Memory-Efficient LLM Training by Various-Grained Low-Rank Projection of Gradients

Yezhen Wang [*1] **Zhouhao Yang** [*2] **Brian K. Chen** [1] **Fanyi Pu** [3] **Bo Li** [3] **Tianyu Gao** [1] **Kenji Kawaguchi** [1]

Abstract

Building upon the success of low-rank adapter (LoRA), low-rank gradient projection (LoRP) has emerged as a promising solution for memory-efficient fine-tuning. However, existing LoRP methods typically treat each row of the gradient matrix as the default projection unit, leaving the role of *projection granularity* underexplored. In this work, we propose a novel framework, **VLoRP**, that extends low-rank gradient projection by introducing an additional degree of freedom for controlling the trade-off between memory efficiency and performance, beyond the rank hyper-parameter. Through this framework, we systematically explore the impact of projection granularity, demonstrating that finer-grained projections lead to enhanced stability and efficiency even under a fixed memory budget. Regarding the optimization for VLoRP, we present **ProjFactor**, an adaptive memory-efficient optimizer, that significantly reduces memory requirement while ensuring competitive performance, even in the presence of gradient accumulation. Additionally, we provide a theoretical analysis of VLoRP, demonstrating the descent and convergence of its optimization trajectory under both SGD and ProjFactor. Extensive experiments are conducted to validate our findings, covering tasks such as commonsense reasoning, MMLU, and GSM8K.

1. Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across various domains (Achiam et al., 2023; Team et al., 2023; Touvron et al., 2023; Dubey et al., 2024; Li et al., 2024; Guo et al., 2025). However, these state-of-the-art models impose substantial memory requirements, making them challenging to finetune in practical settings. For example, fine-tuning an LLM with 7 billion parameters in the bfloat16 format (Dean et al., 2012; Abadi et al., 2016) requires approximately 14 GB of memory for the parameters alone. The gradients demand a similar amount of memory,[1] and, with Adam/AdamW (Kingma & Ba, 2014; Loshchilov & Hutter, 2019) serving as the de facto optimizer, an additional 28 GB is needed for optimizer states. Including the activations required for backpropagation, the total memory footprint exceeds 60 GB, rendering such training prohibitively costly for most users.

To mitigate this training overhead, various parameter-efficient fine-tuning (PeFT) techniques (Houlsby et al., 2019; Razdaibiedina et al., 2023) have been developed. Among the most notable are Low-Rank Adapters (LoRA) (Hu et al., 2022) and its variants (Dettmers et al., 2023; Liu et al., 2024; Hayou et al., 2024; Wang et al., 2024), which decompose matrix-shaped parameters into two low-rank matrices to enable more memory-efficient training. Recently, a novel family of methods, Low-Rank Gradient Projection (LoRP) (Hao et al., 2024; Zhao et al., 2024; Chen et al., 2024b; Zhu et al., 2024), has emerged. LoRP methods also leverage low-rank properties during LLM training, but rather than operating at the parameter level, they focus on exploiting the low-rank structure within the gradient. Concretely, LoRP methods achieve memory reduction by projecting the gradient matrix $G \in \mathbb{R}^{n\times m}$ into a low-dimensional subspace spanned by the columns of a projection matrix $P \in \mathbb{R}^{m\times r}$ with $r \ll m$. When required for parameter updates, the gradient is approximated as $(GP)P^\top$ by projecting the stored low-dimensional gradient GP back to the original space. In general, LoRP can offer better memory efficiency than LoRA, as it only requires the storage of gradient information in a single low-dimensional subspace, whereas LoRA necessitates two.

An interesting observation is that LoRP can also be interpreted from the perspective of stochastic approximation, specifically in the form of the forward gradient method (Baydin et al., 2022): when P is a random matrix sampled from

[*]Equal contribution [1]National University of Singapore [2]Johns Hopkins University [3]S-Lab, Nanyang Technological University. Correspondence to: Yezhen Wang <yezhenw@comp.nus.edu.sg>.

Preprint.

[1]Techniques like layer-wise updates can mitigate this overhead, but gradient accumulation still necessitates storing gradients in practice.

