SubTrack++ : Gradient Subspace Tracking for Scalable LLM Training

Sahar Rajabi, Nayeema Nonta, Sirisha Rambhatla
Critical ML, Department of Management Science and Engineering, University of Waterloo
{srajabi, nnonta, srambhatla}@uwaterloo.ca

Abstract

Training large language models (LLMs) is highly resource-intensive due to their massive number of parameters and the overhead of optimizer states. While recent work has aimed to reduce memory consumption, such efforts often entail trade-offs among memory efficiency, training time, and model performance. Yet, true democratization of LLMs requires simultaneous progress across all three dimensions. To this end, we propose SubTrack++ that leverages Grassmannian gradient subspace tracking combined with projection-aware optimizers, enabling Adam's internal statistics to adapt to subspace changes. Additionally, employing recovery scaling, a technique that restores information lost through low-rank projections, further enhances model performance. Our method demonstrates SOTA convergence by exploiting Grassmannian geometry, **reducing training wall-time by up to 65%** compared to the best performing baseline, LDAdam, while preserving the reduced memory footprint. Code is at `https://github.com/criticalml-uw/SubTrack`.

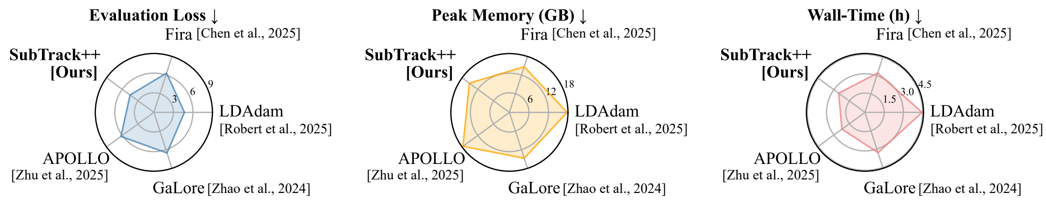
1 Introduction

LLMs have demonstrated state-of-the-art performance across a wide range of tasks and are rapidly growing in popularity. However, training and fine-tuning these models require significant resources, including extensive hardware and time, which limits their practicality for many applications and increases their environmental impact and carbon footprint. [Zhao et al., 2024, Jaiswal et al., 2024, Muhamed et al., 2024, Miles et al., 2024, Modoranu et al., 2024, Hao et al., 2024, Li et al., 2024].

Several techniques have been proposed to mitigate memory bottlenecks[Chen et al., 2016, Rajbhandari et al., 2020]. LoRA [Hu et al., 2021] and other low-rank adaptation methods [Dettmers et al., 2024, Hu et al., 2021, Yaras et al., 2024, Lialin et al., 2023, Renduchintala et al., 2024, Xia et al., 2024, Miles et al., 2024] have gained popularity by optimizing a reduced set of parameters. Such approaches often assume a low-rank parameter space, which can lead to suboptimal performance. In addition, methods like BAdam [Luo et al., 2024] and Block-LLM [Ramesh et al., 2024], utilize block coordinate descent to optimize parameter subsets, achieving memory savings at the cost of reduced accuracy.

However, memory requirements extend beyond trainable parameters, with a significant portion consumed by the optimizer's states [Zhao et al., 2024]. Recent efforts have focused on reducing this space while targeting full parameter training [Li et al., 2023, Anil et al., 2019, Lv et al., 2024, Dettmers et al., 2022, Zhang et al., 2024, Modoranu et al., 2024, Zhao et al., 2024, Muhamed et al., 2024]. Leveraging the low-dimensional nature of gradients during gradient descent [Gur-Ari et al., 2018, Schneider et al., 2024, Yaras et al., 2023] , GaLore [Zhao et al., 2024] reduces memory usage by projecting gradients into a low-rank subspace and periodically updating this approximation via singular value decomposition (SVD). While SVD offers optimal low-rank approximation [Robert et al., 2025], it can pose several challenges. First, it is computationally intensive, and alternatives which use random projections [Zhu et al., 2025] or approximation methods to estimate dominant singular values [Robert et al., 2025, Liang et al., 2024], match or outperform SVD in practice.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) (b) (c)

Figure 1: We compare baselines on pre-training a 1B-parameter model. (a) SubTrack++ achieves the lowest evaluation loss across all methods. (b) Its peak memory usage is significantly lower than APOLLO and LDAdam, and on par with GaLore and Fira. (c) In terms of wall-time, SubTrack++ incurs minimal overhead relative to APOLLO and is markedly faster than GaLore, Fira, and LDAdam. Overall, SubTrack++ outperforms all baselines in evaluation loss while matching or exceeding them in memory and runtime efficiency.

Moreover, SVD is sensitive to noise [Vaswani et al., 2018, He et al., 2025] and tends to degrade in late training stages when gradients are small, often hindering convergence [He et al., 2025].

Geometry-based methods have shown strong performance in various machine learning applications [Zhang et al., 2018, Balzano et al., 2011, He et al., 2011, Blocker et al., 2023]; Grassmannian is the manifold of all subspaces of dimensions r in a space of dimensions d, and using Grassmannian for subspace tracking has led to structurally embedded information, lower computational complexity, and improved performance [Zhang et al., 2018, Balzano et al., 2011, He et al., 2011, Blocker et al., 2023, Chakraborty et al., 2017, Zhang and Balzano, 2016]. This line of work has demonstrated robustness and efficiency in high-dimensional, noisy environments [Zhang et al., 2018, Balzano et al., 2011, He et al., 2011, Balzano et al., 2018, Chakraborty et al., 2017]. Their natural robustness against perturbations and strong theoretical guarantees make them particularly well-suited to tracking the evolving gradient subspaces encountered in LLM training.

To this end, we employ subspace tracking on Grassmannian geodesics (Figure 2), to develop a geometry-based, time- and memory-efficient training method. This way, instead of reconstructing low-rank approximations via expensive SVD, we can efficiently leverage previously computed subspaces and the estimation error to better adjust the projection. We also incorporate subspace shifts into Adam's first and second momentum update rules, ensuring proper alignment with coordinate changes using a Projection-Aware Optimizer [Robert et al., 2025]. Additionally, we recover and scale (Recovery Scaling) the gradient information lost during low-rank projection by utilizing the scaling information of the low-rank, state-full optimizer Chen et al. [2025], Zhu et al. [2025].

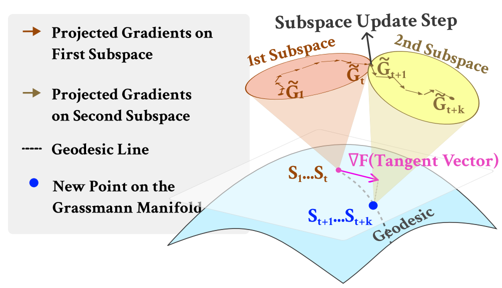


Figure 2: Visualization of Grassmannian subspace tracking: Between subspace updates, gradients are projected onto a fixed subspace. The tangent vector ∇F is computed via the derivative of a loss function, measuring the subspace estimation error. The subspace is then updated by moving along the corresponding geodesic, determined by ∇F to minimize estimation error.

To summarize, SubTrack++ is a projection-aware geometry-based approach that supports full-parameter training and incorporates recovery scaling, offering superior time efficiency compared to SVD or PowerSGD methods (e.g., GaLore [Zhao et al., 2024, Chen et al., 2025, Robert et al., 2025], while maintaining GaLore's memory footprint. It also outperforms online PCA subspace tracking methods [Liang et al., 2024] and achieves SOTA convergence and evaluation loss across all strong baselines; with comparison provided in Figure 1 and Figure 8.

2 SubTrack++

Challenges of Low-Rank Optimization. Projecting gradients into a low-rank subspace reduces memory footprint and enables scalable LLM training, but it introduces important trade-offs. First, gradient subspaces require adaptive tracking; while SVD-based methods can capture these shifts [Zhao et al., 2024, Chen et al., 2025], they are computationally expensive. To address this, recent work has explored cheaper approximations and random projections [Robert et al., 2025, Liang et al., 2024, Zhu et al., 2025, He et al., 2025]. Furthermore, optimizers like Adam assume a fixed coordinate system, and subspace changes must be reflected in their internal states for a consistent momentum updates. Finally, low-rank projections inherently discard some gradient components, that recovering and utilizing these discarded signals can boost performance [Chen et al., 2025, Zhu et al., 2025].

Overview. SubTrack++ , a memory- and time-efficient method, embeds geometric insights into low-rank optimization, and improves efficiency and performance via three core components: 1) Grassmannian subspace tracking that refines projections via estimation error and subspace history; 2) projection-aware optimizer which adapts Adam's state to account for evolving subspaces; and 3) recovery scaling that restores lost information by scaling discarded gradient components.

Subspace Tracking. We frame subspace estimation as selecting a point on the Grassmannian, the manifold of all d-dimensional subspaces in an n-dimensional space [Bendokat et al., 2024]. This perspective offers three key benefits: 1) It refines the subspace using prior subspaces and estimation error, avoiding full reinitialization as done in Zhao et al. [2024], Chen et al. [2025], He et al. [2025]. 2) Updates rely on lightweight algebraic operations. 3) Controlled subspace shifts improve robustness against noise and abrupt changes. Similar to GaLore [Zhao et al., 2024], our proof in Theorem 3.2 shows that applying the projection to linear layers of LLMs preserves convergence while significantly reducing optimizer state memory. The initial subspace is computed using SVD as shown in (1). $G_0 \in \mathbb{R}^{m\times n}$ is the gradient matrix at step 0; U, S, and V are its SVD components, and r is the specified rank.

$$G_0 = USV^\top \approx \sum_{i=1}^{r} s_i u_i v_i^\top \tag{1}$$

At each step, gradients are projected onto the subspace of left singular vectors if $m \leq n$, and right singular vectors otherwise; optimizing memory usage [Zhao et al., 2024]. We assume $m \leq n$ without loss of generality, so the subspace is represented by $S_t \in \mathbb{R}^{m\times r}$ ($S_0 = [u_1, ..., u_r]$), an orthonormal basis spanning the top-r directions. The gradient is projected as $\widetilde{G}_t = S_t^\top G_t \in \mathbb{R}^{r\times n}$, and the optimizer operates in this reduced space, significantly lowering memory and state overhead.

To account for subspace drift, the core subspace is updated every k steps (the subspace update interval) by minimizing the cost function (2), which measures its Euclidean distance to the current gradient.

$$F(S_t) = \min_A \|S_t A - G_t\|_F^2, \tag{2}$$

where A is the solution to the least squares problem. The derivative of (2) with respect to S_t is given in (3), and the residual $R = G_t - S_t A$ lies in the orthogonal complement of S_t. To update the subspace, we calculate the tangent vector ∇F on the Grassmannian, as shown in (4) based on Edelman et al. [1998], where the second equality holds because R is orthogonal to $S_t S_t^\top$.

$$\frac{\partial F}{\partial S_t} = 2(S_t A - G_t)A^\top = -2RA^\top \tag{3}$$

$$\nabla F = (I - S_t S_t^\top)\frac{\partial F}{\partial S_t} = \frac{\partial F}{\partial S_t} = -2RA^\top \approx \widehat{U}_F \widehat{\Sigma}_F \widehat{V}_F^\top \tag{4}$$

For optimizing the loss function (2), the subspace should be moved in the direction of $-\nabla F$ to reduce the estimation error. However, to control subspace changes, SubTrack++ computes a rank-1 approximation of ∇F, determined by its largest singular value and the corresponding singular vector obtained from its SVD, represented as $\widehat{U}_F \widehat{\Sigma}_F \widehat{V}_F^\top$. This approximation is then used for subspace update. As shown by Edelman et al. [1998], Bendokat et al. [2024], we can move along a Grassmannian geodesic guided by rank-1 estimation of $-\nabla F$, with a step-size η, as presented in (5).

$$S_{t+1}(\eta) = \begin{pmatrix} S_t \widehat{V}_F & \widehat{U}_F \end{pmatrix} \begin{pmatrix} \cos \widehat{\Sigma}_F \eta \\ -\sin \widehat{\Sigma}_F \eta \end{pmatrix} \widehat{V}_F^\top + S_t(I - \widehat{V}_F \widehat{V}_F^\top) \tag{5}$$

This update rule preserves the orthonormality of S_{t+1}, ensuring it remains on the Grassmannian. The last term in (5), projects the previous subspace onto the orthogonal complement of $\widehat{V}_F$, ensuring that the portion of S_t which has not been updated in this step is still included.

