag{4}$$

where y_ζ is an additive (possibly adversarial) noise term. At each ζ, the learner outputs a rank-k projector $P^{(\zeta)}$, an orthonormal basis $Q^{(\zeta)} \in \mathbb{R}^{d\times k}$ with $P^{(\zeta)} = Q^{(\zeta)}(Q^{(\zeta)})^\top$. As in (Marinov et al., 2018), we measure performance by the (variance) regret against the stepwise comparator:

$$R_S \;:=\; \sum_{\zeta=1}^{ST}\Big(x_\zeta^\top P_{s(\zeta)} x_\zeta - x_\zeta^\top P^{(\zeta)} x_\zeta\Big) \;=\; \sum_{\zeta=1}^{ST}\langle P_{s(\zeta)} - P^{(\zeta)}, x_\zeta x_\zeta^\top\rangle, \tag{5}$$

where $s(\zeta) = \lceil \zeta/T \rceil$. Since $\|x\|_2^2 = \|x - Px\|_2^2 + x^\top P x$ for any projector P, maximizing captured variance is equivalent to minimizing squared residual—so Equation (5) is also the residual regret.

Establishing this measure of performance leads us to the following guarantee for bounding the residual regret in this setting, the proof of which is found in Appendix A.3.

Theorem 1 (Noisy Oja under λ-drift). *For $\zeta \in [ST]$, let $Q^{(\zeta)}$ follow Equation (4) with $\eta = \sqrt{\frac{\frac{k}{2} + (S-1)\sqrt{k}\lambda}{kd\cdot ST}}$, let $E_{\text{tot}} := \sum_{\zeta=1}^{ST} \|E_\zeta\|_2$ where $E_\zeta := x_\zeta y_\zeta^\top + y_\zeta x_\zeta^\top + y_\zeta y_\zeta^\top$, and suppose $\|x_\zeta\|_2 \le 1$. Then we have*

$$\mathbb{E}[R_S] \;\le\; 2\sqrt{kd\cdot ST\Big(\tfrac{k}{2} + (S-1)\sqrt{k}\,\lambda\Big)} \;+\; 2k\,E_{\text{tot}}. \tag{6}$$

We note that in the noiseless setting, i.e., $E_{\text{tot}} = 0$, we achieve sublinear regret in terms of the total ST updates. Additionally, our results establish a natural balance in the dependence between the number of steps and sequential updates, with the dependence of the regret on S degrading further as λ increases.

Table 1: Comparison with full- and low-rank optimizers on pre-training various sizes of LLaMA models on C4 dataset. Validation perplexity is reported (lower ↓ is better). k is the rank for low-rank optimizers, and d_{model} is the model embedding dimension. We show the best scores in bold and second best ones with underlines. **Ori is on-par with the best baseline on moderately high rank settings, and achieves superior performance when the rank k is severely constrained.**

Method	**60M** ($d_{\text{model}} = 256$)			**130M** ($d_{\text{model}} = 768$)			**350M** ($d_{\text{model}} = 1024$)			**1B** ($d_{\text{model}} = 2048$)		
	$r = 128$	$r = 32$	$r = 8$	$r = 256$	$r = 64$	$r = 16$	$r = 256$	$r = 64$	$r = 16$	$r = 512$	$r = 128$	$r = 32$
Full-Rank Methods												
Adam		39.03			27.60			20.68			19.92	
Low-Rank Methods												
ReLoRA[Lialin et al., 2023]	144.48	103.96	103.99	85.53	85.54	85.54	72.59	72.72	72.69	65.82	65.83	65.79
GaLore[Zhao et al., 2024]	**39.38**	53.27	75.65	**27.70**	36.82	60.29	**21.26**	26.31	41.25	**16.87**	26.49	33.82
OSD[Liang et al., 2024]	49.64	52.35	74.08	49.57	37.56	54.05	28.28	37.13	50.08	59.72	25.43	40.63
Ori (Ours)	40.38	**48.84**	**65.03**	29.08	**35.22**	**43.61**	21.52	**26.11**	**39.24**	18.50	**21.61**	**29.14**
Training Tokens		0.44B			0.88B			2.5B			5.2B	

5 Experiments

In the experiments, we aim to demonstrate that Ori: (i) achieves superior performance against existing low-rank optimizers in both pre-training and fine-tuning, particularly under extremely low rank scenarios; (ii) improves memory and computational efficiency; and (iii) derives its benefits from its core design principles, as shown through ablation studies.

5.1 Baselines and Experimental Setup

We evaluate Ori against a consistent set of state-of-the-art optimizers, all built upon the Adam optimizer. For each baseline, we adopt the hyperparameter configurations recommended in their respective original publications and official codebases. Full experimental details are provided in Appendix D. Our baselines include: **Full-Rank Adam**, the standard optimizer serving as a practical upper bound on performance (using default $\beta_1 = 0.9, \beta_2 = 0.999$, and $\epsilon = 10^{-8}$); **Weight-Subspace Methods**, represented by **LoRA** (Hu et al., 2022) for fine-tuning (with a LoRA alpha of 32 and a dropout of 0.05) and its pre-training variant **ReLoRA** (Lialin et al., 2023) (with optimizer states reset every 1000 steps and no full-rank warmup); and **Gradient-Subspace Methods**. This category includes **GaLore** (Zhao et al., 2024), which employs periodic SVD (update frequency $T = 200$; pre-training scale $\alpha = 0.25$; fine-tuning scale $\alpha \in \{2, 4\}$), and **OSD (Online Subspace Descent)** (Liang et al., 2024), another state-of-the-art method that performs gradient descent directly on the subspace manifold (similarly using $T = 200$, pre-training $\alpha = 0.25$, and fine-tuning $\alpha \in \{2, 4\}$).

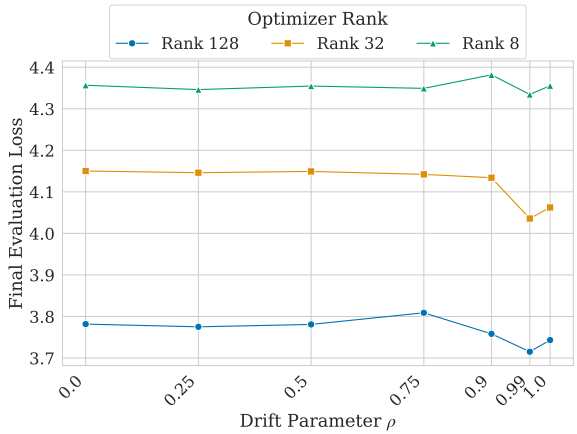
5.2 Main Results: LLaMA Pre-training

We evaluate Ori's pre-training performance against the baselines on the C4 dataset (Raffel et al., 2020) using LLaMA-architecture models of four sizes (60M, 130M, 350M, and 1B), following similar experimental settings as prior work (Lialin et al., 2023; Zhao et al., 2024). Departing from prior work that often tests a single rank, we systematically evaluate all low-rank methods across three distinct ranks for each model (e.g., $r \in \{512, 128, 32\}$ for the 1B model), enabling a rigorous assessment of optimizer performance under varying degrees of memory constraints. To ensure a fair comparison, model hyperparameters were held constant for each model size, while each optimizer's learning rate was tuned from $\{0.01, 0.005, 0.001, 0.0005, 0.0001\}$. We report validation perplexity after a fixed token budget, with complete experimental details in Appendix D.

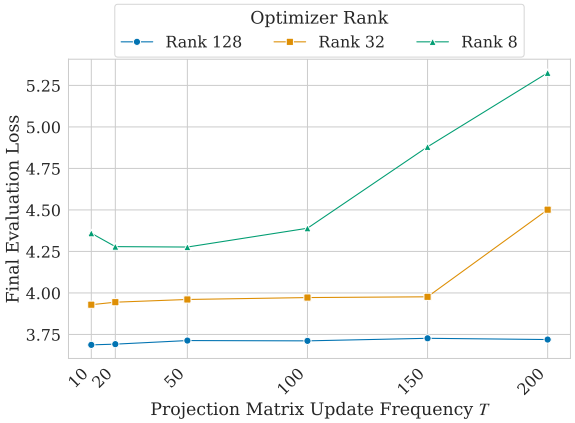
As shown in Table 1, Ori's performance is consistently strong, particularly in memory-constrained scenarios. At higher, less restrictive ranks (e.g., $r = 128$ for the 60M model, $r = 256$ for the 130M model), Ori performs on par with GaLore, establishing it as a highly competitive method in common low-rank settings. However, Ori's distinct advantage emerges when the rank r is severely constrained, where it demonstrates the most robust performance as its accuracy degrades the least under memory pressure. This trend, where the cost of a stale subspace projection becomes most apparent, is consistent across all model scales. For instance, when training the 130M model at an aggressively low rank of just 16, Ori achieves a validation perplexity of 43.61, substantially outperforming GaLore (60.29) and OSD (54.05). This result empirically validates our central hypothesis: actively tracking the gradient subspace drift is most critical when the optimizer has the fewest dimensions to work with. By excelling in these challenging low-rank regimes, Ori consistently narrows the performance gap to full-rank Adam, pushing the performance boundaries of memory-efficient optimization.