Gaussian distribution $\mathcal{N}(0, \frac{1}{r})$, LoRP can be viewed as an unbiased stochastic approximation of the original gradient, applied row-wisely (elaborated in Sec.3):

$$G_{i,:} \approx G_{i,:}PP^\top = \frac{1}{r}\sum_{j=1}^{r}\left(G_{i,:} \cdot v_j\right) v_j^\top, \tag{1}$$

where $i = 1, \ldots, n$, $v_j \sim \mathcal{N}(0, I_m)$ is the j-th column of $\sqrt{r}P$, and $G_{i,:}$ is the i-th row of G, representing a segment of the gradient being approximated. In (1), r serves a dual role: it defines the dimensionality of the projected space in LoRP and also determines the number of samples used for stochastic approximation, akin to the query count in Monte Carlo estimation. From the lens of stochastic approximation, the effectiveness of estimation is highly influenced simultaneously by the target dimensionality (*i.e.*, the size of $G_{i,:}$) and the number of samples r (Berahas et al., 2022; Shen et al., 2024). If r increases, more independent samples can reduce variance in (1). Alternatively, keeping r fixed while decreasing the dimensionality of $G_{i,:}$ improves accuracy by reducing the number of dimensions to estimate. This insight naturally leads to a new degree of freedom in LoRP methods: **rather than fixing the dimensionality of $G_{i,:}$, we can adjust it together with r to balance performance and memory efficiency.**

Building on the above observation, we introduce the concept **Projection Granularity** defined as the length of G's rows, which essentially represents the size of the fundamental projection unit in LoRP. In response, we propose **VLoRP** (**V**arious-Grained **L**ow-**R**ank **P**rojection of Gradients), a novel framework that simultaneously adjusts both the Projection Granularity and the rank r. By systematically exploring different configurations of granularity and rank in VLoRP, we find that, under a fixed memory budget, selecting a finer granularity, even with a smaller rank, consistently leads to superior performance. Additionally, by explicitly analyzing the mean and variance of gradient estimation, we establish that VLoRP achieves an $O(1/T)$ convergence rate when paired with the SGD optimizer.

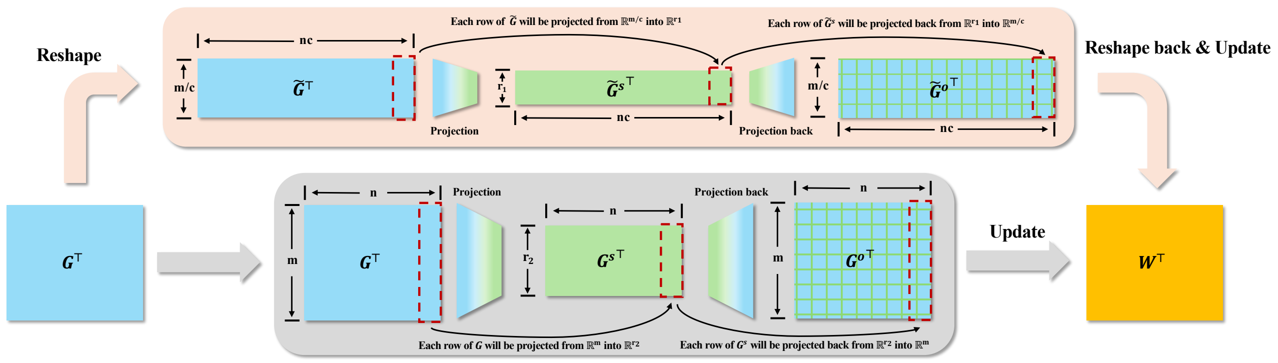
On top of it, given that Adam-based optimizers (Kingma & Ba, 2014; Shazeer & Stern, 2018; Loshchilov & Hutter, 2019) are the de facto standard for training modern LLMs, we also explore the potential adaptive training schemes for the VLoRP framework. Specifically, we investigate two distinct optimization schemes: the **Subspace Scheme (SS)** and the **Original Space Scheme (OS)**. In **SS**, optimization states are maintained within the low-dimensional subspace. In contrast, **OS** projects the low-dimensional gradients back first before storing and updating the optimization states within the original space. While both schemes only access the rank-r alternative of the original gradients, our results indicate that **OS** typically outperforms **SS**. However, since **OS** stores the optimizer states in the original full-rank space, its memory requirements can become prohibitive. To address this issue, we propose **ProjFactor**, a memory-efficient variant of **OS** that significantly reduces the memory usage for storing optimization states while maintaining comparable performance. Finally, we adopt Hamiltonian descent methods (Maddison et al., 2018; Chen et al., 2023; Liang et al., 2024) to show that, despite several approximations made as a compromise of memory efficiency, the Lyapunov function regarding ProjFactor can monotonically decrease with time, ensuring that the optimizer converges to a local optimum.

Our main contributions are as follows: (i) We introduce VLoRP, a method that facilitates the simultaneous adjustment of Projection Granularity and rank, thereby offering a more nuanced control over the trade-off between memory efficiency and performance. Besides, we argue that finer granularity is more significant than larger rank and validate it with comprehensive experiments. (ii) We investigate two Adam-based optimization schemes for the VLoRP framework and propose a novel adaptive optimization algorithm, named ProjFactor. This approach significantly reduces memory consumption while maintaining competitive performance. (iii) We present theoretical proof for the convergence of VLoRP with the SGD optimizer, achieving an $O(1/T)$ convergence rate. Additionally, we provide a rigorous theoretical guarantee for VLoRP under ProjFactor based on the framework of Hamiltonian descent.