Projection-Aware Optimizer. In Adam, the first and second momentum update rules are as shown in (6) and (7), respectively (reminder: $\widetilde{G}_t$ is the projection of gradient into low-rank subspace).

$$M_t \leftarrow \beta_1 \cdot M_{t-1} + (1 - \beta_1) \cdot \widetilde{G}_t \tag{6}$$

$$\mathcal{V}_t \leftarrow \beta_2 \cdot \mathcal{V}_{t-1} + (1 - \beta_2) \cdot \widetilde{G}_t^2 \tag{7}$$

Algorithm 1 SubTrack++
(Subspace Tracking , Projection-Aware Optimizer , Recovery Scaling , Regular Adam)

Require: $W_t, G_t \in \mathbb{R}^{m\times n}$ with $m \leq n$ (w.l.o.g.), learning rate α, decay rates β_1 and β_2, SubTrack++ step-size η, rank r, subspace update interval k, recovery scaling limiter factor ζ. We use $\oslash$ to denote Hadamard division.

$S_0 \leftarrow U[:, :r]$, where $U, S, V \leftarrow \text{SVD}(G_0)$ {Initializing First Subspace}
for $t = 0, \ldots, T$ **do**
  if $t \bmod k == 0$ **then**
    $G_{lr} = \arg\min_A \|(S_{t-1}A - G_t)\|^2$, and $R = G_t - S_{t-1}G_{lr}$
    $\nabla F = -2RG_{lr}^\top \approx \widehat{U}_F\widehat{\Sigma}_F\widehat{V}_F^\top$
    $S_t = (S_{t-1}\widehat{V}_F \quad \widehat{U}_F)\begin{pmatrix}\cos\widehat{\Sigma}_F\eta \\ -\sin\widehat{\Sigma}_F\eta\end{pmatrix}\widehat{V}_F^\top + S_{t-1}(I - \widehat{V}_F\widehat{V}_F^\top)$
    $M_t \leftarrow \beta_1 \cdot (S_t^\top S_{t-1}M_{t-1}) + (1-\beta_1)\cdot\widetilde{G}_t$ {$\widetilde{G}_t = S_t^\top G_t$: low-rank projection of G_t}
    $\mathcal{V}_t \leftarrow \beta_2 \cdot [(1-\beta_2^{t-1})|(S_t^\top S_{t-1})^2 \cdot (\mathcal{V}_{t-1} - M_{t-1}^2) + (S_t^\top S_{t-1}\cdot M_{t-1})^2|] + (1-\beta_2)\cdot\widetilde{G}_t^2$
  else
    $S_t = S_{t-1}$
    $M_t \leftarrow \beta_1 \cdot M_{t-1} + (1-\beta_1)\cdot\widetilde{G}_t$
    $\mathcal{V}_t \leftarrow \beta_2 \cdot \mathcal{V}_{t-1} + (1-\beta_2)\cdot\widetilde{G}_t^2$
  end if
  $\widetilde{G}_t^O = M_t \oslash \sqrt{\mathcal{V}_t + \epsilon}$, $\widehat{G}_t = S_t\widetilde{G}_t^O$ {$\widetilde{G}_t^O$: optimizer's output, $\widehat{G}_t$: projected-back gradients }
  $\phi_t(G_t)_i = \frac{\|\widetilde{G}_{t,:,i}^O\|}{\|\widetilde{G}_{t,:,i}\|}$, $\Lambda_t = \phi_t(G_t)(G_t - S_t\widetilde{G}_t)$ {We use $\oslash$ to denote Hadamard division.}
  if $\frac{\Lambda_t}{\Lambda_{t-1}} > \zeta$ then $\Lambda_t \leftarrow \frac{\Lambda_t}{\|\Lambda_t\|}\cdot\zeta\|\Lambda_{t-1}\|$
  $W_t \leftarrow W_{t-1} - \alpha\cdot\widehat{G}_t$ $-\alpha\cdot\Lambda_t$
end for

Inspired by methods such as LDAdam [Robert et al., 2025], we emphasize the importance of updating optimizer states in a projection-aware manner to account for shifting subspaces; otherwise, misaligned projections can distort the optimizer's performance. To address this, at each subspace update step, we modify Adam's original update rules in (6) and (7), replacing them with projection-aware counterparts represented in (8) and (9), which reflect subspace changes into optimizer statistics [Robert et al., 2025]. Further details regarding these projections can be found in Appendix C.

$$M_t \leftarrow \beta_1 \cdot (S_t^\top S_{t-1}M_{t-1}) + (1-\beta_1)\cdot\widetilde{G}_t \tag{8}$$

$$\mathcal{V}_t \leftarrow \beta_2 \cdot [(1-\beta_2^{t-1})|(S_t^\top S_{t-1})^2 \cdot (\mathcal{V}_{t-1} - M_{t-1}^2) + (S_t^\top S_{t-1}\cdot M_{t-1})^2|] + (1-\beta_2)\cdot\widetilde{G}_t^2 \tag{9}$$

These projection-aware update rules enables Adam optimizer to track optimization dynamics precisely as the subspace evolves, achieving significantly better practical performance.

Recovery Scaling. Optimizer outputs $\widetilde{G}_t^O = M_t \oslash \sqrt{\mathcal{V}_t + \epsilon}$ ('$\oslash$' denotes Hadamard division), which is then projected back via $\widehat{G}_t = S_t\widetilde{G}_t^O$ to be used in weight update; however, low-rank projections inevitably discard some information in the full gradient matrix, that could enhance performance if properly utilized. Fira [Chen et al., 2025] observed that adaptive optimizers like Adam exhibit consistent scaling behaviour in low-rank and full-rank regimes. This suggests that the scaling information of the low-rank optimizer can be used to recover and rescale the discarded components of the gradient. A similar method is also employed in APOLLO [Zhu et al., 2025]. Consequently, an additional correction term is added to the standard weight update rule, as formalized in (10).

$$W_t \leftarrow W_{t-1} - \alpha\cdot\widehat{G}_t - \alpha\cdot\phi_t(G_t)(G_t - S_t\widetilde{G}_t) \tag{10}$$

Here α is the learning rate and $\phi_t(G_t)$ is the column-wise scaling factor computed based on the low-rank gradient representation, $\widetilde{G}_t$, and the optimizer's processed output, $\widetilde{G}_t^O$ as:

$$\phi_t(G_t)_i = \frac{\|\widetilde{G}_{t,:,i}^O\|}{\|\widetilde{G}_{t,:,i}\|} \tag{11}$$

Following Fira's observation [Chen et al., 2025], we employ a gradient-clipping-inspired mechanism to stabilize training. Specifically, we limit the growth rate of $\Lambda_t = \phi_t(G_t)(G_t - S_t\widetilde{G}_t)$ by a factor ζ, and apply a correction whenever it exceeds this threshold as:

$$\Lambda_t \leftarrow \frac{\Lambda_t}{\|\Lambda_t\|} \cdot \zeta\|\Lambda_{t-1}\| \tag{12}$$

By incorporating a geometry-aware perspective into low-rank optimization and applying this approach across all components of the training pipeline, SubTrack++ demonstrates robust and stable training, and achieves state-of-the-art performance while maintaining minimal memory footprint and wall-time. The overall flow of operations in SubTrack++ is illustrated in Algorithm 1.

3 Theoretical Analysis

In this section, we analyze the convergence of Grassmannian Subspace Tracking applied in SubTrack++ using theoretical analysis. To begin, the general weights update rule is as follows:

$$W_t = W_0 - \alpha \cdot \sum_{t'=0}^{t'=t-1} \widehat{G}_{t'} \tag{13}$$

As previously mentioned, we use left projection if $m \leq n$, where m and n are the dimensions of the gradient matrix, and vice versa. Thus, $\widehat{G}_{t'}$ can be computed as shown in (14).

$$\widehat{G}_{t'} = \begin{cases} S_{t'}\rho_{t'}(S_{t'}^\top G_{t'}), & \text{if } m \leq n \\ \rho_{t'}(G_{t'}S_{t'})S_{t'}^\top, & \text{otherwise} \end{cases} \tag{14}$$

Here, $S_{t'}$ is the projection matrix that projects the gradient onto the subspace, and $\rho_{t'}$ is representing the entry-wise regularizer used in the optimizer. If we use the full projection, then $\widehat{G}_{t'}$ will be computed as shown in (15); where $S^l_{t'}$ and $S^r_{t'}$ are the rank-r left and right projection matrices.

$$\widehat{G}_{t'} = S^l_{t'}\rho_{t'}({S^l_{t'}}^\top G_{t'}S^r_{t'}){S^r_{t'}}^\top \tag{15}$$

Definition 3.1 (L-continuity). A function $f(X)$ has Lipschitz-continuity (L-continuity) if for any X_1 and X_2, $\|f(X_2) - f(X_1)\|_F \leq L\|X_2 - X_1\|_F$

Theorem 3.2 (Convergence of Grassmannian Subspace Tracking). *Suppose gradient has the following form with functions A_i, B_i, and C_i being L-continuous as per* ***Def.*** *3.1 with constants L_A, L_B, and L_C w.r.t. weight matrix W_t; and $\|W_t\|_F \leq M$; where W_t denotes the weight matrix at step t, and M is a scalar value,*

$$G = \sum_i A_i + \sum_i B_i W C_i.$$

Now, define $\widehat{B}_{i,t} = (S^l_{i,t})^\top B_i(W_t)S^l_{i,t}$ and $\widehat{C}_{i,t} = (S^r_{i,t})^\top C_i(W_t)S^r_{i,t}$, where $S^l_{i,t}$ and $S^r_{i,t}$ are the rank-r left and right projection matrices; $B_i(W_t)$ and $C_i(W_t)$ denote the dependence of B_i and C_i on the weight matrices W_t. Further letting $P_t = {S^l_t}^\top G_t S^r_t$, and $\kappa_t = \frac{1}{N}\sum_i \lambda_{min}(\widehat{B}_{i,t})\lambda_{min}(\widehat{C}_{i,t})$, where $\lambda_{min}(\cdot)$ denotes the minimum eigenvalue over each batch, and N representing the number of samples in a batch. Assuming that the projection matrices remain constant during the training. Then for learning-rate μ and $min(\kappa_t) > (L_A + 2L_BL_CM^2)$, subspace tracking, with $\rho_t \equiv 1$ (the element-wise regularizer of the optimizer) satisfies:

$$\|P_t\|_F \leq [1 - \mu(\kappa_{t-1} - L_A - 2L_BL_CM^2)]\|P_{t-1}\|_F.$$

That is, $P_t \to 0$ and it converges.

The proof of Theorem 3.2 is provided in Appendix A, based on Zhao et al. [2024]. While both GaLore and SubTrack++ assume the subspace remains unchanged for the proof of convergence, GaLore must limit these updates to ensure convergence, as each update can potentially change the entire subspace. In contrast, SubTrack++ leverages rank-1 updates to the subspace, preventing drastic changes with each update. While a deeper analysis of slowly changing subspaces and their impact on convergence remains an open problem, in practice, this allows SubTrack++ to perform more frequent updates.

Here we investigate the Grassmannian update rule presented in (5), which is a direct application of Grassmann geometry [Edelman et al., 1998, Bendokat et al., 2024].