Table 2: Comparison with full- and low-rank optimizers on fine-tuning Roberta-Base on GLUE dataset. Average accuracy score and standard deviation is reported (lower ↓ is better) for each task, optimizer, and rank combination over three random seeds. We also report average accuracy over all tasks for each method. We show the best scores in bold and second best ones with underlines. **Ori consistently achieves superior performance on 15 out of 16 scenarios.**

Method	Rank	CoLA	STS-B	MRPC	RTE	SST2	MNLI	QNLI	QQP	Avg.
Adam	Full	61.61±0.46	90.88±0.05	92.52±0.12	78.45±1.50	93.88±0.67	87.41±0.05	92.63±0.37	89.21±0.16	85.82
LoRA (Hu et al., 2022)	8	41.85±1.66	87.07±0.25	87.95±2.40	67.03±2.40	93.12±0.20	85.14±0.10	90.62±0.24	84.66±0.14	79.68
GaLore (Zhao et al., 2024)	8	58.24±0.63	90.58±0.07	91.06±0.56	76.30±1.46	93.88±0.24	86.14±0.14	92.41±0.25	86.10±0.30	84.34
OSD (Liang et al., 2024)	8	60.39±0.31	**90.93±0.19**	91.85±0.50	77.36±1.46	94.61±0.30	86.62±0.10	92.29±0.16	87.66±0.78	85.16
Ori (Ours)	8	**63.58±0.93**	90.81±0.21	**92.93±0.23**	**79.42±1.08**	**94.92±0.54**	**87.52±0.05**	**92.55±0.07**	**89.27±0.04**	**86.30**
LoRA (Hu et al., 2022)	4	41.93±1.20	86.77±0.07	87.52±0.98	66.30±1.46	93.00±0.30	84.62±0.16	90.31±0.12	84.37±0.06	79.35
GaLore (Zhao et al., 2024)	4	60.47±2.16	90.67±0.23	91.72±0.48	77.01±2.33	94.19±0.24	87.24±0.15	92.24±0.14	87.96±0.10	85.19
OSD (Liang et al., 2024)	4	59.93±1.04	90.55±0.16	91.67±0.38	77.62±2.01	94.30±0.35	86.00±0.19	92.17±0.16	87.34±0.47	84.95
Ori (Ours)	4	**60.63±0.40**	**90.92±0.11**	**91.91±0.41**	**77.98±1.30**	**94.42±0.24**	**87.30±0.13**	**92.62±0.23**	**89.06±0.15**	**85.61**



(a) Impact of Drift Coefficient (ρ) on final evaluation loss. Lower ↓ is better.



(b) Impact of Update Frequency (T_{period}) on final evaluation loss. Lower ↓ is better.

(c) Oja++ vs. Standard Oja (Perplexity).

Algorithm	Rank (r=128)	Rank (r=32)	Rank (r=8)
Standard Oja	41.10	52,86	96.13
Oja++ (Ours)	**40.38**	**48.84**	**65.03**

Figure 3: Ablation Studies on the 60M LLaMA model pre-trained on C4 dataset for 0.44B tokens. (a) **Varying the drift coefficient ρ shows peak performance around $\rho = 0.99$**, validating the importance of transferring optimizer statistics. (b) **Performance degrades as the update frequency T_{period} increases**, demonstrating the cost of a stale projector. (c) **Oja++ consistently outperforms the standard Oja algorithm across all ranks.**

5.3 Finetuning

To evaluate Ori's effectiveness on adaptation tasks, we fine-tuned a RoBERTa-Base model on the GLUE benchmark (Wang et al., 2018). For each task, we report the mean and standard deviation of task-specific metrics over three runs with different random seeds. Further details on evaluation metrics are provided in Appendix D.

The fine-tuning results in Table 3 underscore Ori's dominant performance. Across both rank settings, Ori achieves the highest average score among all low-rank methods. At rank-8, Ori's average score of 86.30 not only surpasses GaLore (84.34) and LoRA (79.68) but also exceeds the full-rank Adam baseline (85.82). This strong performance continues at rank-4, where Ori (85.61) again leads all low-rank optimizers. Highlighting its remarkable consistency, Ori achieves the top score in 15 out of the 16 total task/rank combinations. Its particularly strong advantage on challenging tasks like CoLA (63.58 vs. GaLore's 58.24 at r=8) further validates its robustness and demonstrates that its drift-aware optimization is highly effective for both pre-training and adaptation tasks.

5.4 Ablation Study

To isolate the contributions of Ori's core design components, we conduct a series of ablation studies, performed via the 60M LLaMA model pre-trained on the C4 dataset (Section 5.2). Details about the ablation experiment settings can be found in Appendix D.6.

Impact of the Drift Coefficient (ρ) We first analyze the effect of the drift coefficient ρ, which controls how much of the previous step's projector is carried over to initialize the current step's projector. As shown in Figure 3a, we observe a clear and consistent trend across all ranks: a value of $\rho = 0.99$ yields the best performance. This has two key implications. First, the significant performance drop at $\rho = 0.0$ (a fully random restart) corroborates our central claim that transferring information about the previous subspace is crucial for effective optimization. Second, the slight performance dip at $\rho = 1.0$ (a direct carryover with no noise) suggests that injecting a small amount of randomness is a beneficial regularization strategy. Overall, we choose $\rho = 0.99$ for Ori in all pre-training and fine-tuning experiments.

Impact of Projector Update Frequency (T) We test our hypothesis that frequent, lightweight updates are superior to infrequent, heavy ones by varying the update period from 10 to 200 (which matches GaLore's recommended schedule (Zhao et al., 2024)). As shown in Figure 3b, more frequent updates consistently lead to lower loss (better performance). This phenomenon is especially prominent in the extremely low-rank setting ($r = 8$), where the cost of using a stale projector is most severe. This finding contrasts sharply with the ablation results from GaLore (Zhao et al., 2024), in which the authors reported no benefit from more frequent GaLore SVD updates. We hypothesize this is a unique advantage of Ori; because GaLore's periodic SVD acts as a "cold restart" that does not carry over subspace statistics, frequent updates may not be as beneficial. In contrast, since Ori is explicitly designed to track subspace drift, it can effectively leverage more frequent updates to maintain a higher-fidelity representation of the gradient landscape. For our main experiments, we set $T = 20$, as it strikes an effective balance between model performance and computational overhead.

Oja++ vs. Standard Oja Finally, to empirically justify choosing Oja++, we compare the performance of Oja++ against the standard Oja algorithm as the subroutine for updating the gradient projector. The results in Figure 3c are unequivocal: Oja++ consistently outperforms standard Oja across all tested ranks. This aligns with prior theoretical work showing Oja++ has a provably better convergence rate in static streaming k-PCA settings (Allen-Zhu & Li, 2017). Our results provide new empirical evidence that this advantage extends to the dynamic, non-stationary environment of LLM training, where Oja++'s superior tracking ability translates directly to better model performance.

5.5 Memory and Computation Time

We profile the optimizer state peak memory and wall-clock time during the pre-training experiments from Section 5.2. Details on the memory profiling procedure can be found in Appendix D.7. As shown in Table 3, Ori exhibits a superior efficiency profile among gradient-subspace methods. Full table on entire pre-training can be found in Appendix E.

Table 3: Memory and Wallclock Time Comparison of Pre-training 1B LLaMA on C4 dataset.

Optimizer rank	$r = 512$	$r = 128$	$r = 32$
Optimizer State Peak Memory (GB)			
Adam	2.4942	2.4942	2.4942
ReLoRA[Lialin et al., 2023]	1.1349	0.4670	0.3000
GaLore[Zhao et al., 2024]	0.3385	0.3385	0.3385
OSD[Liang et al., 2024]	0.2832	0.2832	0.2832
Ori (Ours)	**0.2583**	**0.2506**	**0.2487**
Average Update Time (s / Step)			
Adam	12.6015	12.6015	12.6015
ReLoRA[Lialin et al., 2023]	13.9454	**11.9193**	**11.5661**
GaLore[Zhao et al., 2024]	12.9699	12.9703	13.0401
OSD[Liang et al., 2024]	12.8438	12.8373	12.8594
Ori (Ours)	**12.7266**	12.7206	12.6705

Memory Usage: Ori consistently consumes the least optimizer peak memory across all ranks. For instance, at rank 32, Ori requires only 0.2487 GB, which is 12.1% less than OSD (0.2832 GB) and 26.5% less than GaLore (0.3385 GB). This confirms Ori's tangible memory savings.