2. Background

Low-Rank Gradient Projection (LoRP) Recently, LoRP algorithms (Hao et al., 2024; Zhao et al., 2024; Chen et al., 2024b; Zhu et al., 2024; Vyas et al., 2024) have become prominent PeFT methods. These methods leverage the observation that the gradients in high-dimensional parameter spaces often lie in a low-dimensional manifold. For instance, FLoRA (Hao et al., 2024) demonstrates that LoRA can be approximated by down-projecting gradients into a low-rank subspace and then up-projecting back using the same random projection matrix. Galore (Zhao et al., 2024) follows a similar spirit but derives the projection matrix through the Singular Value Decomposition (SVD) of the gradients. Chen et al. (2024b) extends Galore by incorporating the residual error between the full-rank gradient and its low-rank approximation, effectively simulating full-rank updates. APOLLO (Zhu et al., 2024) approximates channel-wise learning-rate scaling through an auxiliary low-rank optimizer state derived from random projections. However, existing LoRP approaches typically treat each row of the gradient matrix as the default projection unit, leaving the influence of varying Projection Granularity unexplored.

Forward Gradient (FG) Forward Gradient (FG) is a widely used stochastic approximation technique for gradient



Reshape
Each row of $\tilde{G}$ will be projected from $\mathbb{R}^{m/c}$ into $\mathbb{R}^{r_1}$
Each row of $\tilde{G}^s$ will be projected back from $\mathbb{R}^{r_1}$ into $\mathbb{R}^{m/c}$
Reshape back & Update
nc
m/c
$\tilde{G}^\top$
Projection
r_1
$\tilde{G}^{s\top}$
nc
Projection back
m/c
$\tilde{G}^{o\top}$
nc
$G^\top$
n
Projection
m
$G^\top$
n
r_2
$G^{s\top}$
Projection back
n
m
$G^{o\top}$
Update
$W^\top$
Each row of G will be projected from $\mathbb{R}^m$ into $\mathbb{R}^{r_2}$
Each row of G^s will be projected back from $\mathbb{R}^{r_2}$ into $\mathbb{R}^m$

Figure 1: Overview of VLoRP versus standard LoRP. *Bottom (gray)* : In ordinary LoRP, the gradient matrix G is directly projected row-by-row from $\mathbb{R}^{n\times m}$ into $\mathbb{R}^{n\times r}$ and stored as G^s. *Top (pale beige)*: In contrast, VLoRP reshapes the original gradient matrix G first to adjust the granularity of projection (from $\mathbb{R}^{n\times m}$ to $\mathbb{R}^{nc\times \frac{m}{c}}$), and during the update, the project-backed gradient $\tilde{G}^o$ would be reshaped back into $\mathbb{R}^{n\times m}$ to update the parameter $W \in \mathbb{R}^{n\times m}$.

estimation, particularly in scenarios where the direct computation of gradients is infeasible or expensive (Wengert, 1964; Silver et al., 2021; Baydin et al., 2022; Ren et al., 2022). Instead of computing exact gradients, FG estimates them by taking directional derivatives along multiple random perturbation directions. Formally, for a differentiable model with parameters $\theta \in \mathbb{R}^d$, FG approximates the gradient of the objective function $\mathcal{L}$ as:

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\nabla_\theta \mathcal{L}(\theta)^\top z_i\right) z_i, \tag{2}$$

where N is the number of random perturbation directions, and z_i denotes the i.i.d. sampled random vector.

More broadly, FG belongs to the family of stochastic approximation methods (Robbins & Monro, 1951; Nevel'son & Has' minskii, 1976; Spall, 1992), which iteratively estimate function properties based on noisy or partial observations. These methods are widely used in optimization and root-finding problems where exact computations are impractical.

For brevity, additional related work is deferred to Appendix E, and the notation list is provided in Appendix A.

3. Exploring Various Granularities of Low-Rank Gradient Projection

In this section, we provide a comprehensive understanding of LoRP methods and introduce VLoRP, which leverages the concept of Projection Granularity to optimize gradient projection. In Section 3.1, we reinterpret LoRP through the lens of stochastic approximation. In Section 3.2, we define Projection Granularity as a new degree of freedom in VLoRP, enabling a subtler control of the trade-off between memory efficiency and performance. Next, in Section 3.3, we show that finer Projection Granularity under a fixed "Memory Budget" improves performance, highlighting its importance over rank. Finally, in Section 3.4, we prove an $O(1/T)$ convergence rate for VLoRP with SGD, with theoretical proofs provided in Appendix C.