Table 1: We compare evaluation loss (↓) for pre-training Llama-based architectures on the C4 dataset over 10k iterations. SubTrack++ outperforms all other baselines in nearly every configuration. The best results are marked in **bold**, with the second-best performance underlined. *LDAdam could not be run on the 7B configuration due to an out-of-memory error with our available resources.

	60M r=128	**130M** r=256	**350M** r=256	**1B** r=512	**3B** r=512	**7B** r=1024
Full-Rank	3.41	3.25	3.40	4.61	4.52	4.30
GaLore [Zhao et al., 2024]	4.02	3.61	3.62	6.53	6.57	<u>5.55</u>
BAdam [Luo et al., 2024]	7.86	7.08	7.62	7.28	7.12	6.76
Online Subspace Descent [Liang et al., 2024]	4.18	3.88	4.09	6.79	6.85	5.69
LDAdam [Robert et al., 2025]	<u>3.52</u>	<u>3.44</u>	<u>3.67</u>	<u>4.70</u>	**4.39**	OOM*
Fira [Chen et al., 2025]	3.80	3.55	3.56	6.31	6.50	6.83
SubTrack++ (Ours)	**3.43**	**3.24**	**3.29**	**4.52**	<u>4.50</u>	**4.63**

Definition 3.3 (Exponential Map). The exponential map $\exp_p : T_pM \to M$ on a Riemannian manifold M is a mapping that assigns the point $\gamma(1) \in M$ to each tangent vector $\Delta \in T_pM$, where T_pM is the tangent space of M at p, and γ is the unique geodesic originating at p with initial velocity Δ. This map establishes a relationship between geodesics and the Riemannian exponential, such that $\gamma(t) = \exp_p(t\Delta)$ for $t \in \mathbb{R}$.

Definition 3.4 (Stiefel Manifold). The Stiefel manifold $St(n,p)$, parametrizes the set of all $n \times p$ orthonormal matrices U, each representing a rank-p subspace of $\mathbb{R}^n$.

Definition 3.5 (Grassmann Manifold). The Grassmannian manifold $Gr(n,p)$ parametrizes the set of all p-dimensional subspaces of $\mathbb{R}^n$. Each point can be represented by a projection matrix $P = UU^\top$, where $U \in St(n,p)$.

Theorem 3.6 (Grassmann Exponential). *Let $P = UU^\top \in Gr(n,p)$ be a point on the Grassmannian, where $U \in St(n,p)$ is the orthonormal basis of the corresponding subspace. Consider a tangent vector $\Delta \in T_P Gr(n,p)$, and let Δ_U^{hor} denote the horizontal lift of Δ to the horizontal space at U in the Stiefel manifold $St(n,p)$. Suppose the thin SVD of Δ_U^{hor} is given by $\Delta_U^{hor} = \hat{Q}\Sigma V^\top$, where $\hat{Q} \in St(n,r)$, $\Sigma = diag(\sigma_1, \dots, \sigma_r)$ contains the nonzero singular values of Δ_U^{hor} with $r = \min(p, n-p)$, and $V \in St(p,r)$. The Grassmann exponential map, representing the geodesic emanating from P in the direction Δ, is given by:*

$$Exp_P^{Gr}(t\Delta) = [UV\cos(t\Sigma)V^\top + \hat{Q}\sin(t\Sigma)V^\top + UV_\perp V_\perp^\top],$$

where $V_\perp \in \mathbb{R}^{p\times(p-r)}$ is any orthogonal complement of V.

The proof of Theorem 3.6 can be found in Appendix B. Leveraging this theorem and our notation in section 2, one can easily verify that the subspace update rule is as follows:

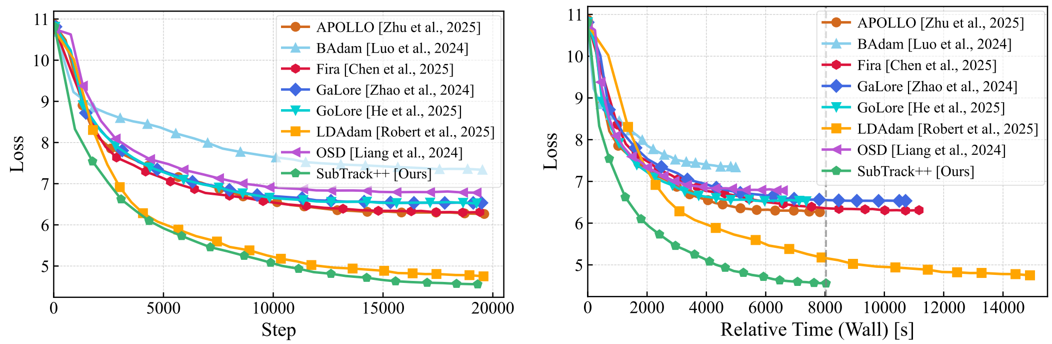
$$S_{t+1}(\eta) = (S_t\widehat{V}_F \quad \widehat{U}_F)\begin{pmatrix}\cos\widehat{\Sigma}_F\eta \\ -\sin\widehat{\Sigma}_F\eta\end{pmatrix}\widehat{V}_F^\top + S_t(I - \widehat{V}_F\widehat{V}_F^\top)$$

This update rule generally converges to a stable subspace if the step size η decreases over time [Balzano et al., 2011]. However, a decreasing step size can impair the ability to accurately track and adapt to subspace changes. Consequently, SubTrack++ uses a constant step size to effectively adjust subspaces. This approach does not hinder convergence, as proved in Theorem 3.2, which guarantees convergence as long as changes are controlled to maintain the stable subspace assumption.

4 Experiments and Results

We evaluated SubTrack++ across diverse models and datasets through pre-training and fine-tuning, measuring key metrics critical to LLM democratization.

Pre-Training Experiments. We pre-trained several Llama-based models on the C4 dataset, with results in Table 1. To ensure a fair comparison, we benchmarked against a diverse set of baselines.



(a) Loss vs. Training Steps.

(b) Loss vs. Relative Wall-Time.

Figure 3: Comparison of baselines in pre-training Llama-1B architecture. (a) shows training loss (↓) versus training steps. (b) shows the same runs against wall-time. SubTrack++ outperforms all baselines; substantially reducing wall-time, especially compared to LDAdam, the top-performing baseline.

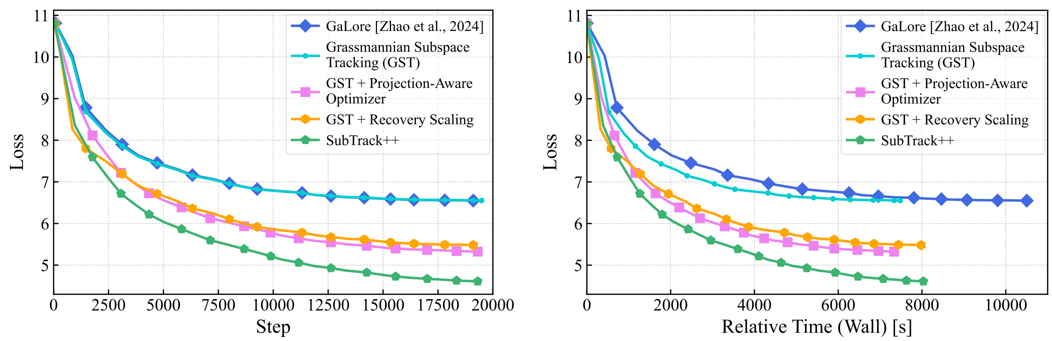
While all compared methods aim for memory-efficient training, their architectural principles yield distinct computational and convergence trade-offs. BAdam [Luo et al., 2024] achieves strong memory and time efficiency, yet its partial parameter update strategy compromises final performance. GaLore [Zhao et al., 2024] constrains gradients to a low-rank subspace estimated via periodic SVD, an approach sensitive to noise and costly to compute, while discarding information residing in the orthogonal complement. Fira [Chen et al., 2025] introduces norm-based recovery scaling to mitigate this information loss, but its reliance on frequent SVD still leads to substantial wall-time overhead. LDAdam [Robert et al., 2025] replaces SVD with PowerSGD-based iterative updates and a projection-aware optimizer that synchronizes internal states with evolving subspaces, improving convergence and stability but incurring high per-step costs due to continual subspace updates. It also adds an extended error-feedback mechanism to compensate for both gradient and optimizer compression to target GaLore's shortcomings. Online Subspace Descent (OSD) [Liang et al., 2024], our tracking-based baseline, further reduces complexity by employing Online PCA for subspace tracking.

SubTrack++ departs from these formulations by treating subspace evolution as a geometric tracking problem on the Grassmannian. It performs efficient rank-1 geodesic updates that reuse historical subspace information, inherently preserving consistency and avoiding the instability of discrete subspace resets. Simultaneously, it aligns Adam's internal states through projection-aware optimizer and leverages recovery scaling to reintegrate the lost information. This unified approach yields a new training paradigm that retains the memory efficiency of low-rank methods, while achieving on-par runtime efficiency compared to fastest baselines like APOLLO [Zhu et al., 2025]. SubTrack++ sets new state-of-the-art results across model scales, running 43% faster than LDAdam, the strongest prior baseline, on the 1B model, and 67% faster on the 3B model (see Table 6).

Figure 3 demonstrates the pre-training of a 1B-parameter Llama model across several baselines. SubTrack++ has the fastest convergence in both training steps and wall-time, highlighting the effectiveness of geometry-aware optimization in improving performance and reducing resource consumption. To further assess its generalization in longer training regimes and larger models, we extend training to 100k steps and compare SubTrack++ with GaLore. As shown in Figure 7, SubTrack++ converges substantially faster, achieving an evaluation loss of 3.37 (a significant improvement over the 10k-step setting), while GaLore reaches 4.64 under the same conditions.

As shown in Table 1, SubTrack++ occasionally outperforms full-rank training. This effect may be attributed to the implicit regularization introduced by low-rank projections, which can enhance generalization in overparameterized models. Similar trends have been reported in other studies [Robert et al., 2025, Zhu et al., 2025, Chen et al., 2025]. In addition, to further validate that convergence on the projected gradient reflects convergence of the original full-gradient, we measured both norms during Llama-1B pre-training. The full-gradient norm drops from $0.46 \rightarrow 0.08$, while the projected-gradient norm drops from $0.45 \rightarrow 0.05$, following nearly identical trajectories. This confirms that the optimization progress observed on the projected gradient accurately mirrors convergence in the original gradient space, consistent with prior low-rank gradient findings.

Hyperparameters of pre-training experiments are provided in Appendix E, with detailed runtime and memory reports in Appendix F.



(a) Loss vs. Training Steps.

(b) Loss vs. Relative Wall-Time.

Figure 4: Ablation study comparing pure Grassmannian subspace tracking with incremental additions of the projection-aware optimizer and recovery scaling, leading to SubTrack++ . While Grassmannian tracking alone almost matches GaLore's step-wise convergence (a), it significantly reduces wall-time (b).

Ablation Studies. We conducted an ablation study to assess the individual and combined contributions of the projection-aware optimizer and recovery scaling, integrated with Grassmannian subspace tracking, on the 1B Llama model. Experimental settings are summarized in Table 4. As illustrated in Figure 4-b, Grassmannian subspace tracking alone substantially reduces wall-time compared to frequent SVD updates. Both the projection-aware optimizer and recovery scaling independently provide notable performance gains over baseline subspace tracking, lowering the loss from 6.53 to 5.43 and 5.28, respectively. Their combination, SubTrack++ , further improves the loss to 4.51, surpassing all baselines. Importantly, these improvements are achieved with only minimal increases in runtime and memory overhead, thanks to the efficiency of Grassmannian subspace tracking.