Training Speed: Despite updating its subspace projector 10x more frequently ($T_{\text{Ori}} = 20$ vs. $T_{\text{GaLore/OSD}} = 200$), Ori is still the fastest among all gradient-projection methods. While the weight-subspace method ReLoRA is faster at lower ranks due to its different update mechanism, its worse model performance (see Table 1) highlights a clear trade-off. In contrast, Ori achieves its superior model performance without compromising on speed, making it a more effective and efficient choice overall. The lightweight nature of the Oja++ update allows Ori to be both more accurate and faster than its direct competitors, making it a better choice for efficient training.

6 Conclusion

We introduced Ori, a drift-aware low-rank optimizer that continuously tracks the gradient subspace with streaming k-PCA and preserves optimizer statistics across projector updates, mitigating staleness while keeping memory and compute low. Across theory (dynamic-regret under drift) and experiments, Ori consistently narrows the gap to full-rank training in the extremely low-rank regime while remaining faster and more memory-efficient than prior gradient-projection methods.

Reproducibility Statement

The reproducibility of our research is ensured through two key measures. Firstly, the algorithm proposed in this paper has been explicitly described in detail in the appendix, allowing for a clear understanding of our approach. Secondly, to facilitate direct replication of our work, we have provided the complete implementations as anonymously downloadable source code in the supplementary materials. These measures should enable other researchers to fully reproduce and validate our findings.

References

Zeyuan Allen-Zhu and Yuanzhi Li. First efficient convergence for streaming k-pca: a global, gap-free, and near-optimal rate. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 487–492. IEEE, 2017.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of llms under low-rank constraint? *arXiv preprint arXiv:2410.01623*, 2024.

Koby Crammer, Yishay Mansour, Eyal Even-Dar, and Jennifer Wortman Vaughan. Regret minimization with concept drift. In *COLT*, pp. 168–180, 2010.

Arijit Das. Natural galore: Accelerating galore for memory-efficient llm training and fine-tuning. *arXiv preprint arXiv:2410.16029*, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Robert Joseph George, David Pitt, Jiawei Zhao, Jean Kossaifi, Cheng Luo, Yuandong Tian, and Anima Anandkumar. Tensor-galore: Memory-efficient training via gradient tensor decomposition. 2025.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Andi Han, Jiaxiang Li, Wei Huang, Mingyi Hong, Akiko Takeda, Pratik Kumar Jawanpuria, and Bamdev Mishra. Sltrain: a sparse plus low rank approach for parameter and memory efficient pretraining. *Advances in Neural Information Processing Systems*, 37:118267–118295, 2024.

Yutong He, Pengrui Li, Yipeng Hu, Chuyan Chen, and Kun Yuan. Subspace optimization for large language models with convergence guarantees. *arXiv preprint arXiv:2410.11289*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Siddhartha Rao Kamalakara, Acyr Locatelli, Bharat Venkitesh, Jimmy Ba, Yarin Gal, and Aidan N Gomez. Exploring low rank training of deep neural networks. *arXiv preprint arXiv:2209.13569*, 2022.

Brett W Larsen, Stanislav Fort, Nic Becker, and Surya Ganguli. How many degrees of freedom do we need to train deep networks: a loss landscape perspective. *arXiv preprint arXiv:2107.05802*, 2021.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates. *arXiv preprint arXiv:2307.05695*, 2023.

Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. Memory-efficient llm training with online subspace descent. *Advances in Neural Information Processing Systems*, 37:64412–64432, 2024.

Xutao Liao, Shaohui Li, Yuhui Xu, Zhi Li, Yu Liu, and You He. Galore +: Boosting low-rank adaptation for llms with cross-head projection. *arXiv preprint arXiv:2412.19820*, 2024.

Sebastian Loeschcke, Mads Toftrup, Michael Kastoryano, Serge Belongie, and Vésteinn Snæbjarnarson. Loqt: Low-rank adapters for quantized pretraining. *Advances in Neural Information Processing Systems*, 37:115282–115308, 2024.

Xingtai Lv, Ning Ding, Kaiyan Zhang, Ermo Hua, Ganqu Cui, and Bowen Zhou. Scalable efficient training of large language models with low-dimensional projected attention. *arXiv preprint arXiv:2411.02063*, 2024.

Teodor Vanislavov Marinov, Poorya Mianjy, and Raman Arora. Streaming principal component analysis in noisy setting. In *International Conference on Machine Learning*, pp. 3413–3422. PMLR, 2018.

Zhanfeng Mo, Long-Kai Huang, and Sinno Jialin Pan. Parameter and memory efficient pretraining via low-rank riemannian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

Aashiq Muhamed, Oscar Li, David Woodruff, Mona Diab, and Virginia Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients. *arXiv preprint arXiv:2406.17660*, 2024.

Thien Hang Nguyen and Huy Le Nguyen. Lean and mean adaptive optimization via subset-norm and subspace-momentum with convergence guarantees. *arXiv preprint arXiv:2411.07120*, 2024.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3): 267–273, 1982.

Shikai Qiu, Andres Potapczynski, Marc Finzi, Micah Goldblum, and Andrew Gordon Wilson. Compute better spent: Replacing dense layers with structured matrices. *arXiv preprint arXiv:2406.06248*, 2024.

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

Sahar Rajabi and Sirisha Rambhatla. Subtrack your grad: Gradient subspace tracking for memory-efficient llm training and fine-tuning.

Sahar Rajabi, Nayeema Nonta, and Sirisha Rambhatla. Subtrack++: Gradient subspace tracking for scalable llm training. *arXiv preprint arXiv:2502.01586*, 2025.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. Ldadam: Adaptive optimization from low-dimensional gradient statistics. *arXiv preprint arXiv:2410.16103*, 2024.

DiJia Su, Andrew Gu, Jane Xu, Yuandong Tian, and Jiawei Zhao. Galore 2: Large-scale llm pre-training by gradient low-rank projection. *arXiv preprint arXiv:2504.20437*, 2025.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Ziyao Wang, Zheyu Shen, Yexiao He, Guoheng Sun, Hongyi Wang, Lingjuan Lyu, and Ang Li. Flora: Federated fine-tuning large language models with heterogeneous low-rank adaptations. *Advances in Neural Information Processing Systems*, 37:22513–22533, 2024.

Jinqi Xiao, Shen Sang, Tiancheng Zhi, Jing Liu, Qing Yan, Linjie Luo, and Bo Yuan. Coap: Memory-efficient training with correlation-aware gradient projection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 30116–30126, 2025.

Zi Yang, Ziyue Liu, Samridhi Choudhary, Xinfeng Xie, Cao Gao, Siegfried Kunzmann, and Zheng Zhang. Comera: Computing-and memory-efficient training via rank-adaptive tensor optimization. *Advances in Neural Information Processing Systems*, 37:77200–77225, 2024.

Haochen Zhang, Junze Yin, Guanchu Wang, Zirui Liu, Tianyi Zhang, Anshumali Shrivastava, Lin Yang, and Vladimir Braverman. I3s: Importance sampling subspace selection for low-rank optimization in llm pretraining. *arXiv preprint arXiv:2502.05790*, 2025.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Zhenyu Zhang, Ajay Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. Q-galore: Quantized galore with int4 projection and layer-adaptive low-rank gradients. *arXiv preprint arXiv:2407.08296*, 2024.

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

Kaiye Zhou, Shucheng Wang, and Jun Xu. Switchlora: Switched low-rank adaptation can learn full-rank information. *arXiv preprint arXiv:2406.06564*, 2024.

Philip Zmushko, Aleksandr Beznosikov, Martin Takáč, and Samuel Horváth. Frugal: Memory-efficient optimization by reducing state overhead for scalable training. *arXiv preprint arXiv:2411.07837*, 2024.

A PROOFS

A central tool is the identity relating projectors and principal angles.

Definition 1 (Principal angles). *For orthonormal $Q_1, Q_2 \in \mathbb{R}^{d\times k}$, let $\Theta(Q_1, Q_2) = diag(\theta_1, \dots, \theta_k)$ with $\cos\theta_i$ the singular values of $Q_1^\top Q_2$.*

Lemma 2. *(Distance Identity.) Let $P_1 = Q_1 Q_1^\top$ and $P_2 = Q_2 Q_2^\top$ be two rank-k orthogonal projection matrices, where $Q_1, Q_2 \in \mathbb{R}^{d\times k}$ are matrices with orthonormal columns. Then, we have the following:*

$$\|P_1 - P_2\|_F^2 = 2\,\|\sin\Theta(Q_1, Q_2)\|_F^2\,.$$

Proof. Provided in Appendix A.1. □

Theorem 2 converts the drift assumption into an angle bound:

$$\|\sin\Theta(V_s, V_{s-1})\|_F^2 \;\le\; \lambda^2/2. \tag{7}$$

Per-step potential drop. Define the misalignment potential $d_\zeta^2 := \left\|\sin\Theta(Q^{(\zeta)}, V_s)\right\|_F^2$. Analyses of noisy Oja yield a one-step inequality that links potential drop to the instantaneous regret:

$$\mathbb{E}\big[d_{\zeta+1}^2\big] - d_\zeta^2 \;\le\; -2\eta\,\langle P_s - P^{(\zeta)}, \hat{g}_\zeta\rangle \;+\; \mathcal{O}(\eta^2), \qquad P^{(\zeta)} := Q^{(\zeta)}(Q^{(\zeta)})^\top, \tag{8}$$

where the $\mathcal{O}(\eta^2)$ term represents the second-order error from the Taylor expansion of the potential function. As detailed in Appendix A.2, the magnitude of this geometrically-induced error is bounded by a constant proportional to $\eta^2\,\|\hat{g}_\zeta\|_F^2$. This quantity is, in turn, controlled by our assumption of bounded input data ($\|x_\zeta\|_2 \le 1$), which ensures the stability of the analysis.