3.1. Understanding LoRPs through the Lens of Stochastic Approximation

Given the gradient matrix $G \in \mathbb{R}^{n\times m}$ and the projection matrix $P \in \mathbb{R}^{m\times r}$, LoRP approaches project the gradient into a low-dimensional subspace spanned by columns of projection matrix P, and project it back to the original space when gradient information is required for parameter update:

$$G^s := GP, \quad G^o := \gamma G^s P^\top = \gamma GPP^\top, \tag{3}$$

where we use G^s and G^o to represent the projected gradient in the subspace and the project-backed rank-r approximation of the original gradient respectively, and γ denotes the scaling coefficient and usually set as 1. In practice, only the matrix $G^s \in \mathbb{R}^{n\times r}$ needs to be stored with $r \ll \min\{m, n\}$. Particularly, if each element of P is i.i.d. sampled from Gaussian distribution $\mathcal{N}(0, \frac{1}{r})$, then G^o can be reformulated into

$$G^o = \frac{1}{r}\sum_{j=1}^{r} G v_j v_j \top = \begin{pmatrix} \frac{1}{r}\sum_{j=1}^{r}\left(G_{1,:}\cdot v_j\right) v_j^\top \\ \vdots \\ \frac{1}{r}\sum_{j=1}^{r}\left(G_{n,:}\cdot v_j\right) v_j^\top \end{pmatrix}, \tag{4}$$

where $v_j \sim \mathcal{N}(0, I_m)$ is the j-th column vector of $\sqrt{r}P$, and $G_{i,:} \in \mathbb{R}^{1\times m}$ is the i-th row of the gradient matrix G. (4) highlights that: (a) G^o can be represented as an average over r rank-one estimation, each involving one random direction v_j; (b) Each row $\frac{1}{r}\sum_{j=1}^{r}(v_j^\top G_{i,:}^\top)v_j^\top$ aligns with the formulation of the averaged forward gradients as in (2). Thus, we have the following observation:

Observation 3.1. *Equation* (4) *indicates that for each parameter matrix, LoRP can be interpreted as a row-wise application of forward gradient estimation with* r *samples.*

In general, when training an LLM parametrized by θ with an objective function $\mathcal{L}$, the stochastic approximation of the gradient can be applied at different levels. Consider two extreme cases: (1) We can flatten the entire parameter set θ into a single vector $\theta' \in \mathbb{R}^D$, where D is the total number of parameters, and estimate the gradient using a single random vector $v \in \mathbb{R}^D$. This approach aligns with the spirit of the MeZO algorithm (Malladi et al., 2023)[2]; (2) Alternatively, the estimation can be applied element-wise: for each 1-dimensional parameter $\xi \in \theta$, the coordinate partial derivative $\nabla_\xi \mathcal{L}$ can be approximated as $\frac{1}{r'} \sum_{i=1}^{r'} v_i \nabla_\xi \mathcal{L} v_i$, where $v_i \sim \mathcal{N}(0, 1)$. Concatenating all such approximations yields an estimate of the full gradient matrix G, resembling the coordinate gradient estimation (CGE) approach (Chen et al., 2024a).

These examples illustrate that stochastic approximation can be applied to parameters of varying sizes, leading to different methods. Naturally, according to Observation 3.1, the Projection Granularity, defined by the row size of G, can also be adjusted in LoRPs. This insight motivates our exploration of low-rank gradient projections with varying granularities.

3.2. Various-Grained Low-Rank Projection of Gradient

In this section, we present the details of our framework, termed **VLoRP**, which introduces a novel degree of freedom—namely, the Projection Granularity—beyond the conventional hyperparameter rank to balance memory efficiency and training performance in LoRP approaches.

The core idea is straightforward: because gradient projection is typically applied row-wisely, one can reshape the gradient matrix to modify the row size and correspondingly adjust the shape of the projection matrix generated, leading to the adjustment of the Projection Granularity naturally. Concretely, given an LLM, we introduce the **granularity factor** c, which can reshape any gradient $G \in \mathbb{R}^{n\times m}$ into $\tilde{G} \in \mathbb{R}^{nc\times(m/c)}$. Analogous to the rank r, which globally sets the projection rank for all matrix-shaped parameters, c similarly serves as a global hyperparameter that controls the granularity of projection. Then, we formally have[3]:

$$\begin{aligned} \tilde{G}^s &:= \underbrace{\mathrm{Reshape}\left(G, [nc, \tfrac{m}{c}]\right)}_{\text{denoted as } \tilde{G}} \tilde{P}, \\ G^o &:= \mathrm{Reshape}\Big(\underbrace{\tilde{G}^s \tilde{P}^\top}_{\text{denoted as } \tilde{G}^o}, [n, m]\Big), \end{aligned} \tag{5}$$

[2]MeZO uses the central difference to estimate the JVP in (2).