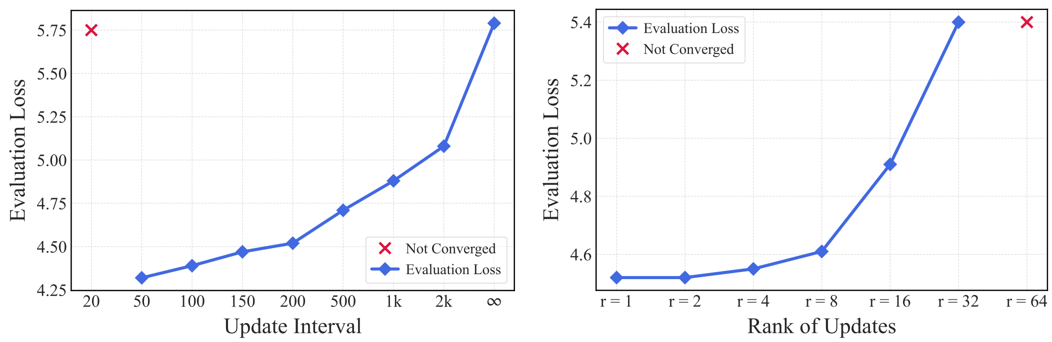
In addition, we conducted ablations on the subspace update rank and update frequency. As shown in Figure 5-a, more frequent updates can further improve performance, and the computational efficiency of SubTrack++ enables higher update frequencies with minimal overhead. However, excessively frequent updates may impede convergence. Figure 5-b shows that rank-1 updates achieve the best performance among all tested values. This suggests that making controlled, small adjustments to the underlying subspace helps maintain stability, while updating the subspace along the most informative direction prevents stagnation in a low-rank region and enhances generalization.

Fine-Tuning Experiments. RoBERTa-Base and RoBERTa-Large are fine-tuned on GLUE [Wang et al., 2019] and SuperGLUE [Sarlin et al., 2020] tasks; with the results presented in Table 7 and 8, respectively. We also conducted supervised fine-tuning of the Llama-2-7B-chat-hf model for one epoch on the Alpaca [Taori et al., 2023] dataset. In this experiment, SubTrack++ achieved 36% lower wall-time compared to GaLore and 65% compared to LDAdam. The results are presented in Table 9. More details and hyperparameters are provided in Appendix G.

Time and Space Complexity. Table 2 provides memory requirements of the optimizer states and the time complexity of the subspace update step considering an $m \times n$ gradient matrix with $m \leq n$. GaLore [Zhao et al., 2024] and Fira [Chen et al., 2025] periodically perform SVD to estimate the underlying subspace, while LDAdam [Robert et al., 2025] relies on the faster PowerSGD to update the subspace at every iteration. In contrast, SubTrack++ employs Grassmannian-based subspace tracking at the same frequency as GaLore and Fira. Comparing the time complexities of these methods, highlights why SubTrack++ is significantly more efficient than SVD-based methods. A breakdown of the subspace update time complexity for SubTrack++ is shown in Appendix D. Additionally, the memory required for storing optimizer states in SubTrack++ , is equivalent to GaLore and other baselines.

Table 2: The optimizer's state parameter count and subspace update time complexity across baselines, given a gradient matrix of dimension $m \times n$ and projection rank r where $r \ll m \leq n$. *LDAdam updates the subspace at every iteration, while other methods update it every k steps.

	Optimizer Mem.	**Subspace Update Time**
Adam	$2mn$	–
LDAdam*	$mr + 2nr$	$O(mnr)$
GaLore, Fira	$mr + 2nr$	$O(nm^2)$
SubTrack++	$mr + 2nr$	$O(mnr)$



(a) Loss vs. Subspace Update Frequency.

(b) Loss vs. Subspace Update Rank.

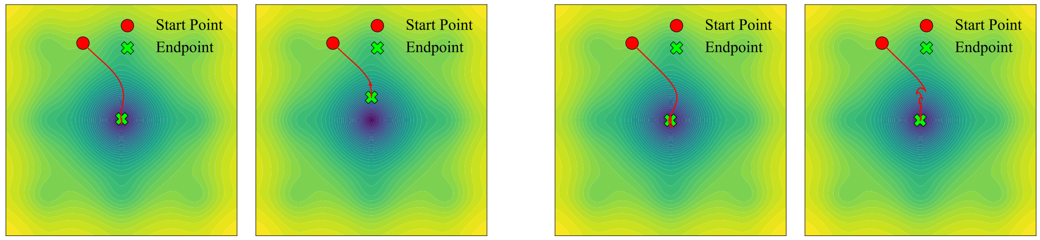
Figure 5: Ablation results on (a) update frequency: decreasing the update interval (i.e., increasing the frequency) improves evaluation performance up to a point, but overly frequent updates hinder training convergence. (b) update rank: increasing the rank of updates degrades model performance, and beyond a certain threshold, can prevent convergence. These results emphasize the importance of controlled subspace adjustments.

Robust Subspace Tracking. Relying on SVD for subspace updates makes methods sensitive to noise and abrupt changes [He et al., 2025]. Figure 6 compares Grassmannian subspace tracking with GaLore's SVD on the Ackley function, highlighting how SVD causes erratic jumps, while our subspace tracking ensures robust optimization. GaLore struggles to reach the global minimum within 100 steps at scale factor 1, and although increasing the scale factor to 3 improves performance, it amplifies jumps that hinder convergence in non-convex settings, revealing sensitivity to hyperparameters, noise, and abrupt changes. To empirically measure the robustness of SubTrack++ in tracking evolving subspaces, we quantified subspace drift using the norm of the tangent vector ∇F, which reflects the deviation between the projected gradient and the original gradient matrix. On Llama-350M pre-training, this norm rapidly decays from 0.06 to below 0.0002 within 5-7 subspace updates and remains near zero thereafter, indicating highly stable and well-aligned subspace tracking. This quantitative stability supports our qualitative observations in Figure 6 and aligns with prior findings in Grassmannian optimization.

5 Related Works

Parameter-Efficient Training. Several works aim to improve the efficiency of training LLMs, addressing a growing demand as their popularity rapidly increases. Popular LoRA [Hu et al., 2021] significantly reduces memory requirements for fine-tuning LLMs by leveraging two low-rank trainable low-rank matrices. Dettmers et al. [2024] employ quantization techniques and paged optimizers to further reduce memory usage. Additionally, Yaras et al. [2024] introduce Deep LoRA to address overfitting issues and reducing the need for precise tuning of the rank parameter. Several other works have also extended LoRA to enhance the efficiency of training and fine-tuning LLMs [Lialin et al., 2023, Renduchintala et al., 2024, Xia et al., 2024, Pan et al., 2024]. Miles et al. [2024] propose compressing intermediate activations and reconstructing them during backpropagation to enhance memory efficiency. Yen et al. [2025] propose adjustments to the LoRA factorization that promotes balanced training and correspondingly update the optimizer's internal states (i.e., first and second moments) to remain consistent under the change of basis. Additionally, Hao et al. [2024] demonstrate that full-parameter fine-tuning is feasible by random projections on the gradient matrix, showing that LoRA essentially performs a down-projection of the gradient. BAdam [Luo et al., 2024] leverages the block coordinate descent framework to reduce memory consumption while maintaining capabilities comparable to Adam.

Gradient Low-Rank Projection. Several approaches aim to reduce optimizer states, as optimizers like Adam [Kingma and Ba, 2017] account for a significant portion of memory footprint [Li et al., 2023, Anil et al., 2019, Lv et al., 2024, Dettmers et al., 2022]. MicroAdam [Modoranu et al., 2024] tackles this by compressing the gradient space and utilizing the compression error through feedback loops. Adam-mini [Zhang et al., 2024] partitions model into blocks, assigning a single learning rate to each block to preserve performance while saving memory. Gur-Ari et al. [2018], Schneider et al. [2024], Yaras et al. [2023] suggest that a substantial portion of gradients lies within a largely consistent subspace. GaLore [Zhao et al., 2024] leverages this fact to reduce the optimizer's memory



(a) Ours, SF=1 (b) GaLore, SF=1 (c) Ours, SF=3 (d) GaLore, SF=3

Figure 6: Comparison of Grassmannian subspace tracking (Ours) (a, c) and GaLore's SVD (b, d) on the Ackley Function over 100 optimization steps, with a subspace update interval of 10. SF stands for scale factor; with a scale factor of 1, GaLore fails to reach the global minimum due to abrupt jumps. At a scale factor of 3, while the minimum is reached, the jump length increases. This demonstrates SVD's sensitivity to noise and abrupt changes, highlighting the robustness of our subspace tracking method with its controlled subspace updates.

by projecting gradients into a low-rank subspace and then projecting them back for full parameter tuning. This approach has been integrated with other methods regarding efficient LLM training [Li et al., 2024]. However, not all layers' gradients evolve within a stable low-rank subspace. Jaiswal et al. [2024] identify layers where gradients evolve within a low-dimensional subspace and fine-tune only those layers, freezing the others to avoid inefficient low-rank updates. Grass [Muhamed et al., 2024] reduces memory usage by applying sparse projection matrices to the gradient. Ramesh et al. [2024] dynamically select and update a subset of parameters, for a fast and memory-efficient training. Fira [Chen et al., 2025] utilize a norm-based scaling method along with GaLore to maintain performance comparable to full-rank training. GoLore [He et al., 2025] addresses GaLore's convergence issues and employ random projection in latter steps as a solutions. LDAdam [Robert et al., 2025] performs optimization within lower-dimensional subspaces, incorporating a projection-aware optimization update rule and a generalized error feedback mechanism. Projection-Aware APOLLO [Zhu et al., 2025] approximates channel-wise learning rate scaling based on random projection. Also, Liang et al. [2024] introduce a dynamically evolving projection matrix updated via online PCA, enhancing the model's ability to navigate the parameter space efficiently without relying on expensive SVD.

Geometric Subspace Updates. A common approach in working with high-dimensional data is to project the data into a lower-dimensional space, and many studies focus on tracking these subspaces as they evolve. Balzano et al. [2011] introduce an incremental method for updating subspaces on the Grassmannian when the data is partially observed. Zhang and Balzano [2016] and Kasai [2017] propose methods to handle noise effect in tracking these subspaces. Furthermore, Blocker et al. [2023] present a method for evolving geodesic-based data in the Grassmannian for updating the subspace effectively. Mo et al. [2025] propose LORO, a low-rank pretraining method that performs Riemannian optimization on the manifold of fixed-rank matrices, enabling parameter- and memory-efficient training by updating low-rank factors via manifold-aware gradients and retractions.

6 Discussion and Conclusion

We propose SubTrack++ , a time- and memory-efficient approach that projects gradients into a low-rank subspace and uses Grassmannian subspace tracking to preserve the computed subspace while incorporating gradient components from the orthogonal complement. By integrating projection-aware optimizers that reflect subspace changes in Adam's internal statistics and utilizing the gradient information lost during low-rank projection, SubTrack++ achieves state-of-the-art convergence and accuracy across all baselines. While Grassmannian subspace tracking integrates seamlessly with various optimizers as a plug-and-play module, extending projection-aware optimization beyond the Adam family requires further design and investigation. Also benchmarking on models with more than 7B parameters was not feasible regarding our limited time and resources.

Acknowledgments and Disclosure of Funding

Sirisha Rambhatla would like to acknowledge support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, RGPIN-2022-03512.

References

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection, 2024. URL `https://arxiv.org/abs/2403.03507`.

Ajay Jaiswal, Lu Yin, Zhenyu Zhang, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. From galore to welore: How low-rank weights non-uniformly emerge from low-rank gradients, 2024. URL `https://arxiv.org/abs/2407.11239`.

Aashiq Muhamed, Oscar Li, David Woodruff, Mona Diab, and Virginia Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients, 2024. URL `https://arxiv.org/abs/2406.17660`.

Roy Miles, Pradyumna Reddy, Ismail Elezi, and Jiankang Deng. Velora: Memory efficient training using rank-1 sub-token projections, 2024. URL `https://arxiv.org/abs/2405.17991`.