Decomposing $\hat{g}_\zeta = x_\zeta x_\zeta^\top + E_\zeta$ with the error term $E_\zeta := x_\zeta y_\zeta^\top + y_\zeta x_\zeta^\top + y_\zeta y_\zeta^\top$, we get

$$\langle P_s - P^{(\zeta)}, \hat{g}_\zeta\rangle \;=\; \langle P_s - P^{(\zeta)}, x_\zeta x_\zeta^\top\rangle \;+\; \langle P_s - P^{(\zeta)}, E_\zeta\rangle.$$

Hölder's inequality and $\|P_s\|_* = \|P^{(\zeta)}\|_* = k$ imply $|\langle P_s - P^{(\zeta)}, E_\zeta\rangle| \le \|P_s - P^{(\zeta)}\|_*\,\|E_\zeta\|_2 \le 2k\,\|E_\zeta\|_2$. Rearranging Equation (8) and using $\mathcal{O}(\eta^2) \le 4kd\cdot\eta^2$ gives

$$\langle P_s - P^{(\zeta)}, x_\zeta x_\zeta^\top\rangle \;\le\; \frac{d_\zeta^2 - \mathbb{E}[d_{\zeta+1}^2]}{2\eta} + 2kd\eta + 2k\,\|E_\zeta\|_2. \tag{9}$$

Detailed proof for Equation (9) is shown at Appendix A.2. Summing Equation (9) over $\zeta \in \{(s-1)T+1, \dots, sT\}$ yields the per-step bound

$$R_s \;\le\; \frac{d_{(s-1)T+1}^2 - \mathbb{E}\big[d_{sT+1}^2\big]}{2\eta} \;+\; 2kd\eta T \;+\; 2k\,E_s, \qquad E_s := \sum_{\zeta=(s-1)T+1}^{sT} \|E_\zeta\|_2. \tag{10}$$

Linking steps via drift in angle space. By Theorem 2, we have that

$$d_s(V_s)^2 - d_s(V_{s-1})^2 \;=\; \tfrac{1}{2}\Big(\|P_Q - P_s\|_F^2 - \|P_Q - P_{s-1}\|_F^2\Big) \;\le\; \sqrt{k}\,\lambda,$$

where $P_Q := Q^{((s-1)T+1)}(Q^{((s-1)T+1)})^\top$. With any orthonormal initialization $Q^{(1)}$, we have $d_1(V_1)^2 \le k$.

A.1 PROOF OF LEMMA 2: DISTANCE IDENTITY

Proof.

$$\|P_1 - P_2\|_F^2 = Tr\left((P_1 - P_2)^\top (P_1 - P_2)\right) = Tr(P_1^2 - P_1P_2 - P_2P_1 + P_2^2)$$

Since P_1 and P_2 are orthogonal projectors, they are symmetric $P_i^\top = P_i$ and idempotent ($P_i^2 = P_i$). The trace of a rank-k projector is its rank, so $Tr(P_1) = Tr(P_2) = k$. The expression simplifies to:

$$||P_1 - P_2||_F^2 = \mathrm{Tr}(P_1) + \mathrm{Tr}(P_2) - 2\mathrm{Tr}(P_1P_2) = 2k - 2\mathrm{Tr}(P_1P_2).$$

The trace term can be rewritten as:

$$\mathrm{Tr}(P_1P_2) = \mathrm{Tr}(Q_1Q_1^\top Q_2Q_2^\top) = \mathrm{Tr}(Q_2^\top Q_1Q_1^\top Q_2) = ||Q_1^\top Q_2||_F^2.$$

The singular values of the matrix $Q_1^\top Q_2$ are the cosines of the principal angles, $\{\cos\theta_i\}_{i=1}^k$. The squared Frobenius norm is the sum of the squares of the singular values:

$$\left\|Q_1^\top Q_2\right\|_F^2 = \sum_{i=1}^k \cos^2\theta_i.$$

Substituting this back gives:

$$\|P_1 - P_2\|_F^2 = 2k - 2\sum_{i=1}^k \cos^2\theta_i = 2\sum_{i=1}^k (1-\cos^2\theta_i) = 2\sum_{i=1}^k \sin^2\theta_i = 2\left\|\sin\Theta(Q_1,Q_2)\right\|_F^2.$$

□

A.2 Detailed proof of equation 8 and equation 9(one-step potential drop for Oja)

Recall $P^{(\zeta)} = Q^{(\zeta)}(Q^{(\zeta)})^\top$, $P_s = V_s V_s^\top$, and define the misalignment potential

$$d_\zeta^2 \;:=\; \left\|\sin\Theta\big(Q^{(\zeta)}, V_s\big)\right\|_F^2 \;=\; \tfrac{1}{2}\left\|P^{(\zeta)} - P_s\right\|_F^2.$$

Denote $P := P^{(\zeta)}$ and $Z := \hat{g}_\zeta$ for brevity, where Z is symmetric. Oja's update reads

$$Q^{(\zeta+1)} \;=\; \mathrm{QR}\Big((I+\eta Z)\,Q^{(\zeta)}\Big), \qquad P^{(\zeta+1)} \;=\; Q^{(\zeta+1)}(Q^{(\zeta+1)})^\top. \tag{11}$$

Here is the proof of equation 8

$$\mathbb{E}\big[d_{\zeta+1}^2\big] - d_\zeta^2 \;\le\; -2\eta\,\langle P_s - P, Z\rangle \;+\; \mathcal{O}(\eta^2),$$

Proof. Consider the pre-retraction iterate $\widetilde{Q} = (I+\eta Z)Q^{(\zeta)}$ and its (non-idempotent) projector $\widetilde{P} := \widetilde{Q}\widetilde{Q}^\top$:

$$\widetilde{P} = (I+\eta Z)\,P^2\,(I+\eta Z) = (I+\eta Z)\,P\,(I+\eta Z) = P \;+\; \eta(ZP+PZ) \;+\; \eta^2 ZPZ. \tag{12}$$

Let $\mathsf{T}_P := \{\Delta = \Delta^\top :\; P\Delta P = 0,\; (I-P)\Delta(I-P) = 0\}$ denote the tangent space of the Grassmannian at P. Projecting the first-order term onto T_P gives the canonical tangent increment

$$\Delta_P \;:=\; \Pi_{\mathsf{T}_P}\big(ZP+PZ\big) \;=\; (I-P)\,Z\,P \;+\; P\,Z\,(I-P). \tag{13}$$

Standard properties of retractions (here, QR) imply that the reorthonormalized projector satisfies

$$P^{(\zeta+1)} \;=\; P \;+\; \eta\,\Delta_P \;+\; \mathcal{O}(\eta^2), \tag{14}$$

i.e., retraction preserves the first-order tangent component and pushes all curvature effects into $\mathcal{O}(\eta^2)$; which is a similar "one-step" analysis strategy of Appendix i.B in Allen-Zhu & Li (2017).