[3]with a little abuse of notations, G^o in (3) is now redefined in (5), as they both represent the rank-r estimation of G.

where the projection matrix $\tilde{P}$ is of shape $\frac{m}{c} \times r$, with each element i.i.d. sampled from $\mathcal{N}(0, \frac{1}{r})$. Under this setting, (3) emerges as a special case with $c = 1$. Decreasing $c < 1$ horizontally compresses G, thereby coarsening the Projection Granularity, whereas increasing $c > 1$ does the opposite. In practice, c is chosen to be a power of two for implementation convenience, and both $\frac{m}{c}$ and nc must be integers.

The overall framework is illustrated on the left of Figure 1: at each iteration, VLoRP first reshapes the original gradient matrix G (from $\mathbb{R}^{n\times m}$ to $\mathbb{R}^{nc\times(m/c)}$) to adjust the Projection Granularity. The reshaped $\tilde{G}$ is then projected into the subspace to obtain $\tilde{G}^s$ which needs to be stored. During the parameter update, $\tilde{G}^s$ will be projected back to the original space to obtain $\tilde{G}^o$, after which we reshape $\tilde{G}^o$ back to update the parameter. Notably, in practice, the only but important difference between VLoRP and other LoRP methods is the pair of reshaping operations, underscoring its simplicity of implementation.

3.3. Benefits of Finer Projection Granularity

Fundamentally, VLoRP introduces a novel degree of freedom that extends beyond the rank parameter in LoRP approaches. This raises a critical question: *What level of the Projection Granularity is optimal for gradient projection?*

To address this question, it is essential to establish a fair basis for comparison first: on the one hand, for a fixed rank r, employing a finer-grained projection typically improves performance but increases memory requirements; on the other hand, for a fixed granularity factor c, increasing the rank similarly enhances performance while incurring additional memory costs. We thereby denote a specific pair of (c, r) as a "configuration" of VLoRP, and define the "Memory Budget" $\mathcal{M}$ as the product of c and r—for configurations sharing the same $\mathcal{M}$, the size of $\tilde{G}^s$, which is needed to be stored, is the same. Based on this, we claim that **for all configurations sharing the same memory budget, *i.e.*,** $c_i r_i = c_j r_j = \mathcal{M}, \forall i, j$, **opting for a finer-grained projection (larger** c**) is generally preferable, despite the associated reduction in rank** ($r = \frac{\mathcal{M}}{c}$). There are several immediate advantages:

Computational Efficiency Given the granularity factor c, the memory budget $\mathcal{M}$ and the reshaped gradient $\tilde{G} \in \mathbb{R}^{nc\times(m/c)}$, the gradient projection operation $\tilde{G}\tilde{P}$ leads to a computational complexity $O\left(\frac{nm\mathcal{M}}{c}\right)$. By choosing a finer granularity (larger c), we can reduce the overall FLOPs involved.

Generation Efficiency of Projection Matrix Obviously, the efficiency of this generation process depends on the dimensions of the projection matrix $\tilde{P}$ of the shape $\frac{m}{c} \times \frac{\mathcal{M}}{c}$. As c increases, the projection granularity becomes finer, leading to a quadratic reduction in the size of the generated matrix and hence more efficient generation.

Numerical Error As c increases, the numerical accuracy of the random projection tends to improve. This can be attributed to the interplay between matrix dimensions and floating-point arithmetic properties. A larger c reduces the number of columns in $\tilde{G}$, thereby decreasing the number of floating-point operations per dot product in matrix multiplications. Consequently, this helps mitigate the accumulation of numerical errors, which is particularly important in low-precision training scenarios such as float16 or bfloat16, where limited mantissa precision can lead to instability. We provide an analytical experiment in Appendix D.3 to support this observation. Furthermore, this numerical stability advantage is further amplified when using gradient accumulation or Adam-based optimizers, as these methods involve iteratively accumulating optimization states.

Besides, our experiments in Section 5 demonstrate that under the same memory budget $\mathcal{M}$, finer-grained projections can consistently outperform coarser counterpart, suggesting that c play a more critical factor than rank r.

3.4. Theoretical Guarantee

Incorporating Projection Granularity into gradient estimation offers clear benefits, but it is crucial to ensure that finer-grained projections do not compromise the convergence of model training. This section provides theoretical guarantees that under a fixed memory budget $\mathcal{M}$, varying the granularity factor c and rank r does not significantly affect the mean and variance of one-step gradient estimation or the overall convergence rate.

We consider a loss function $\mathcal{L} : \mathbb{R}^{n\times m} \to \mathbb{R}$ defined over matrix parameters $W \in \mathbb{R}^{n\times m}$. Fix a memory budget $\mathcal{M}$, a granularity factor c, rank r, and $G = \nabla_W \mathcal{L}(W)$. Throughout this paper, we adopt the Frobenius norm and inner product as the primary metrics for matrices and vectors.