Ionut-Vlad Modoranu, Mher Safaryan, Grigory Malinovsky, Eldar Kurtic, Thomas Robert, Peter Richtarik, and Dan Alistarh. Microadam: Accurate adaptive optimization with low space overhead and provable convergence, 2024. URL `https://arxiv.org/abs/2405.15593`.

Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors, 2024. URL `https://arxiv.org/abs/2402.03293`.

Pengxiang Li, Lu Yin, Xiaowei Gao, and Shiwei Liu. Owlore: Outlier-weighed layerwise sampled low-rank projection for memory-efficient llm fine-tuning, 2024. URL `https://arxiv.org/abs/2405.18380`.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost, 2016. URL `https://arxiv.org/abs/1604.06174`.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2020. URL `https://arxiv.org/abs/1910.02054`.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. *arXiv preprint arXiv:2406.04112*, 2024.

Vladislav Lialin, Namrata Shivagunde, Sherin Muckatira, and Anna Rumshisky. Relora: High-rank training through low-rank updates, 2023. URL `https://arxiv.org/abs/2307.05695`.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-LoRA: Enhancing parameter efficiency of LoRA with weight tying. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8694–8705, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.481. URL `https://aclanthology.org/2024.naacl-long.481`.

Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning, 2024. URL `https://arxiv.org/abs/2401.04151`.

Qijun Luo, Hengxu Yu, and Xiao Li. Badam: A memory efficient full parameter optimization method for large language models, 2024. URL `https://arxiv.org/abs/2404.02827`.

Amrutha Varshini Ramesh, Vignesh Ganapathiraman, Issam H. Laradji, and Mark Schmidt. Blockllm: Memory-efficient adaptation of llms by selecting and optimizing the right coordinate blocks, 2024. URL `https://arxiv.org/abs/2406.17296`.

Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states, 2023. URL https://arxiv.org/abs/2309.01507.

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory-efficient adaptive optimization, 2019. URL https://arxiv.org/abs/1901.11150.

Kai Lv, Yuqing Yang, Tengxiao Liu, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8187–8198, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.445.

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

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more, 2024. URL https://arxiv.org/abs/2406.16793.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace, 2018. URL https://arxiv.org/abs/1812.04754.

Jan Schneider, Pierre Schumacher, Simon Guist, Le Chen, Daniel Häufle, Bernhard Schölkopf, and Dieter Büchler. Identifying policy gradient subspaces, 2024. URL https://arxiv.org/abs/2401.06604.

Can Yaras, Peng Wang, Wei Hu, Zhihui Zhu, Laura Balzano, and Qing Qu. Invariant low-dimensional subspaces in gradient descent for learning deep matrix factorizations. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. LDAdam: Adaptive optimization from low-dimensional gradient statistics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Zkp1GuHerF.

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

Kaizhao Liang, Bo Liu, Lizhang Chen, and qiang liu. Memory-efficient LLM training with online subspace descent. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=P8rTCT6g45.

Namrata Vaswani, Thierry Bouwmans, Sajid Javed, and Praneeth Narayanamurthy. Robust subspace learning: Robust pca, robust subspace tracking, and robust subspace recovery. *IEEE Signal Processing Magazine*, 35(4):32–55, July 2018. ISSN 1558-0792. doi: 10.1109/msp.2018.2826566. URL http://dx.doi.org/10.1109/MSP.2018.2826566.

Yutong He, Pengrui Li, Yipeng Hu, Chuyan Chen, and Kun Yuan. Subspace optimiztion for large language models with convergence guarantees, 2025. URL https://openreview.net/forum?id=udtrtwkvk5.

Jiayao Zhang, Guangxu Zhu, Robert W Heath Jr, and Kaibin Huang. Grassmannian learning: Embedding geometry awareness in shallow and deep learning. *arXiv preprint arXiv:1808.02229*, 2018.

Laura Balzano, Robert Nowak, and Benjamin Recht. Online identification and tracking of subspaces from highly incomplete information, 2011. URL https://arxiv.org/abs/1006.4046.

Jun He, Laura Balzano, and John C. S. Lui. Online robust subspace tracking from partial information, 2011. URL https://arxiv.org/abs/1109.3827.

Cameron J. Blocker, Haroon Raja, Jeffrey A. Fessler, and Laura Balzano. Dynamic subspace estimation with grassmannian geodesics, 2023. URL https://arxiv.org/abs/2303.14851.

Rudrasis Chakraborty, Søren Hauberg, and Baba C. Vemuri. Intrinsic grassmann averages for online linear and robust subspace learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 801–809, 2017. doi: 10.1109/CVPR.2017.92.

Dejiao Zhang and Laura Balzano. Global convergence of a grassmannian gradient descent algorithm for subspace estimation, 2016. URL https://arxiv.org/abs/1506.07405.

Laura Balzano, Yuejie Chi, and Yue M Lu. Streaming pca and subspace tracking: The missing data case. *Proceedings of the IEEE*, 106(8):1293–1310, 2018.

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?, 2025. URL https://openreview.net/forum?id=lR7rqLtsXZ.

Thomas Bendokat, Ralf Zimmermann, and P.-A. Absil. A grassmann manifold handbook: basic geometry and computational aspects. *Advances in Computational Mathematics*, 50(1), January 2024. ISSN 1572-9044. doi: 10.1007/s10444-023-10090-8. URL http://dx.doi.org/10.1007/s10444-023-10090-8.

Alan Edelman, T. A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints, 1998. URL https://arxiv.org/abs/physics/9806030.

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, 2019. URL https://arxiv.org/abs/1804.07461.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks, 2020. URL https://arxiv.org/abs/1911.11763.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning, 2024. URL https://arxiv.org/abs/2403.17919.

Jui-Nan Yen, Si Si, Zhao Meng, Felix Yu, Sai Surya Duvvuri, Inderjit S Dhillon, Cho-Jui Hsieh, and Sanjiv Kumar. LoRA done RITE: Robust invariant transformation equilibration for loRA optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VpWki1v2P8.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Hiroyuki Kasai. Fast online low-rank tensor subspace tracking by cp decomposition using recursive least squares from incomplete observations, 2017. URL https://arxiv.org/abs/1709.10276.

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. URL https://openreview.net/forum?id=i0zzO7Hslk.

A Convergence of SubTrack++

Theorem 3.2 (Convergence of Grassmannian Subspace Tracking). *Suppose gradient has the following form with functions A_i, B_i, and C_i being L-continuous as per **Def.** 3.1 with constants L_A, L_B, and L_C w.r.t. weight matrix W_t; and $\|W_t\|_F \leq M$; where W_t denotes the weight matrix at step t, and M is a scalar value,*

$$G = \sum_i A_i + \sum_i B_i W C_i.$$

Now, define $\widehat{B}_{i,t} = (S^l_{i,t})^\top B_i(W_t) S^l_{i,t}$ and $\widehat{C}_{i,t} = (S^r_{i,t})^\top C_i(W_t) S^r_{i,t}$, where $S^l_{i,t}$ and $S^r_{i,t}$ are the rank-r left and right projection matrices; $B_i(W_t)$ and $C_i(W_t)$ denote the dependence of B_i and C_i on the weight matrices W_t. Further letting $P_t = {S^l_t}^\top G_t S^r_t$, and $\kappa_t = \frac{1}{N}\sum_i \lambda_{min}(\widehat{B}_{i,t})\lambda_{min}(\widehat{C}_{i,t})$, where $\lambda_{min}(\cdot)$ denotes the minimum eigenvalue over each batch, and N representing the number of samples in a batch. Assuming that the projection matrices remain constant during the training. Then for learning-rate μ and $min(\kappa_t) > (L_A + 2L_B L_C M^2)$, subspace tracking, with $\rho_t \equiv 1$ (the element-wise regularizer of the optimizer) satisfies:

$$\|P_t\|_F \leq [1 - \mu(\kappa_{t-1} - L_A - 2L_B L_C M^2)]\|P_{t-1}\|_F.$$

That is, $P_t \to 0$ and it converges.

proof. To demonstrate that SubTrack++ converges to the global minimum during training, we begin by deriving the recursive form of the gradients.

Let $\otimes$ denote the Kronecker product. Then, $vec(AXB) = (B^\top \otimes A)vec(X)$.

By applying vec to the gradient form given in the theorem, we obtain:

$$g_t = vec(G_t) = vec(\sum_i A_i + \sum_i B_i W C_i) = a_t - D_t w_t \tag{16}$$

where $g_t := vec(G_t)$, $w_t := vec(W_t)$, $a_t := \frac{1}{N}\sum_i vec(A_{i,t})$, and $D_t = \frac{1}{N}\sum_i C_{i,t} \otimes B_{i,t}$.

As defined in the theorem, let $P_t = {S^l_t}^\top G_t S^r_t$. Its vectorized form can be expressed using the Kronecker product as follows:

$$\begin{aligned} p_t = vec(P_t) &= vec({S^l_t}^\top G_t S^r_t) = ({S^r_t}^\top \otimes {S^l_t}^\top) vec(G_t) \\ &= (S^r_t \otimes S^l_t)^\top vec(G_t) = (S^r_t \otimes S^l_t)^\top g_t \end{aligned} \tag{17}$$

Now recalling $\widehat{G}_t$ from (15), it can be written as:

$$\widehat{G}_t = S^l_t {S^l_t}^\top G_t S^r_t {S^r_t}^\top$$

Thus, its vectorized form will be:

$$\begin{aligned} vec(\widehat{G}_t) = \widehat{g}_t &= vec(S^l_t {S^l_t}^\top G_t S^r_t {S^r_t}^\top) = vec(S^l_t P_t {S^r_t}^\top) \\ &= (S^r_t \otimes S^l_t) vec(P_t) = (S^r_t \otimes S^l_t) p_t \end{aligned} \tag{18}$$

This is where the constant subspace assumption becomes necessary. To derive the recursive form of g_t, we assume that the projection matrices remain fixed throughout training, i.e., $S^r_t = S^r$ and $S^l_t = S^l$. Consequently, we can restate equations (17) and (18) as follows:

$$p_t = (S^r \otimes S^l)^\top g_t \tag{19}$$

$$\widehat{g}_t = (S^r \otimes S^l) p_t \tag{20}$$

Then we can write the recursive form of g_t:

$$\begin{aligned} g_t = a_t - D_t w_t &= (a_t - a_{t-1}) + (D_{t-1} - D_t) w_t + a_{t-1} - D_{t-1} w_t \\ &= e_t + a_{t-1} - D_{t-1}(w_{t-1} + \mu \widehat{g}_{t-1}) = e_t + g_{t-1} - \mu D_{t-1} \widehat{g}_{t-1} \end{aligned} \tag{21}$$

where $e_t := (a_t - a_{t-1}) + (D_{t-1} - D_t) w_t$.