Define $\phi(P) = \frac{1}{2}\|P - P_s\|_F^2$, whose Euclidean gradient is $\nabla\phi(P) = P - P_s$. By expanding, we have

$$\begin{aligned} d_{\zeta+1}^2 - d_\zeta^2 = \phi\big(P^{(\zeta+1)}\big) - \phi(P) &= \langle P - P_s, P^{(\zeta+1)} - P\rangle \;+\; \frac{1}{2}\left\|P^{(\zeta+1)} - P\right\|_F^2 \\ &= \eta\,\langle P - P_s, \Delta_P\rangle \;+\; \frac{\eta^2}{2}\,\|\Delta_P\|_F^2. \end{aligned} \tag{15}$$

Write

$$\Delta_P \;=\; (I-P)ZP + PZ(I-P) \;=\; ZP + PZ - 2PZP.$$

A direct computation (e.g., in a block basis that diagonalizes P) yields the identity

$$\langle P - P_s, \Delta_P\rangle \;=\; -2\,\langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle, \qquad \Pi_{\mathsf{T}_P}(P_s - P) \;=\; (I-P)P_sP + PP_s(I-P). \tag{16}$$

Proof of equation 16 (sketch). Work in the orthonormal basis $[Q \;\; Q_\perp]$ with $P = \mathrm{diag}(I_k, 0)$ and decompose $Z = \left(\begin{smallmatrix} A & B^\top \\ B & C \end{smallmatrix}\right)$, $P_s = \left(\begin{smallmatrix} X & Y^\top \\ Y & Z_0 \end{smallmatrix}\right)$. Then $\Delta_P = \left(\begin{smallmatrix} 0 & B^\top \\ B & 0 \end{smallmatrix}\right)$ and $\Pi_{\mathsf{T}_P}(P_s - P) = \left(\begin{smallmatrix} 0 & Y^\top \\ Y & 0 \end{smallmatrix}\right)$. Hence $\langle P - P_s, \Delta_P\rangle = -2\,\mathrm{Tr}(Y^\top B) = -2\,\langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle$. □

Combining equation 15 and equation 16 we obtain

$$d_{\zeta+1}^2 - d_\zeta^2 \;=\; -2\eta\,\langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle \;+\; \mathcal{O}(\eta^2). \tag{17}$$

We now relate $\langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle$ to the full inner product $\langle P_s - P, Z\rangle$ that appears in the instantaneous regret. Observe that

$$\begin{aligned}\langle P_s - P, Z\rangle &= \langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle \;+\; \underbrace{\langle (I-P)(P_s-P)(I-P), Z\rangle}_{\geq 0 \text{ if } Z\succeq 0} \;+\; \underbrace{\langle P\,(P_s-P)\,P, Z\rangle}_{\leq 0 \text{ if } Z\succeq 0} \\ &= \langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle \;+\; \langle (I-P)P_s(I-P), Z\rangle \;-\; \langle P - PP_sP, Z\rangle. \end{aligned} \tag{18}$$

Since $0 \preceq P_s \preceq I$ and $Z \succeq 0$, both $(I-P)P_s(I-P)$ and $(P - PP_sP)$ are positive semidefinite, and hence the last line of equation 18 shows

$$\langle \Pi_{\mathsf{T}_P}(P_s - P), Z\rangle \;\leq\; \langle P_s - P, Z\rangle \qquad \text{whenever } Z \succeq 0. \tag{19}$$

Plugging equation 19 into equation 17 yields

$$d_{\zeta+1}^2 - d_\zeta^2 \;\leq\; -2\eta\,\langle P_s - P, Z\rangle \;+\; \mathcal{O}(\eta^2), \qquad (Z \succeq 0). \tag{20}$$

To be explicit about the control of the $\mathcal{O}(\eta^2)$ higher-order terms, we trace their origin. The term $P^{(\zeta+1)} - P$ in the Taylor expansion represents the full update step on the manifold. As established in equation 12 and the subsequent discussion on retractions, this step can be decomposed into a first-order tangent component $\eta\Delta_P$, and higher-order components. These higher-order components, which we collect into the $\mathcal{O}(\eta^2)$ term, are a direct consequence of two factors: (i) the explicit algebraic term $\eta^2 ZPZ$ that arises from the pre-retraction update, and (ii) the intrinsic curvature of the manifold, which means the path defined by the QR retraction deviates from the straight-line path of the tangent direction.

Recalling the algebraic second-order term, we have

$$\tilde{P} = P + \eta(ZP + PZ) + \eta^2 ZPZ, \quad \left\|\eta^2 ZPZ\right\|_F \leq \eta^2 \left\|Z\right\|_F \left\|P\right\|_F \left\|Z\right\|_F = \eta^2 \left\|P\right\|_F \left\|Z\right\|_F^2.$$

Since P is a rank-k orthogonal projector, its Frobenius norm is a constant: $\left\|P\right\|_F = \sqrt{Tr(P^\top P)} = \sqrt{Tr(P)} = \sqrt{k}$. Therefore:

$$\left\|\eta^2 ZPZ\right\|_F \leq \sqrt{k}\cdot\eta^2 \left\|Z\right\|_F^2.$$

For geometric second-order term, we need to relate $\|\Delta_P\|_F^2$ to $\|Z\|_F^2$. From equation 13, we have $\Delta_P = (I-P)ZP + PZ(I-P)$. Using the triangle inequality and the sub-multiplicativity of the Frobenius norm, we get:

$$\|\Delta_P\|_F \leq \|(I-P)ZP\|_F + \|PZ(I-P)\|_F \leq 2\|P\|_F\|I-P\|_F\|Z\|_F.$$

Since P is a rank-k projector and $I-P$ is a rank-$(d-k)$ projector, their norms are constants

$$\|P\|_F = \sqrt{k} \quad \text{and} \quad \|I-P\|_F = \sqrt{d-k},$$

respectively. This means $\|\Delta_P\|_F$ is bounded by a constant times $\|Z\|_F$. Consequently, the second-order geometric error $\eta^2\|\Delta_P\|_F^2$ is bounded by $(4kd - 4k^2)\eta^2\|Z\|_F^2$. In total, we have $\mathcal{O}(\eta^2) \leq 4kd\cdot\eta\,\|Z\|_F^2$. The norm of these higher-order terms is therefore fundamentally governed by the norm of the update matrix $Z = \hat{g}_\zeta$. Under $\|x_\zeta\|_2 \leq 1$ (so $\|Z\|_F = \left\|x_\zeta x_\zeta^\top\right\|_F = \|x_\zeta\|_2^2 \leq 1$) we have $\mathcal{O}(\eta^2) \leq 4kd\cdot\eta^2$. Finally, taking expectation over the randomness at time ζ (data and any internal noise) yields

$$\mathbb{E}\big[d_{\zeta+1}^2\big] - d_\zeta^2 \;\leq\; -2\eta\,\langle P_s - P, \hat{g}_\zeta\rangle \;+\; 4kd\cdot\eta^2,$$

which is Eq. equation 8. □

If $\hat{g}_\zeta = x_\zeta x_\zeta^\top + E_\zeta$ with an arbitrary symmetric perturbation E_ζ, the argument above applies verbatim to the PSD part $x_\zeta x_\zeta^\top$; the additional term satisfies

$$\left|\langle P_s - P, E_\zeta\rangle\right| \;\leq\; \|P_s - P\|_* \,\|E_\zeta\|_2 \;\leq\; 2k\,\|E_\zeta\|_2,$$

since $\|P_s\|_* = \|P\|_* = k$. This gives the noise contribution used in the main text. □

A.3 Proof of Theorem 1

Proof. Fix a step $s \in [S]$ and write $P^{(\zeta)} = Q^{(\zeta)}(Q^{(\zeta)})^\top$ and $P_s = V_sV_s^\top$. Define the misalignment potential $d_\zeta^2 := \left\|\sin\Theta(Q^{(\zeta)}, V_s)\right\|_F^2 = \frac{1}{2}\left\|P^{(\zeta)} - P_s\right\|_F^2$ (Lemma 2). By the one-step potential drop for Oja (proved in appendix A.2), for the noisy update $Q^{(\zeta+1)} = QR\big((I+\eta\,\hat{g}_\zeta)Q^{(\zeta)}\big)$ with $\hat{g}_\zeta = \hat{x}_\zeta\hat{x}_\zeta^\top$ and $\|x_\zeta\|_2 \leq 1$, we have

$$\mathbb{E}\big[d_{\zeta+1}^2\big] - d_\zeta^2 \;\leq\; -2\eta\,\langle P_s - P^{(\zeta)}, \hat{g}_\zeta\rangle \;+\; 4kd\eta^2. \tag{21}$$

Decompose the noisy rank-one matrix as $\hat{g}_\zeta = x_\zeta x_\zeta^\top + E_\zeta$, where $E_\zeta := x_\zeta y_\zeta^\top + y_\zeta x_\zeta^\top + y_\zeta y_\zeta^\top$. Using Hölder and $\|P_s\|_* = \|P^{(\zeta)}\|_* = k$,

$$\left|\langle P_s - P^{(\zeta)}, E_\zeta\rangle\right| \;\le\; \|P_s - P^{(\zeta)}\|_* \, \|E_\zeta\|_2 \;\le\; 2k \, \|E_\zeta\|_2.$$

Rearranging equation 21 gives the per-iteration regret bound

$$\langle P_s - P^{(\zeta)}, x_\zeta x_\zeta^\top\rangle \;\le\; \frac{d_\zeta^2 - \mathbb{E}[d_{\zeta+1}^2]}{2\eta} \;+\; 2kd\eta \;+\; 2k \, \|E_\zeta\|_2. \tag{22}$$

Within a step. Summing equation 22 over $\zeta = (s-1)T+1, \ldots, sT$ yields

$$R_s \;:=\; \sum_{\zeta=(s-1)T+1}^{sT} \langle P_s - P^{(\zeta)}, x_\zeta x_\zeta^\top\rangle \;\le\; \frac{d_{(s-1)T+1}^2 - \mathbb{E}[d_{sT+1}^2]}{2\eta} \;+\; 2kd\eta \;+\; 2k \, E_s, \tag{23}$$

where $E_s := \sum_{\zeta=(s-1)T+1}^{sT} \|E_\zeta\|_2$.