Proposition 3.2. *The gradient estimator $G^o \in \mathbb{R}^{n\times m}$ satisfies the following properties:*

$$\mathbb{E}[G^o] = G, \quad \mathbb{E}\|G^o - G\|^2 = \frac{m+c}{\mathcal{M}}\|G\|^2.$$

Proposition 3.2 demonstrates that the estimator G^o is unbiased and its variance is bounded by $O\left(\frac{m+c}{\mathcal{M}}\right)$. For LLMs, where the granularity factor c is significantly smaller than the parameter dimension m, the effect of altering c on gradient approximation is minimal under a fixed memory budget $\mathcal{M}$. The unbiasedness and bounded variance of G^o lead to the following convergence guarantee:

Theorem 3.3. *Let $\mathcal{L}$ be an L-smooth function with respect to the matrix-shaped parameter W. Assume the parameter updates are given by $W_{t+1} = W_t - \eta\, G_t^o$, where the step size is defined as $\eta = \frac{\mathcal{M}}{(m+c+\mathcal{M})L} := C$. Then, for any $T \geq 1$:*

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\big[\|G_t\|^2\big] \leq \frac{2C}{T}\big(\mathcal{L}(W_0) - \mathcal{L}(W^*)\big),$$

where W^* is a global minimizer of $\mathcal{L}$.

Theorem 3.3 confirms that using random-projection-based gradient estimation with VLoRP in conjunction with SGD achieves an $O(1/T)$ convergence rate, regardless of the granularity factor c.

4. Adaptive Memory-Efficient Optimization

In this section, we first investigate potential Adam-based optimization schemes for VLoRP, then introduce a memory-efficient variant, **ProjFactor**, for the superior scheme, and finally provide theoretical convergence proof for it. We enable gradient accumulation (Wang et al., 2013; Smith et al., 2018) for the projected gradient by default, which means at each update stage, we can only access $\tilde{\boldsymbol{G}}^s = \sum_{i=1}^K \tilde{G}_i^s / K$, where K is the number of accumulation substeps.

Optimization Schemes: SS vs. OS Broadly, with access only to $\tilde{\boldsymbol{G}}^s$, there are two typical schemes for storing the optimization states of Adam and performing the associated updates for VLoRP, as illustrated on the left of Figure 2: (1) **Subspace Scheme (SS)** retains the optimization states m^s and v^s and computes the adapted gradient $\tilde{\Delta}_t^s = \tilde{m}_t^s / \sqrt{\tilde{v}_t^s}$ within the subspace, while (2) **Original Space Scheme (OS)** projects $\tilde{\boldsymbol{G}}^s$ back to the original space, where the optimization states $\tilde{m}^o$ and $\tilde{v}^o$ are stored and the adapted gradient is computed directly. Mathematically, the update rules for both schemes are defined as follows[4]:

$$\begin{aligned}
\textbf{SS:}\quad & \tilde{m}_t^s = \beta_1 \tilde{m}_{t-1}^s + (1-\beta_1)\tilde{\boldsymbol{G}}_t^s,\\
& \tilde{v}_t^s = \beta_2 \tilde{v}_{t-1}^s + (1-\beta_2)(\tilde{\boldsymbol{G}}_t^s)^{\odot 2},\\
& \tilde{\Delta}_t^s = \tilde{m}_t^s / \sqrt{\tilde{v}_t^s},\\
& W_t = W_{t-1} - \eta\,\mathrm{Reshape}\left(\tilde{\Delta}_t^s \tilde{P}^\top, [n,m]\right);\\
\textbf{OS:}\quad & \tilde{m}_t^o = \beta_1 \tilde{m}_{t-1}^o + (1-\beta_1)(\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top),\\
& \tilde{v}_t^o = \beta_2 \tilde{v}_{t-1}^o + (1-\beta_2)(\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top)^{\odot 2},\\
& \tilde{\Delta}_t^o = \tilde{m}_t^o / \sqrt{\tilde{v}_t^o},\\
& W_t = W_{t-1} - \eta\,\mathrm{Reshape}\left(\tilde{\Delta}_t^o, [n,m]\right).
\end{aligned} \tag{6}$$

OS Performs Better From (6), it can be observed that for momentum-based SGD optimizers, where the second moment v is excluded, the **Subspace Scheme (SS)** and **Original Space Scheme (OS)** are equivalent, since $\tilde{m}_t^o = \tilde{m}_t^s \tilde{P}^\top$. However, when the nonlinear second moment is incorporated, **SS** and **OS** become different. An intuition is

[4] $\odot 2$ denotes element-wise squaring.