Note that in deriving (21), we utilized the general form of the weight update rule, $w_{t+1} = w_t - \mu g_t$, which can be rewritten as $w_t = w_{t+1} + \mu g_t$. By applying this rule along with (16), we arrive at the second equality in (21) as follows:

$$\begin{aligned} g_t &= a_t - D_t w_t = a_t - D_t w_t - g_{t-1} + g_{t-1} \\ &= a_t - D_t w_t - a_{t-1} + D_{t-1} w_{t-1} + a_{t-1} - D_{t-1} w_{t-1} \\ &= a_t - D_t w_t - a_{t-1} + D_{t-1}(w_t + \mu g_{t-1}) + a_{t-1} - D_{t-1}(w_t + \mu g_{t-1}) \\ &= a_t - D_t w_t - a_{t-1} + D_{t-1} w_t + \mu D_{t-1} g_{t-1} + a_{t-1} - D_{t-1} w_t - \mu D_{t-1} g_{t-1} \\ &= a_t - a_{t-1} + (D_{t-1} - D_t) w_t + a_{t-1} - D_{t-1} \end{aligned}$$

To obtain p_t from this recursive formulation, we can left-multiply by $(S^r \otimes S^l)^\top$, as shown in (20):

$$p_t = (S^r \otimes S^l)^\top e_t + (S^r \otimes S^l)^\top g_{t-1} - \mu (S^r \otimes S^l)^\top D_{t-1} \widehat{g}_{t-1} \tag{22}$$

Now, based on (19) and (20), p_t can be written as:

$$p_t = (S^r \otimes S^l)^\top e_t + p_{t-1} - \mu (S^r \otimes S^l)^\top D_{t-1} (S^r \otimes S^l) p_{t-1} \tag{23}$$

Let define:

$$\begin{aligned} \widehat{D}_t &:= (S^r \otimes S^l)^\top D_t (S^r \otimes S^l) = \frac{1}{N} \sum_i (S^r \otimes S^l)^\top (C_{i,t} \otimes B_{i,t})(S^r \otimes S^l) \\ &= \frac{1}{N} \sum_i ({S^r}^\top C_{i,t} S^r) \otimes ({S^l}^\top B_{i,t} S^l) \end{aligned} \tag{24}$$

Then we can expand (23) and show that:

$$p_t = (I - \mu \hat{D}_{t-1}) p_{t-1} + (S^r \otimes S^l)^\top e_t \tag{25}$$

Note that S^l and S^r are orthonormal matrices. This is ensured because the subspace is initialized using the SVD of G_0, and the Grassmannian update rule provided in (5) preserves the orthonormality of the subspace matrices throughout training. Since S^l and S^r are orthonormal, we have ${S^l}^\top S^l = I$ and ${S^r}^\top S^r = I$. Consequently, we can bound the norm of the second term in (25) as follows:

$$\|(S^r \otimes S^l)^\top e_t\|_2 = \|vec({S^l}^\top E_t S^r)\|_2 = \|{S^l}^\top E_t S^r\|_F \le \|E_t\|_F \tag{26}$$

Here E_t is the matrix form of e_t, and as declared before, $e_t := (a_t - a_{t-1}) + (D_{t-1} - D_t) w_t$, thus:

$$E_t := \frac{1}{N} \sum_i (A_{i,t} - A_{i,t-1}) + \frac{1}{N} \sum_i (B_{i,t-1} W_t C_{i,t-1} - B_{i,t} W_t C_{i,t}) \tag{27}$$

Next, we need to find an upper bound for the norm of each term in (27) to establish an upper bound for $\|E_t\|_F$. Based on the assumptions of the theorem, A_i, B_i, and C_i exhibit L-Lipschitz continuity with constants L_A, L_B, and L_C, respectively. Additionally, $\|W_t\|_F$ is bounded by a scalar M. We have:

$$\|A_t - A_{t-1}\|_F \quad \le \quad L_A \|W_t - W_{t-1}\|_F = \mu L_A \|\tilde{G}_{t-1}\|_F \le \mu L_A \|P_{t-1}\|_F \tag{28}$$

In the first equality, we apply (13), while the last equality holds due to (20) and the orthonormality of the projection matrices. The subsequent two inequalities can be derived similarly using these equations.

$$\begin{aligned} \|(B_t - B_{t-1}) W_t C_{t-1}\|_F &\le L_B \|W_t - W_{t-1}\|_F \|W_t\|_F \|C_{t-1}\|_F \\ &= \mu L_B L_C M^2 \|P_{t-1}\|_F \end{aligned} \tag{29}$$

$$\begin{aligned} \|B_t W_t (C_{t-1} - C_t)\|_F &\le L_C \|B_t\|_F \|W_t\|_F \|W_{t-1} - W_t\|_F \\ &= \mu L_B L_C M^2 \|P_{t-1}\|_F \end{aligned} \tag{30}$$

We can now derive the bound for $\|E_t\|_F$ as follows:

$$\begin{aligned} \|E_t\|_F \le \mu L_A \|\tilde{G}_{t-1}\|_F &\le \mu L_A \|P_{t-1}\|_F + \mu L_B L_C M^2 \|P_{t-1}\|_F + \mu L_B L_C M^2 \|P_{t-1}\|_F \\ &= \mu (L_A + 2 L_B L_C M^2) \|P_{t-1}\|_F \end{aligned} \tag{31}$$

To calculate the norm bound for the first term in (25), we first need to establish the bounds for $\widehat{D}_t$. This involves estimating the minimum eigenvalue of $\widehat{D}_t$.

If we define $\gamma_{min,i,t} = \lambda_{min}(S^{l^\top} B_{i,t} S^l)\lambda_{min}(S^{r^\top} C_{i,t} S^r)$, then it follows that $\lambda_{min}((S^{l^\top} B_{i,t} S^l) \otimes (S^{r^\top} C_{i,t} S^r)) = \gamma_{min,i,t}$. Consequently, $\widehat{D}_t$ will satisfy the following inequality for every unit vector $\vec{v}$:

$$\vec{v}^\top \widehat{D}_t \vec{v} = \frac{1}{N} \sum_i \vec{v}^\top \left[(S^{l^\top} B_{i,t} S^l) \otimes (S^{r^\top} C_{i,t} S^r) \right] \vec{v} \geq \frac{1}{N} \sum_i \gamma_{min,i,t} \tag{32}$$

this actually provides a lower bound for eigenvalues of $\widehat{D}_t$, thus:

$$\lambda_{\max}(I - \mu \widehat{D}_{t-1}) \leq 1 - \frac{\mu}{N} \sum_i \gamma_{min,i,t-1} \tag{33}$$

considering the definition of κ_t in the theorem, we can now easily show that:

$$\|P_t\|_F \leq [1 - \mu(\kappa_{t-1} - L_A - 2L_B L_C M^2)] \|P_{t-1}\|_F.$$

and completing the proof.

While SubTrack++ utilizes right/left projections to reduce memory consumption, the proof is presented using both projection matrices to ensure generality. Here, we demonstrate how the proof proceeds under the assumption $m \leq n$ (without loss of generality), which allows the use of the left projection matrix.

Using the left projection matrix, the current formulation of P_t, defined as $P_t = S_t^{l^\top} G_t S_t^r$, simplifies to $P_t = S_t^{l^\top} G_t$. Similarly, $\widehat{G}_t = S_t^l S_t^{l^\top} G_t S_t^r S_t^{r^\top}$ reduces to $\widehat{G}_t = S_t^l S_t^{l^\top} G_t$. From this point, the proof continues by substituting S_t^r with the identity matrix, allowing the derivation of the vectorized forms of g_t, $\widehat{g}_t$, p_t, and related terms.

The remainder of the proof remains largely unaffected. It can be readily verified that the recursive formulation of g_t is unchanged. Although the definition of P_t is modified, it continues to satisfy the bounds required for convergence, ensuring that P_t converges to 0 when the left projection matrix is used.

B Grassmann Exponential

Theorem 3.6 (Grassmann Exponential). *Let $P = UU^\top \in Gr(n,p)$ be a point on the Grassmannian, where $U \in St(n,p)$ is the orthonormal basis of the corresponding subspace. Consider a tangent vector $\Delta \in T_P Gr(n,p)$, and let Δ_U^{hor} denote the horizontal lift of Δ to the horizontal space at U in the Stiefel manifold $St(n,p)$. Suppose the thin SVD of Δ_U^{hor} is given by $\Delta_U^{hor} = \hat{Q}\Sigma V^\top$, where $\hat{Q} \in St(n,r)$, $\Sigma = diag(\sigma_1, \ldots, \sigma_r)$ contains the nonzero singular values of Δ_U^{hor} with $r = \min(p, n-p)$, and $V \in St(p,r)$. The Grassmann exponential map, representing the geodesic emanating from P in the direction Δ, is given by:*

$$Exp_P^{Gr}(t\Delta) = \left[UV \cos(t\Sigma) V^\top + \hat{Q} \sin(t\Sigma) V^\top + UV_\perp V_\perp^\top \right],$$

where $V_\perp \in \mathbb{R}^{p \times (p-r)}$ is any orthogonal complement of V.

proof. Using Grassmannina mathematics, we know that every $\Delta \in T_P Gr(n,p)$ is of the form

$$\Delta = Q \begin{pmatrix} 0 & B^\top \\ B & 0 \end{pmatrix} Q^\top = \left[Q \begin{pmatrix} 0 & -B^\top \\ B & 0 \end{pmatrix} Q^\top, P \right] \tag{34}$$

Then the lift of $\Delta \in T_P Gr(n,p)$ to $Q = (U \quad U_\perp)$ can also be calculated explicitly as follows:

$$\Delta_Q^{\text{hor}} = [\Delta, P] Q = Q \begin{pmatrix} 0 & -B^\top \\ B & 0 \end{pmatrix} \tag{35}$$

To resume our proof, we need to define the orthogonal group and specifying its tangent space.

Definition B.1 (Orthogonal Group). The orthogonal group $O(n)$ is defined as the set of all $n \times n$ matrices Q over $\mathbb{R}$ such that $Q^\top Q = QQ^\top = I_n$, where $Q^\top$ is the transpose of Q and I_n is the $n \times n$ identity matrix:

$$O(n) = \{Q \in \mathbb{R}^{n\times n} \mid Q^\top Q = I_n = QQ^\top\}.$$

Then the tangent space of the orthogonal group $O(n)$ at a point Q, denoted $T_Q O(n)$, is defined as the set of matrices of the form $Q\Omega$, where $\Omega \in \mathbb{R}^{n\times n}$ is a skew-symmetric matrix, i.e., $\Omega^\top = -\Omega$:

$$T_Q O(n) = \{Q\Omega \mid \Omega \in \mathbb{R}^{n\times n}, \Omega^\top = -\Omega\}.$$

The geodesic from $Q \in O(n)$ in direction $Q\Omega \in T_Q O(n)$ is calculated via

$$\mathrm{Exp}^O_Q(tQ\Omega) = Q \exp_m(t\Omega), \tag{36}$$

If $P \in Gr(n,p)$ and $\Delta \in T_P Gr(n,p)$ with $\Delta^{\text{hor}}_Q = Q\begin{pmatrix} 0 & -B^\top \\ B & 0\end{pmatrix}$, the geodesic in the Grassmannian is therefore

$$\mathrm{Exp}^{Gr}_P(t\Delta) = \pi^{OG}\left(Q \exp_m\left(t\begin{pmatrix} 0 & -B^\top \\ B & 0\end{pmatrix}\right)\right). \tag{37}$$

where π^{OG} is the projection from $O(n)$ to $Gr(n,p)$. If the thin SVD of B is given by

$$B = U_\perp^\top \Delta^{\text{hor}}_U = U_\perp^\top \hat{Q}\Sigma V^\top$$

with $W := U_\perp^\top \hat{Q} \in St(n-p,r), \Sigma \in \mathbb{R}^{r\times r}, V \in St(p,r)$. Let $W_\perp, V_\perp$ be suitable orthogonal completions. Then,

$$\exp_m\begin{pmatrix} 0 & -B^\top \\ B & 0\end{pmatrix} = \begin{pmatrix} V & V_\perp & 0 & 0 \\ 0 & 0 & W & W_\perp\end{pmatrix}\begin{pmatrix} \cos(\Sigma) & 0 & -\sin(\Sigma) & 0 \\ 0 & I_{p-r} & 0 & 0 \\ \sin(\Sigma) & 0 & \cos(\Sigma) & 0 \\ 0 & 0 & 0 & I_{n-p-r}\end{pmatrix}\begin{pmatrix} V^\top & 0 \\ V_\perp^\top & 0 \\ 0 & W^\top \\ 0 & W_\perp^\top\end{pmatrix},$$

which leads to the desired result when inserted into (37). For more mathematical details, you can refer to Edelman et al. [1998], Bendokat et al. [2024], or other useful resources on Grassmann geometry.