Across steps (drift linking). Summing equation 23 over $s = 1$ to S gives

$$\mathbb{E}[R_S] \;\le\; 2kd \cdot \eta T \;+\; \frac{1}{2\eta} \sum_{s=1}^{S} \Big(d_{(s-1)T+1}^2 - \mathbb{E}[d_{sT+1}^2] \Big) \;+\; 2k \, E_{\mathrm{tot}}, \qquad E_{\mathrm{tot}} := \sum_{s=1}^{S} E_s. \tag{24}$$

The terminal terms are non-positive and can be dropped: $-\sum_{s=1}^{S} \mathbb{E}[d_{sT+1}^2] \le 0$. It remains to upper bound $\sum_{s=1}^{S} d_{(s-1)T+1}^2$ where each $d_{(s-1)T+1}^2$ is measured against the *local* comparator V_s. Let $P_Q := Q^{((s-1)T+1)}(Q^{((s-1)T+1)})^\top$. By Lemma 2 and Cauchy-Schwarz inequality,

$$d_{(s-1)T+1}(V_s)^2 - d_{(s-1)T+1}(V_{s-1})^2 \;=\; \tfrac{1}{2}\big(\|P_Q - P_s\|_F^2 - \|P_Q - P_{s-1}\|_F^2\big) \;=\; \langle P_Q, P_{s-1} - P_s\rangle \;\le\; \sqrt{k}\,\lambda,$$

using $\|P_Q\|_F = \sqrt{k}$ and the drift assumption $\|P_s - P_{s-1}\|_F \le \lambda$. With the initialization $P^{(1)} = 0$ (hence $d_1(V_1)^2 = \frac{1}{2}\|P_1\|_F^2 = \frac{k}{2}$), we obtain

$$\sum_{s=1}^{S} d_{(s-1)T+1}^2 \;\le\; \frac{k}{2} \;+\; (S-1)\sqrt{k}\,\lambda. \tag{25}$$

Conclusion. Plugging equation 25 into equation 24 yields

$$\mathbb{E}[R_S] \;\le\; 2kd \cdot \eta ST \;+\; \frac{\frac{k}{2} + (S-1)\sqrt{k}\,\lambda}{2\eta} \;+\; 2k \, E_{\mathrm{tot}},$$

which is (7). Optimizing the right-hand side over $\eta > 0$ gives $\eta^\star = \sqrt{\frac{\frac{k}{2} + (S-1)\sqrt{k}\lambda}{2kd \cdot ST}}$ and

$$\mathbb{E}[R_S] \;\le\; \sqrt{2kdST\Big(\tfrac{k}{2} + (S-1)\sqrt{k}\,\lambda\Big)} \;+\; 2k \, E_{\mathrm{tot}},$$

which is (8). □

Remark (initialization). If one starts Oja from any orthonormal $Q^{(1)}$ (so $P^{(1)}$ is a rank-k projector), then $d_1(V_1)^2 \le k$ and the bound above holds with $\frac{k}{2}$ replaced by k; using $P^{(1)} = 0$ in the analysis yields the slightly tighter $\frac{k}{2}$ constant used in the theorem.

B More discussions on related work

B.1 Low-Rank Training Paradigms

Parameter weights subspace projection based descent. Recent research has explored learned subspace descent methods, where gradient updates are confined to a low-dimensional subspace of the model's weights that is itself learned during training. A straightforward instance is to parameterize each weight matrix as a product of low-rank factors (e.g. $W = BA$ with $B \in \mathbb{R}^{m\times r}, A \in \mathbb{R}^{r\times n}$) and optimize those factors via gradient descent (Hu et al., 2022; Kamalakara et al., 2022; Lv et al., 2024). While such plain low-rank factorization can significantly reduce parameters, it often falls short of full-rank performance (Zhao et al., 2024), prompting more adaptive approaches. ReLoRA (Lialin et al., 2023) tackles this by periodically absorbing the learned low-rank updates into the main

weights and introducing new factor matrices, gradually increasing the effective rank of W over training to recover expressiveness. SLTrain (Han et al., 2024) takes a complementary route by representing each weight as the sum of a trainable low-rank matrix and a sparse matrix ($W = BA + S$) and jointly optimizing both; this hybrid decomposition boosts model capacity while still confining updates to a learned low-dimensional (and partially sparse) subspace. Other methods learn to adjust the subspace itself: AdaLoRA (Zhang et al., 2023) dynamically adapts the rank of the low-rank adapters based on an importance metric (simulating SVD pruning of negligible directions), effectively learning the most productive subspace basis during training. Likewise, LORO (Mo et al., 2025) directly optimizes the model on a low-rank manifold (using Riemannian optimization), ensuring all gradient steps stay within a trainable low-dimensional parameter space. Beyond pure low-rank factors, structured subspace strategies have been proposed: BTT (Qiu et al., 2024) introduces a block tensor train factorization (via Kronecker-product structured weight matrices) with the factor components learned by gradient descent, and CoMERA (Yang et al., 2024) reshapes weight matrices into higher-order tensors that are decomposed into smaller learned tensors, enforcing a compact learned subspace of parameters. All of these approaches maintain that the projection basis (e.g. the low-rank factors or tensor components) is updated by training, allowing the subspace to evolve to fit the data.

In contrast, other methods restrict gradient descent to a fixed or partially fixed subspace of the weights, where the projection basis remains static (or changes only via non-gradient steps) for some duration. SwitchLoRA (Zhou et al., 2024) exemplifies this paradigm by using LoRA-style low-rank adapters but periodically switching their basis vectors (columns of B or rows of A) with new ones during training, instead of continuously refining the same basis. This schedule of refreshing the adapter subspace (inspired by ReLoRA's rank-growth idea) lets the model incrementally cover diverse low-dimensional subspaces without explicitly learning the basis via backpropagation. Similarly, LoQT (Loeschcke et al., 2024) employs a fixed-basis approach for efficiency: it initializes each weight with two low-rank factors $W \approx BA$, then freezes one factor (keeping it constant and even quantized to save memory) and only trains the other factor B. Periodically, the product BA is added into the full weight matrix W (and possibly B is reinitialized), but the subspace defined by the fixed factor A remains unchanged during those intervals. By operating in a predetermined low-rank subspace (or a sequence of static subspaces), SwitchLoRA and LoQT avoid the overhead of learning new basis directions, instead confining updates to coefficient adjustments within a fixed projector. This ensures that gradient descent explores a low-dimensional slice of the parameter space defined a priori (or piecewise fixed), in contrast to the above methods that co-learn the subspace itself during training.

Gradient subspace projection descent. Recent research has explored gradient subspace projection descent methods to reduce memory and computation by restricting updates to a low-dimensional subspace. GaLore (Zhao et al., 2024) pioneered this approach, using SVD to project each gradient onto a top-r singular vector subspace. This low-rank update dramatically lowers memory usage for gradients and optimizer states. Numerous extensions build on GaLore's foundation. Fira (Chen et al., 2024) adds an orthogonal complement term to reintroduce full-rank gradient information, closing the performance gap to full Adam without extra memory. Natural GaLore (Das, 2024) integrates natural gradient preconditioning into GaLore's update, while Q-GaLore (Zhang et al., 2024) combines low-rank projection with low-bit quantization for further memory savings. Other variants modify how the subspace is selected. COAP (Xiao et al., 2025) learns a correlation-aware projection matrix for improved efficiency, and GRASS (Muhamed et al., 2024) constructs sparse projection bases guided by gradient row norms to accelerate training. FRUGAL (Zmushko et al., 2024) performs an Adam-style update in the projected subspace and a simultaneous SGD step on the orthogonal complement, enabling updates across the full parameter space with minimal extra memory overhead. Tensor-GaLore (George et al., 2025) extends the low-rank projection principle to higher-order tensor parameters (beyond flat matrices), broadening applicability while preserving memory benefits.

Meanwhile, some works address the computational cost of maintaining the subspace or explore dynamic subspace strategies. Galore+ (Liao et al., 2024) reduces SVD overhead by performing the decomposition on only a randomly chosen component (e.g. one attention head) per update, significantly speeding up training at the cost of a slight approximation error. SubTrack-Grad (Rajabi & Rambhatla) and Online Subspace Descent (Liang et al., 2024) both dynamically update the projection basis: they solve auxiliary optimization problems (e.g. least-squares fits) to adjust the subspace so that it continues to approximate the current gradient well over time. In contrast, Flora (Wang et al., 2024) and GoLore (He et al., 2024) propose using fixed random projections of gradients, achieving memory savings with competitive performance to GaLore but without expensive SVD computations. I3S (Zhang et al., 2025) introduces an importance sampling subspace selection scheme, weighting basis directions by their singular values to adaptively refine GaLore's subspace and further improve convergence. Finally, the recent GaLore 2 framework (Su et al., 2025) addresses practical challenges like SVD cost and parallelization integration (e.g. with FSDP), and demonstrates the scalability of subspace-projected training by pre-training a 7B-parameter LLM with up to 500B tokens using significantly less memory.