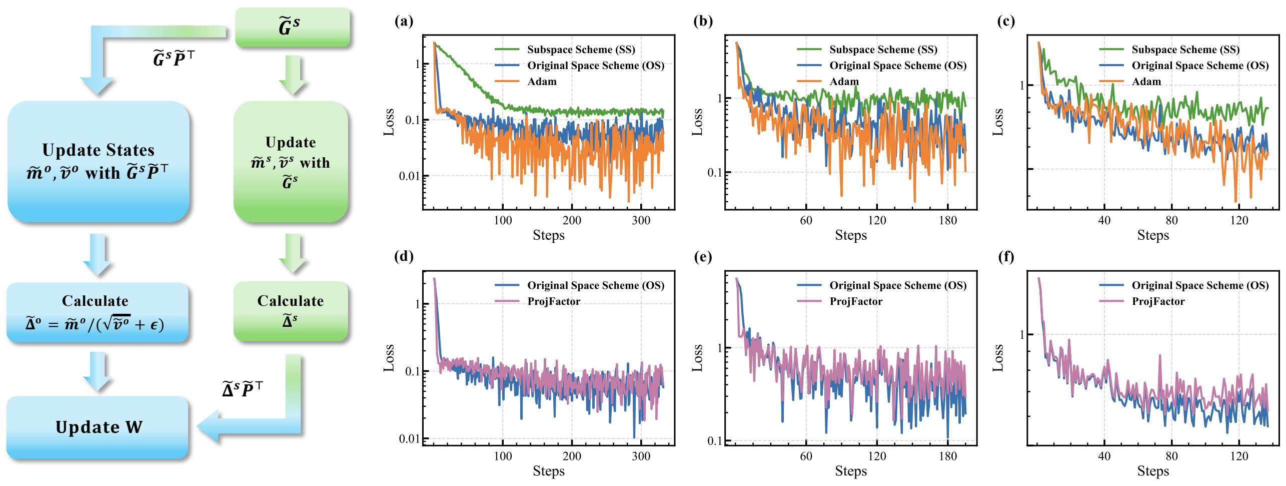


Figure 2: **Left:** Schematic illustration of the Subspace Scheme (SS, green) operating in a learned subspace, and the Original Scheme (OS, blue) operating in the original space. **Right (top row, panels a–c):** Fine-tuning loss curves of SS (green), OS (blue), and Adam (orange) on three tasks, showing that OS outperforms SS by a large margin while has a comparable performance with Adam. **Right (bottom row, panels d–f):** Comparison of OS (blue) and its approximate algorithm ProjFactor (purple), indicating that ProjFactor closely approximates the dynamic of OS. Specifically, columns (a) and (d) test on the Commonsense Reasoning task, columns (b) and (e) test on the MMLU, and columns (c) and (f) test on the GSM8K.

that the dynamics of **OS** are closer to Adam's, as (1) both **OS** and Adam adapt the gradient directly in the original space, and (2) previous works (Zhao et al., 2024; Hao et al., 2024) have shown that the gradient of LLMs exhibits a low-rank structure, implying that $\tilde{G}\tilde{P}\tilde{P}^\top$ is capable of preserving the primary information of $\tilde{G}$. To substantiate it, we finetune a LLaMA2-7B on three different benchmarks and compare the loss curves of **SS**, **OS**, and the vanilla Adam. As depicted in Figure 2 (right, top rows, panels a-c), while all three schemes reduce the loss, **SS** exhibits a noticeably slower and less effective convergence compared to **OS** and Adam. In contrast, **OS** closely tracks the loss curve of Adam.

ProjFactor: a Memory-Efficient Optimizer for VLoRP While **OS** achieves superior performance, it generally requires more memory during training compared to **SS**, as it does not reduce the memory usage of optimization states, which still occupy $O(nm)$ space. To address this, we propose **ProjFactor** which (1) maintains $\tilde{m}^s$ instead of $\tilde{m}^o$ in the subspace and project it back to the original space when calculating Δ_t^o, which is justified by the fact $\tilde{m}^o = \tilde{m}^s\tilde{P}^\top$; (2) follows the spirit of Adafactor (Shazeer & Stern, 2018) by applying rank-1 decomposition on $(\tilde{G}_t^s\tilde{P}^\top)^{\odot 2}$, which only requires the storage of two vectors, significantly reducing memory consumption while preserving effective second-moment estimates. Compared to Adafactor, which applies the rank-1 decomposition directly on the gradient, our method applies the rank-r approximation of $\tilde{G}$ first before the squaring and decomposition. The final algorithm is outlined in Appendix B. Since Adafactor can achieve performance comparable to Adam (Shazeer & Stern, 2018; Hao et al., 2024), it is reasonable to conjecture that ProjFactor can similarly match **OS**. We showcase the performance comparison between ProjFactor and **OS** in Figure 2(d-f), which aligns with our expectations.