C Projection-Aware Optimizer

When projecting into a lower-dimensional space and tracking coordinate changes, we typically use orthonormal projection matrices to represent the subspaces and their multiplications to effect a change of basis. While this works well for purely linear operations, Adam's updates also incorporate non-linear elements.

Suppose the subspace changes at step t, transitioning from the subspace spanned by the orthonormal matrix S_{t-1} to that spanned by S_t. Since both matrices are orthonormal—preserved by the Grassmannian update rule in (5)—the matrix $S_t^\top S_{t-1}$ represents the change of basis between the two subspaces. In other words, if $\mathbb{E}_{t-1} = (e^1_{t-1}, \ldots, e^r_{t-1})$ and $\mathbb{E}_t = (e^1_t, \ldots, e^r_t)$ are orthonormal bases for the subspaces at steps $t-1$ and t, respectively, then the ith column of matrix X transforms under the change of basis as $X^i_t = \sum_{j=1}^r \langle e^i_t, e^j_{t-1}\rangle X^j_{t-1}$, where X^j_{t-1} is the jth column of X based on the basis of the time step $t-1$.

$\mathbf{E}_{t,\beta}\left[.\right]$ denotes the exponential time-weighted expectation at time t with decay rate β. Following the reinterpretation by Robert et al. [2025], Adam's first and second moment estimates can be expressed as $\widetilde{M}_t = \mathbf{E}_{t,\beta_1}\left[\widetilde{G}_t\right]$ and $\widetilde{\mathcal{V}}_t = \mathbf{E}_{t,\beta_2}\left[(\widetilde{G}_t)^2\right]$, where $\widetilde{G}_t$ denotes the low-rank representation of the gradient at time step t. As shown in (38), the first moment estimate can be transformed under a change of basis using the change-of-basis matrix, $S_t^\top S_{t-1}$. Notably, $\langle \widetilde{G}_t, e^i_t\rangle$ gives the ith column of $\widetilde{G}_t$ when the subspace has the basis $\mathbb{E}_t$. We use the superscripts to indicate a column of a matrix.

$$\mathbf{E}_{t,\beta_1}\left[\langle \widetilde{G}_t, e^i_t\rangle\right] = \sum_{j=1}^r \langle e^i_t, e^j_{t-1}\rangle \mathbf{E}_{t,\beta_1}\left[\langle \widetilde{G}_t, e^j_{t-1}\rangle\right] = \sum_{j=1}^r \langle e^i_t, e^j_{t-1}\rangle \widetilde{M}^j_t = \left(S_t^\top S_{t-1}\widetilde{M}_t\right)^i \tag{38}$$

Following the same approach, we can change the basis for the second moment estimate as described in (39).

$$
\begin{aligned}
\mathbf{E}_{t,\beta_2}\left[\left(\langle \widetilde{G}_t, e_t^i\rangle\right)^2\right] &= \sum_{j=1}^{r}\langle e_t^i, e_{t-1}^j\rangle^2\,\mathbf{E}_{t,\beta_2}\left[\left(\langle \widetilde{G}_t, e_{t-1}^j\rangle\right)^2\right] \\
&\quad + \sum_{k\neq l}^{r}\langle e_t^i, e_{t-1}^k\rangle\langle e_t^i, e_{t-1}^l\rangle\,\mathbf{E}_{t,\beta_2}\left[\langle \widetilde{G}_t, e_{t-1}^k\rangle\langle \widetilde{G}_t, e_{t-1}^l\rangle\right] \\
&= \sum_{j=1}^{r}\langle e_t^i, e_{t-1}^j\rangle^2\widetilde{\mathcal{V}}_t^j \\
&\quad + \sum_{k\neq l}^{r}\langle e_t^i, e_{t-1}^k\rangle\langle e_t^i, e_{t-1}^l\rangle\widetilde{M}_t^k\widetilde{M}_t^l.
\end{aligned}
\tag{39}
$$

In transitioning from the first equality to the second, we assume independence among the gradient coordinates. This enables us to approximate the covariance using a product of first-order moment estimates. This assumption is often reasonable in practice because we compute the SVD of the gradient and maintain an orthonormal subspace projection matrix, updating it along the Grassmannian geodesic to track the optimal subspace. Since SVD tends to diagonalize the covariance, the off-diagonal entries are typically negligible. Additionally, we clip any negative values to zero to ensure valid (non-negative) variance estimates. Moreover, to rewrite the second term in the final equality of (39), we employ the following equation:

$$
\begin{aligned}
&\sum_k\sum_l\langle e_t^i, e_{t-1}^k\rangle\langle e_t^i, e_{t-1}^l\rangle\widetilde{M}_t^k\widetilde{M}_t^l \\
&= \sum_k\langle e_t^i, e_{t-1}^k\rangle^2\left(\widetilde{M}_t^k\right)^2 \;+\; \sum_{k\neq l}\langle e_t^i, e_{t-1}^k\rangle\langle e_t^i, e_{t-1}^l\rangle\widetilde{M}_t^k\widetilde{M}_t^l
\end{aligned}
\tag{40}
$$

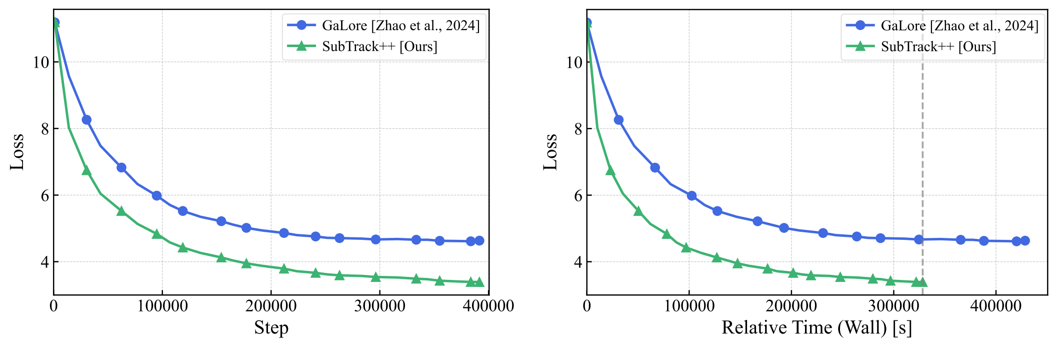
Given these, we can rewrite (39) as follows:

$$
\begin{aligned}
\mathbf{E}_{t,\beta_2}\left[\left\langle \widetilde{G}_t, e_t^i\right\rangle^2\right] &= \sum_j\langle e_t^i, e_{t-1}^j\rangle^2\widetilde{\mathcal{V}}_t^j + \\
&\left[\sum_k\sum_l\langle e_t^i, e_{t-1}^k\rangle\langle e_t^i, e_{t-1}^l\rangle\widetilde{M}_t^k\widetilde{M}_t^l - \sum_k\langle e_t^i, e_{t-1}^k\rangle^2\left(\widetilde{M}_t^k\right)^2\right] \\
&= \sum_j\langle e_t^i, e_{t-1}^j\rangle^2\left[\widetilde{\mathcal{V}}_t^j - \left(\widetilde{M}_t^j\right)^2\right] + \left(\langle e_t^i, e_{t-1}^j\rangle\widetilde{M}_t\right)^2 \\
&= \left(\left(S_t^\top S_{t-1}\right)^2\left[\widetilde{\mathcal{V}}_t - \widetilde{M}_t^2\right]\right)^i + \left(\left(S_t^\top S_{t-1}\widetilde{M}_t\right)^2\right)^i
\end{aligned}
\tag{41}
$$

By applying (38) and (41), we can directly derive the update rules for the projection-aware optimizer, as expressed in (8) and (9).

D Time Complexity Analysis

Table 3 presents the time complexity breakdown for the subspace update step in the SubTrack++ algorithm assuming a $m \times n$ gradient matrix and rank r projection matrix, where $r \ll m \leq n$. As outlined in Algorithm 1, the subspace update step begins by solving the least squares problem (2) to estimate the optimal update for S_t, the $m \times r$ orthonormal matrix. This operation has a time complexity of $O(mr^2)$. Computing the residual and the partial derivative with respect to S_t requires $O(mrn)$ and $O(mnr)$ time respectively. This is because the solution to the least squares problem, A, has shape $r \times n$ which is multiplied by S_t in the residual $R = G_t - S_tA$, resulting in time complexity $O(mrn)$. The following operation for the partial derivative is $-2RA^T$, where the matrix multiplication has $O(mnr)$ complexity. The tangent vector computation (4) which involves an identity transformation and matrix multiplication has time complexity of $O(m^2r)$.



(a) Loss vs. Training Steps.

(b) Loss vs. Relative Wall-Time.

Figure 7: Comparison of pre-training Llama-7B architecture for 100k iterations. (a) shows training loss (↓) versus training steps. (b) shows the same runs against wall-time. SubTrack++ outperforms GaLore; substantially reducing wall-time.

The rank-1 approximation step uses largest singular value from the SVD of the $m \times r$ tangent vector, and has time complexity of $O(mr^2)$. Finally, the update rule as shown in (13) which has a time complexity of $O(mr^2)$. The overall complexity of the algorithm is dominated by the matrix multiplication calculations of time complexity $O(mnr)$. However, unlike GaLore, since we avoid computing SVD operation on the $m \times n$ gradient matrix, which has complexity of $O(nm^2)$, the overall update step in SubTrack++ is still more efficient with respect to time complexity.

Table 3: Time Complexity for SubTrack++ Subspace Update

Computation Step	**Time**
Cost function	$O(mr^2)$
Residual	$O(mrn)$
Partial derivative	$O(mnr)$
Tangent vector ΔF	$O(m^2r)$
Rank-1 approximation of ΔF	$O(mr^2)$
Update rule	$O(mr^2)$
Overall	$O(mnr)$

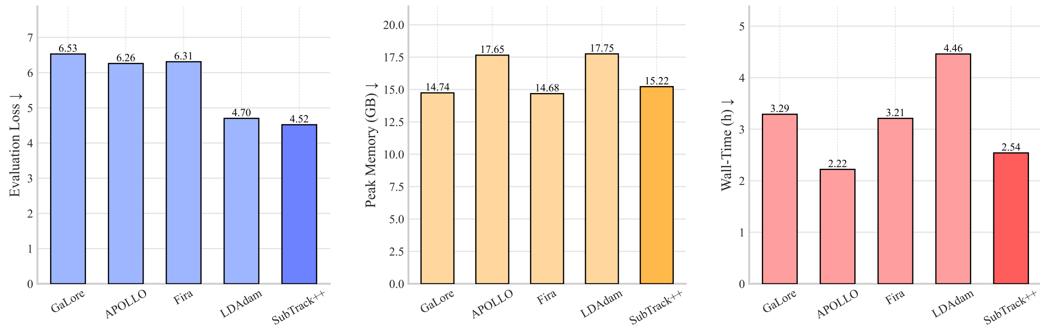
E Pre-Training Llama-Based Architectures

We pre-trained all six Llama-based architectures for 10k iterations using hyperparameters reported in Table 4. To demonstrate the generalizability of the proposed method, we also present results from pre-training the 7B architecture with both SubTrack++ and GaLore for 100k iterations, as shown in Figure 7. SubTrack++ maintains its advantage in terms of faster convergence and superior performance. The hyperparameters of this run are identical to those reported in Table 4, except for the number of iterations which is 100k. Also the bar-chart version of Figure 1 is represented in Figure 8.

F Memory and Time Comparison

Table 5 presents the the peak memory consumption measured to compare SubTrack++ and other baselines. It shows that SubTrack++ requires on-par or better memory compared to GaLore [Zhao et al., 2024] while getting state-of-the-art results. As detailed in Table 2, all baselines except BAdam [Luo et al., 2024] use the same number of optimizer parameters; therefore, any differences in their peak memory consumption stem from variations in their runtime parameters.