B.2 Online Subspace Tracking

Online k-PCA subspace descent related method. Recent optimizers exploit low-dimensional subspaces to reduce memory cost in large-scale training via online k-PCA. For example, Subspace-Momentum confines the

momentum vector to a fixed k-dimensional subspace (e.g., spanned by top singular directions) while performing standard SGD in the orthogonal complement, substantially lowering optimizer-state memory and admitting high-probability convergence guarantees under standard conditions (Nguyen & Nguyen, 2024). The subspace can be chosen once or periodically refreshed via streaming singular vectors (e.g., computing top-k components every G steps) to adapt to evolving gradients (Nguyen & Nguyen, 2024). Similarly, Online Subspace Descent forgoes expensive SVD-based updates by continuously adapting the projection matrix through an online k-PCA algorithm (Liang et al., 2024). This approach provides a convergence guarantee for arbitrary subspace update rules in such low-rank optimizers and maintains an up-to-date basis with minimal overhead, eliminating full SVD computations during training (Liang et al., 2024). Applied to LLM pre-training, the online subspace optimizer achieves lower perplexity than prior low-rank methods and narrows the performance gap to full-rank training on benchmarks (Liang et al., 2024).

B.3 Concept Drift in Online Learning

Subspace learning. A growing body of work indicates that SGD dynamics concentrate in a low-dimensional, slowly evolving subspace, motivating algorithms that explicitly track or exploit subspace drift during training. Theory shows: (i) gradient updates lie within a tiny active subspace, suggesting that learning can be captured by a small set of dominant directions (Gur-Ari et al., 2018); (ii) online learning with concept drift admits regret guarantees against a slowly shifting comparator, formalizing performance when the target (or optimal subspace) moves over time (Crammer et al., 2010); and (iii) low-dimensional training succeeds once the subspace dimension exceeds a threshold tied to the loss landscape's Gaussian width, explaining why few degrees of freedom can suffice and how this threshold depends on initialization (Larsen et al., 2021). On the practical side for large-scale LLMs, recent methods instantiate drifting-subspace training by refreshing projection bases during optimization; among them, I3S (Zhang et al., 2025) samples singular directions by importance to diversify consecutive subspaces, alleviating "frozen" dominant bases while retaining convergence guarantees comparable to random-projection methods and delivering stronger pretraining perplexity in practice. Together, these results motivate modeling—and algorithmically tracking—gradient subspace drift when designing memory-efficient, very low-rank training procedures for LLMs.

C Ori algorithm details

C.1 AdamW with Ori

Algorithm 2 shows the full algorithm of an AdamW optimizer equipped with Ori low-rank gradient projector updates. We note that Ori is not intrinsically entangled with AdamW; rather similar to GaLore (Zhao et al., 2024), the Ori projector update subroutine can be easily integrated with other optimizers. We leave such exploration to future work.

C.2 Ori Hyperparameters and Recommended Search Strategy

In total, Ori consists of the following 7 important hyperparameters, listed roughly in the order of the most to least influential to model performance as we observed during method development.

- Scale γ: Similar to GaLore's scale α (Zhao et al., 2024), this parameter effectively scales the actual learning rate for the model layers for which Ori project the gradients. We suggest always search over this parameter as it highly correlates with model performance. Empirically, we found that one can set a higher γ value in the extremely low rank settings, while smaller γ tend to yield more stable training at higher ranks. We suggest $\gamma \in \{0.25, 0.5, 0.75, 1.0\}$ for large model pre-training, and $\gamma \in \{2.0, 4.0\}$ for very low-rank fine-tuning.
- Drift coefficient ρ: This parameter controls how much the last step's projector is carried over to warm-start the next step's projector Oja updates. A $\rho = 0.0$ means total random restart, and a $\rho = 1.0$ means total carry-over with no randomness injected. As established in Section 5.4, the $\rho = 0.99$ tend to be the universally good setting. We also found that $\rho \in \{0.8, 0.9\}$ can be good alternatives in a few scenarios.
- Projector update period T_{period}: This parameter controls the frequency of Ori's update. As established in Section 5.4, we suggest $T_{\text{period}} = 20$ as it strikes a perfect balance between model performance and computation budget.
- Ori warmup period T_{warmup}: This parameter controls for how many steps at the begging does Ori ignore T_{period} and update its projector consecutively. We recommand avoid setting too large a value as it may lead to overfitting. Empirically, we observe that $T_{\text{warmup}} = 20$ tend to be stable.
- Initial Oja learning rate η: The initial learning rate for the Oja update (Equation (2)), which gets decreased by a cosine annealing scheduler, and then gets reset at the next model update step. We observe that, as long as η is not too small, Ori yields stable learning and η's specific value do not have too much influence. We recommend setting $\eta = 1.0$ for all scenarios.

Algorithm 2 AdamW with Ori

Require: A layer weight matrix $W \in \mathbb{R}^{m \times n}$
Require: Learning rate η, AdamW decay rates β_1, β_2, rank k, scale factor γ
Require: Ori hyperparameters: warmup steps T_{warmup}, update period T_{period}, Oja++ settings H_{oja}

Initialize $Q_0 \in \mathbb{R}^{d \times r}$ with random values
$P_0 \leftarrow \text{QR}(Q_0)$
Initialize first-order moment $M_0 \in \mathbb{R}^{r \times n} \leftarrow 0$ ▷ Assuming $m \geq n$
Initialize second-order moment $V_0 \in \mathbb{R}^{r \times n} \leftarrow 0$
Initialize step $t \leftarrow 0$
repeat
  $G_t \leftarrow \nabla_{W_t} \mathcal{L}(W_t)$ ▷ Compute full-rank gradient
  if $t < T_{\text{warmup}}$ or $t \pmod{T_{\text{period}}} = 0$ **then** ▷ Periodically update the projector
    $Q_t \leftarrow \textsc{Update-Projector}(Q_{t-1}, G_t, H_{\text{oja}})$ ▷ Ori projector update, See Algorithm 1
    $P_t \leftarrow \text{QR}(Q_t)$
  else
    $Q_t \leftarrow Q_{t-1}$
    $P_t \leftarrow P_{t-1}$
  $R_t \leftarrow P_t^T G_t$ ▷ Project gradient into low-rank space
  ▷ **Update low-rank optimizer states with AdamW**
  $M_t \leftarrow \beta_1 M_{t-1} + (1 - \beta_1) R_t$
  $V_t \leftarrow \beta_2 V_{t-1} + (1 - \beta_2) R_t^2$
  $\hat{M}_t \leftarrow M_t / (1 - \beta_1^t)$
  $\hat{V}_t \leftarrow V_t / (1 - \beta_2^t)$
  $N_t \leftarrow \hat{M}_t / (\sqrt{\hat{V}_t} + \epsilon)$
  $\tilde{G}_t \leftarrow \gamma \cdot (P_t N_t)$ ▷ Project update back to original space and scale
  $W_{t+1} \leftarrow W_t - \eta \cdot \tilde{G}_t$ ▷ Update layer weights
  $t \leftarrow t + 1$
until convergence criteria met
return W_t

Table 4: Accuracy metrics we used for the Roberta-Base fine-tuning experiments of Table 2.

Task	CoLA	STS-B	MRPC	RTE	SST-2	MNLI	QNLI	QQP
Metric	Matthews Correlation	Pearson	F1	Accuracy	Accuracy	Accuracy	Accuracy	F1

Table 5: Ori hyperparameter used for the LLaMA pre-training results on the C4 dataset (Table 1).

Ori Hyperparameters	**60M**			**130M**			**350M**			**1B**		
	$r = 128$	$r = 32$	$r = 8$	$r = 256$	$r = 64$	$r = 16$	$r = 256$	$r = 64$	$r = 16$	$r = 512$	$r = 128$	$r = 32$
γ	0.5	0.5	1.0	0.25	0.25	0.25	0.75	0.75	0.5	0.5	0.5	0.5
ρ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
T_{period}	20	20	20	20	20	20	20	20	20	20	20	20
T_{warmup}	20	20	20	20	20	20	20	20	20	20	20	20
η	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
N_{update}	2	2	2	2	2	2	2	2	2	2	2	2
N_{vecs}	256	256	256	768	768	768	256	256	256	512	512	512

- Number of inner Oja-loop updates N_{update}: This parameter controls the number of inner Oja updates (see Algorithm 1). Note that since Ori uses Oja++ outer loop, the total number of inner Oja updates depends on the optimizer rank: $(\log_2(k) + 1)N_{\text{update}}$. In practice, we set $N_{\text{update}} = 2$ for all experiments.
- Number of gradient columns to sample for Oja update N_{vecs}: In practice, we observe that setting it in a way that scales with the model's embedding dimension is a good choice. We suggest $N_{\text{vecs}} \in \{d_{\text{model}}/2, d_{\text{model}}, d_{\text{model}} \times 2\}$.