Additionally, Table 6 presents the wall-time consumed by each baseline across all model architectures during pre-training. Each run is configured to include exactly 10 subspace updates for SubTrack++ and other baselines employing periodic subspace updates. Specifically, models ranging from 60M to 3B use a subspace update interval of 200, resulting in 2,000 total iterations, while the 7B models use an interval of 500, yielding 5,000 iterations. Experiments for the 60M to 3B models are conducted on an NVIDIA A100 GPU, while the 7B model experiments are run on an NVIDIA RTX A6000.



(a) (b) (c)

Figure 8: We compare baselines on pre-training a 1B-parameter model. (a) SubTrack++ achieves the lowest evaluation loss across all methods. (b) Its peak memory usage is significantly lower than APOLLO and LDAdam, and on par with GaLore and Fira. (c) In terms of wall-time, SubTrack++ incurs minimal overhead relative to APOLLO and is markedly faster than GaLore, Fira, and LDAdam. Overall, SubTrack++ outperforms all baselines in evaluation loss while matching or exceeding them in memory and runtime efficiency.

Table 4: Hyperparameters of pre-training Llama-based architectures.

		60M	130M	350M	1B	3B	7B
Architectural Parameters	Hidden	512	768	1024	2048	2560	4096
	Intermediate	1376	2048	2736	5461	6848	11008
	Heads	8	12	16	24	32	32
	Layers	8	12	24	32	32	32
Shared Parameters	Learning Rate	1e-3	1e-3	1e-3	1e-4	1e-4	1e-4
	Batch Size	128	128	64	8	8	4
	Gradient Accumulation	2	2	2	2	2	4
	Iterations	10k					
	Gradient Clipping	1.0					
	Warmup Steps	1000					
	scale	0.25					
	dtype	bfloat16					
Low-Rank Optimizer Methods Parameters	Rank	128	256	256	512	512	1024
	Subspace Update Interval	200	200	200	200	200	500
	SubTrack++ Step-Size	10000					
BAdam Parameters	Block Swithch Interval	100					
	Switch Mode	Random					

G Fine-Tuning Experiments

As described, we examined SubTrack++ on fine-tuning RoBERTa-base and RoBERTa-large models to evaluate them on GLUE and SuperGLUE benchmarks. The results for GLUE task is summarized in Table 7, and SuperGLUE in 8.

The hyperparameters for fine-tuning RoBERTa-base are detailed in Table 10, matching those reported in the GaLore [Zhao et al., 2024] for rank-8 subspaces, with a subspace update interval set at 500 iterations. We also fine-tuned RoBERTa-Large on SuperGLUE tasks using the hyperparameters from Luo et al. [2024], as detailed in Table 11, with the exception that we fine-tuned each task for 30 epochs.

The results of supervised fine-tuning of the Llama-2-7B-chat-hf model on the Alpaca dataset on an Nvidia-H100 GPU are presented in Table 9. The fine-tuning was performed for one epoch, and the corresponding hyperparameters are listed in Table 12.

Table 5: Peak memory consumption of pre-training Llama-based architectures on C4 dataset. The 7B models are trained using the 8-bit Adam optimizer, except for the runs marked with *. SubTrack++ demonstrates better or on-par memory compared to other low-rank methods that allows full-parameter training.

	60M r=128	**130M** r=256	**350M** r=256	**1B** r=512	**3B** r=512	**7B** r=1024
Full-Rank	16.86	25.32	28.67	18.83	34.92	50.50
BAdam [Luo et al., 2024]	13.34	20.01	16.45	9.18	14.75	22.35
GaLore [Zhao et al., 2024]	16.89	25.52	27.85	14.74	26.03	36.00
Online Subspace Descent Liang et al. [2024]	16.61	25.86	28.76	18.45	32.57	33.10
LDAdam [Robert et al., 2025]	17.18	25.94	28.03	18.45	32.57	OOM*
Fira [Chen et al., 2025]	16.39	24.99	27.33	14.68	25.62	47.84*
SubTrack++ (Ours)	16.40	25.06	27.42	15.22	25.54	49.82*

Table 6: Wall-time comparison of pre-training Llama-based architectures on the C4 dataset. The number of iterations is set to ensure 10 subspace updates for methods using periodic subspace adjustments. SubTrack++ achieves the lowest wall-time among all baselines that support full-parameter training on large models. The 7B models are trained using the 8-bit Adam optimizer, except for the runs marked with *. Since this can impact wall-time comparisons, it is more appropriate to compare runs within the same cluster.

	60M r=128	**130M** r=256	**350M** r=256	**1B** r=512	**3B** r=512	**7B** r=1024
Full-Rank	524.0	1035.1	1396.4	974.9	1055.9	12726.2
BAdam [Luo et al., 2024]	511.3	779.2	961.6	798.6	1004.1	7283.7
GaLore [Zhao et al., 2024]	547.8	1094.2	1589.0	1729.5	2715.5	21590.4
Online Subspace Descent [Liang et al., 2024]	662.8	1228.2	1818.6	1438.7	1676.9	18221.9
LDAdam [Robert et al., 2025]	639.9	1342.2	2083.4	2780.9	4625.4	OOM*
Fira [Chen et al., 2025]	635.3	1180.8	1729.7	1938.5	2898.4	22554.3*
SubTrack++ (Ours)	627.6	1140.6	1593.2	1304.3	1517.6	16491.7*

Table 7: Evaluating the performance of SubTrack++ and other baselines when fine-tuning RoBERTa-Base on GLUE tasks for $r = 8$. The performance is measured via Accuracy (↑) for SST-2 and RTE tasks, F1 (↑) for MRPC, Pearson Correlation (↑) for STS-B, and Matthews Correlation (↑) for COLA. The best results are marked in **bold**, with the second-best performance <u>underlined</u>.

	COLA	**STS-B**	**MRPC**	**RTE**	**SST-2**
Full-Rank	62.57	91.03	91.32	77.98	94.27
BAdam [Luo et al., 2024]	54.44	89.01	91.35	68.59	94.15
GaLore [Zhao et al., 2024]	58.54	90.61	91.30	74.37	**94.50**
LDAdam [Robert et al., 2025]	**58.81**	<u>90.90</u>	**92.22**	<u>76.53</u>	<u>94.27</u>
SubTrack++ (Ours)	<u>58.55</u>	**90.95**	<u>92.04</u>	**78.34**	90.02

Table 8: Evaluating the performance of SubTrack++ and other baselines when fine-tuning RoBERTa-Large on SuperGLUE tasks with $r = 8$. The performance is measured via Accuracy (↑) for COPA, WIC, WSC, BoolQ, and AX_g tasks, and F1 (↑) for CB. The best results are marked in **bold**, with the second-best performance underlined.

	BoolQ	**CB**	**COPA**	**WIC**	**WSC**	**AX_g**
Full-Rank	85.96	90.33	76.00	71.79	63.46	96.30
GaLore [Zhao et al., 2024]	85.44	**88.85**	80.00	**71.47**	63.46	**100.00**
BAdam [Luo et al., 2024]	82.51	53.28	59.00	70.38	60.58	51.85
LDAdam Robert et al. [2025]	**85.75**	56.16	80.00	71.00	**64.42**	70.37
SubTrack++ (Ours)	85.38	83.96	**82.00**	70.70	62.5	**100.00**

Table 9: Comparing final evaluation loss (↓), wall-time, and memory of fine-tuning Llama-2-7B-chat-hf on Alpaca dataset. cd

Method	**Evaluation Loss**	**Wall-Time** (min)	**Memory** (GB)
GaLore [Zhao et al., 2024]	0.88	178	59.4
LDAdam [Robert et al., 2025]	0.85	342	66.1
SubTrack++ (Ours)	0.88	117	62.5

Table 10: Hyperparameters of fine-tuning RoBERTa-Base on GLUE tasks.

<table>
<tr><th></th><th></th><th>SST-2</th><th>MRPC</th><th>CoLA</th><th>RTE</th><th>STS-B</th></tr>
<tr><td rowspan="3">Shared Parameters</td><td>Batch Size</td><td>16</td><td>16</td><td>32</td><td>16</td><td>16</td></tr>
<tr><td># Epochs</td><td colspan="5">30</td></tr>
<tr><td>Max Seq. Len.</td><td colspan="5">512</td></tr>
<tr><td rowspan="5">Low-Rank Optimizer Methods Parameters</td><td>Learning Rate</td><td>2E-05</td><td>2E-05</td><td>1E-05</td><td>2E-05</td><td>3E-05</td></tr>
<tr><td>SubTrack Step-Size</td><td>0.1</td><td>3.0</td><td>5.0</td><td>15.0</td><td>10.0</td></tr>
<tr><td>Subspace Update Interval</td><td colspan="5">500</td></tr>
<tr><td>Rank Config</td><td colspan="5">8</td></tr>
<tr><td>α</td><td colspan="5">2</td></tr>
<tr><td rowspan="3">BAdam Parameters</td><td>Learning Rate</td><td>2E-05</td><td>2E-05</td><td>1E-05</td><td>2E-05</td><td>3E-05</td></tr>
<tr><td>Block Switch Interval</td><td colspan="5">100</td></tr>
<tr><td>Switch Mode</td><td colspan="5">Random</td></tr>
</table>

Table 11: Hyperparameters of fine-tuning RoBERTa-Large on SuperGLUE tasks.

<table>
<tr><th></th><th></th><th>BoolQ</th><th>CB</th><th>COPA</th><th>WIC</th><th>WSC</th><th>AX_g</th></tr>
<tr><td rowspan="4">Shared Parameters</td><td>Batch Size</td><td colspan="6">16</td></tr>
<tr><td># Epochs</td><td colspan="6">30</td></tr>
<tr><td>Learning Rate</td><td colspan="6">1e-5</td></tr>
<tr><td>Max Seq. Len.</td><td colspan="6">512</td></tr>
<tr><td rowspan="4">Low-Rank Optimizer Methods Parameters</td><td>SubTrack++ Step-Size</td><td>0.1</td><td>10.0</td><td>10.0</td><td>100.0</td><td>1.0</td><td>1.0</td></tr>
<tr><td>Subspace Update Interval</td><td>500</td><td>100</td><td>100</td><td>500</td><td>250</td><td>100</td></tr>
<tr><td>Rank Config.</td><td colspan="6">8</td></tr>
<tr><td>α</td><td colspan="6">4</td></tr>
<tr><td rowspan="2">BAdam Parameters</td><td>Block Switch Interval</td><td>100</td><td>50</td><td>50</td><td>100</td><td>50</td><td>50</td></tr>
<tr><td>Switch Mode</td><td colspan="6">Random</td></tr>
</table>

Table 12: Hyperparameters of fine-tuning Llama-2-7B-chat-hf on Alpaca dataset.

Parameter	Value
Subspace Update Interval	500
Rank	1024
α	0.25
Target Modules	att, mlp
Batch Size	8
Epoch	1

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We included the overview of method, achieved results, and the scope of experiments in abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: The limitations and future directions is included in the Discussion section. Also the computational complexity is reported.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical proofs are aligned with well-known prior works while showing improvement over them, and backed up by strong sources for Grassmannian manifolds geometry.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: All resources and hyperparameters used for testing the proposed method and baselines are provided in the supplemental material.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will include the code in the main paper after submission, and the anonymized repository in supplemental material of this version.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.

- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All the details regarding resources and hyperparameters for the proposed method and baselines are included in supplemental material.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: As the experiments of these paper are highly resource- and time-consuming, performing statistical test was not possible for this version. However, a variety of experiments are included to ensure generalization.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: All the necessary information is concluded in supplemental material.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: Yes, it totally conform.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: This is a foundational research, and further investigation for preserving safety is left for future work.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: It is a foundational research and do not poses such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We have cited properly.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The code that is included anonymously on supplemental material is documented.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This term does not apply to our research.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This term does not apply to our research.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: It does not apply to our research.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.