D Experiment Details

D.1 Dataset and Training Details

We use the C4 dataset for the LLaMA model pre-training task. C4 dataset is a colossal, cleaned version of Common Crawl's web crawl corpus, mainly intended for pre-training language models and word representations (Raffel et al., 2020). Similar to prior work's experiment settings, we train the model without data repetition over a sufficiently large amount of data. We report the final validation perplexity (the lower the better) for the main experiment table (Table 1).

For all the pre-trianing experiments, we adjust the gradient accumulation steps accordingly, depending on the available GPUs, so that the effective total batch size for all runs are 512 (for instance, an actual batch size of 64 with 8 gradient accumulation steps result in the effective batch size of 512). We set the LLaMA model max length to 256, adopted a cosine learning rate scheduler with a 0.1 minimum learning rate ratio, and trained with bfloat16 data type.

We use the GLUE dataset for the Roberta-Base model fine-tuning task. GLUE is a benchmark for evaluating the performance of NLP models on a variety of tasks, including question answering, sentiment analysis, and textual entailment (Wang et al., 2018). We report the final accuracy score (the higher the better) for the main experiment table (Table 2). The metric of choice for each GLUE task is listed in Table 4. Notably, different from many of the prior optimization work, we run each task + optimizer + rank combination for three runs with different random seeds to improve statistical confidence level. Thus, we were able to report the error bar (standard deviation) in the table.

For all the fine-tuning experiments, similar to prior work's experiment settings, we train all models for a total of 30 epochs, with 1 gradient accumulation step. We set the model max length to 512, and adopted a linear learning rate scheduler.

D.2 Compute Resources

We used a variety of compute resources with different GPU hardware, including RTX 4090, RTX 4070 Ti Super, RTX A500, RTX A6000, 40GB A100, 80GB A100, H100, MI100, and MI210. For the large model pre-training, such as the 350M and 1B LLaMA model pre-training experiments, we trained the model on nodes equipped with 4 H100 or 4 MI210.

D.3 Ori Hyperparameters

We report the detailed Ori hyperparameters used in each experiment setting for reproducibility. Table 5 lists all hyperparameters used for the LLaMA-C4 pre-trianing tasks, and Table 6 lists all hyperparameters used for the Roberta-Base-GLUE fine-tuning tasks.

Table 6: Ori hyperparameter used for the Roberta-Base fine-tuning results on the GLUE dataset

Ori Hyperparameters	GLUE all tasks
γ	4.0
ρ	0.99
T_{period}	20
T_{warmup}	20
η	1.0
N_{update}	2
N_{vecs}	768

D.4 Baseline hyperparameters

Baseline hyperparameters are discussed in the main experiment section (see Section 5.1).

D.5 Visualizing subspace drift

We reused the setting of 60M LLaMA model pre-training on C4 dataset to visualize an actual gradient subspace drift as the model is being trained in real life, resulting in Figure 2.

For this experiment, we trained the 60M LLaMA model with the Adam optimizer for a reduced 1000 training steps, totalling 100M training tokens. We set $\Delta t = 10$, $k \in \{128, 32, 8\}$, and computed overlap $\left(U_k^{(t)}, U_k^{(t+\Delta t)}\right)$ (Equation (1)) for every layers in the model. We then report and plot the mean overlap, averaged over all layers, through the training trajectories.

To further demonstrate that the overlap values shown in Figure 2 are not ad-hoc and it indeed signify that the gradient subspace has non-trivial overlapping, we simulated three random baselines, one for each rank $k \in \{128, 32, 8\}$. The random baseline is conducted by sampling a random matrix with the same shape and Frobenius norm as the true gradient matrix, for each layer at each step. We then pretend these random matrices were some "gradients", and record their overlap values with the earlier random matrices.

D.6 Ablation study details

We reused the setting of 60M LLaMA model pre-training on C4 dataset to perform the ablation study. The ori optimizer in the ablation studies used the same hyperparameters as detailed in Table 5.

For the ablation of Oja++ vs. Standard Oja, recall that the default choice of Oja++ algorithm consists of an outer loop and an inner loop Algorithm 1, and the total number of Oja inner updates in the default setting is $(\log_2(k) + 1)N_{\text{update}}$. Thus, to ensure fairness, we set this number to the Oja-variant of Ori. That is, since we use $N_{\text{update}} = 2$ for all experiments, in this ablation study, we set the number of Oja baseline udpates to 16 when $k = 128$, 12 when $k = 32$, and finally 8 when $k = 8$.

D.7 Measuring optimizer state peak memory

Finally, we report our procedure for measuring the memory footprint of the optimizers evaluated in our experiments. Our goal is to record the peak memory usage, incurred solely by the said optimizer's update step. To this end, we take advantage of PyTorch's memory recording functionalities.

The below code listing shows the pseudocode for recording peack memory usage incurred solely by the optimizer's `step()` operations. We use this procedure to obtain all the results in Tables 3 and 7.

```
import torch

# Model and optimizer setup
model = ...
optimizer = ...

# Main training loop
for batch in enumerate(dataloader):

    # Forward and backward pass
    loss = model(**batch).loss
    loss.backward()

```

Table 7: Memory and Wallclock Time Comparison of pre-training various sized LLaMa model on the C4 dataset.

Method	60M			130M			350M			1B		
	$r = 128$	$r = 32$	$r = 8$	$r = 256$	$r = 64$	$r = 16$	$r = 256$	$r = 64$	$r = 16$	$r = 512$	$r = 128$	$r = 32$
Optimizer Peak Memory (GB)												
Adam	0.1082	–	–	0.2498	–	–	0.6854	–	–	2.4942	–	–
ReLoRA[Lialin et al., 2023]	0.0797	0.0657	0.0622	0.1751	0.1125	0.0968	0.3450	0.1779	0.1361	1.1349	0.4670	0.3000
GaLore[Zhao et al., 2024]	0.0670	0.0670	0.0670	0.1049	0.1049	0.1049	0.1458	0.1458	0.1458	0.3385	0.3385	0.3385
OSD[Liang et al., 2024]	0.0635	0.0635	0.0635	0.0970	0.0970	0.0970	0.1318	0.1318	0.1318	0.2832	0.2832	0.2832
Ori (Ours)	**0.0623**	**0.0619**	**0.0618**	**0.0948**	**0.0932**	**0.0927**	**0.1256**	**0.1237**	**0.1232**	**0.2583**	**0.2506**	**0.2487**
Average Update Time (s / Step)												
Adam	0.5198	–	–	1.0309	–	–	2.655	–	–	12.6015	–	–
ReLoRA[Lialin et al., 2023]	0.6622	0.6395	0.6413	1.3324	1.2824	1.2626	3.3348	3.1886	3.1423	13.9454	**11.9193**	**11.5661**
GaLore[Zhao et al., 2024]	0.5454	0.5262	0.5270	1.0617	1.0575	1.0601	2.7733	2.7702	2.7754	12.9699	12.9703	13.0401
OSD[Liang et al., 2024]	0.5664	0.5614	0.5645	1.0917	1.0866	1.1003	2.7717	2.8100	2.8067	12.8438	12.8373	12.8594
Ori (Ours)	**0.5313**	**0.5260**	**0.5229**	**1.0490**	**1.0473**	**1.0383**	**2.7260**	**2.7018**	**2.6894**	**12.7266**	12.7206	12.6705
Total Update Time (s)												
Adam	520	–	–	1032	–	–	2658	–	–	12614	–	–
ReLoRA[Lialin et al., 2023]	663	640	641	1334	1284	1264	3338	3192	3145	13931	**11931**	**11578**
GaLore[Zhao et al., 2024]	546	527	528	1063	1059	1061	2776	2772	2778	13053	12984	12983
OSD[Liang et al., 2024]	567	562	565	1093	1088	1101	2774	2812	2809	12857	12850	12872
Ori (Ours)	**532**	**526**	**523**	**1049**	**1047**	**1038**	**2728**	**2704**	**2692**	**12740**	12733	12683

```
    # Record the current memory usage after model backward
    mem_after_backward = torch.cuda.memory_allocated()

    # Reset the max memory counter
    torch.cuda.reset_peak_memory_stats()

    # Optimizer step
    optimizer.step()

    # Read the optimizer-incurred max memory
    # The difference with mem_after_backward is the maximum additional memory
    # caused by optimizer.step()
    peak_mem_during_step = torch.cuda.max_memory_allocated()
    optim_peak_mem = peak_mem_during_step - mem_after_backward
```

Listing 1: Python code for profiling optimizer peak memory.

E ADDITIONAL EXPERIMENT RESULTS

Table 7 presents the full results of memory and Wallclock time comparison of all the optimizer methods on the LLaMA model pre-training task on the C4 dataset. As shown by the table, our method Ori achieves superior memory and computation time profile almost across the broad, demonstrating its general efficiency under various settings.