Convergence Analysis To analyze ProjFactor's convergence, we adopt the Hamiltonian descent framework, following the line of work (Maddison et al., 2018; Chen et al., 2023; Liang et al., 2024; Nguyen et al., 2024). The infinitesimal updates of Projfactor is defined as follows:

$$\begin{aligned}
&\frac{d}{dt}\tilde{m}_t^s = a(\tilde{G}_t^s - \tilde{m}_t^s);\ \tilde{v}_t^o = \frac{\tilde{v}_{rt}^o\tilde{v}_{ct}^o}{\mathbf{1}_n^T\tilde{v}_{rt}^o};\\
&\frac{d}{dt}\tilde{v}_{rt}^o = b((\tilde{G}_t^o)^{\odot 2}\mathbf{1}_m - \tilde{v}_{rt}^o);\\
&\frac{d}{dt}\tilde{v}_{ct}^o = b(\mathbf{1}_n^T(\tilde{G}_t^o)^{\odot 2} - \tilde{v}_{ct}^o);\\
&\frac{d}{dt}W_t = \text{Reshape}\left(-\tilde{m}_t^s\tilde{P}^\top\Big/\sqrt{\tilde{v}_t^o},\ [n,m]\right),
\end{aligned} \tag{7}$$

where a, b are constants. The corresponding Lyapunov function (Hamiltonian) is defined as

$$\mathcal{H}(W, \tilde{m}^s, \tilde{v}_r^o, \tilde{v}_c^o) = \mathcal{L}(W) + \frac{1}{2a}\left\langle \tilde{m}^s, \frac{\tilde{m}^s}{\sqrt{\tilde{v}^o}}\right\rangle.$$

Then we have the following convergence guarantee:

Theorem 4.1. *Suppose the functions in system (7) are continuously differentiable. Under mild assumptions, we have*

(1) For $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)$ satisfying (7),

$$\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) \leq 0.$$

Table 1: Performance Comparison on Commonsense Reasoning. All models are first finetuned on the Commonsense170k (Hu et al., 2023a) dataset and then evaluated separately on 8 reasoning tasks. We set the Memory Budget $\mathcal{M} = 256$ for VLoRP, *i.e.*, the product of c and r equal 256. For a fair comparison, we also set the rank of all other low-rank-based methods as 256.

Methods	**ARC_C**	**ARC_E**	**BoolQ**	**HellaSwag**	**OBQA**	**PIQA**	**SIQA**	**winogrande**	**Avg.**
Pretrain-Untuned	42.26 $\pm$ 1.45	75.26 $\pm$ 0.87	76.86 $\pm$ 0.73	56.17 $\pm$ 0.49	30.40 $\pm$ 2.08	77.91 $\pm$ 0.97	45.11 $\pm$ 1.13	58.78 $\pm$ 1.30	58.84
Adam	47.12 $\pm$ 1.46	78.55 $\pm$ 0.83	82.27 $\pm$ 0.65	56.08 $\pm$ 0.49	33.60 $\pm$ 2.13	77.45 $\pm$ 0.96	52.53 $\pm$ 1.13	71.69 $\pm$ 1.25	62.41
Adafactor	48.06 $\pm$ 1.46	79.22 $\pm$ 0.82	80.50 $\pm$ 0.68	56.24 $\pm$ 0.49	34.20 $\pm$ 2.14	77.56 $\pm$ 0.96	52.02 $\pm$ 1.13	71.22 $\pm$ 1.26	62.38
LoRA$(r = 256)$	44.23 $\pm$ 1.46	76.66 $\pm$ 0.85	80.21 $\pm$ 0.68	55.79 $\pm$ 0.49	33.00 $\pm$ 2.13	76.57 $\pm$ 0.97	47.94 $\pm$ 1.13	69.69 $\pm$ 1.27	60.51
Galore$(r = 256)$	44.13 $\pm$ 1.45	76.65 $\pm$ 0.87	78.23 $\pm$ 0.72	57.59 $\pm$ 0.49	32.00 $\pm$ 2.09	77.95 $\pm$ 0.97	46.01 $\pm$ 1.13	69.71 $\pm$ 1.29	60.28
fira$(r = 256)$	44.06 $\pm$ 1.45	76.63 $\pm$ 0.87	78.12 $\pm$ 0.73	57.69 $\pm$ 0.49	32.40 $\pm$ 2.09	77.78 $\pm$ 0.97	46.21 $\pm$ 1.13	69.89 $\pm$ 1.29	60.35
APOLLO$(r = 256)$	44.28 $\pm$ 1.45	76.26 $\pm$ 0.87	77.74 $\pm$ 0.73	57.00 $\pm$ 0.49	31.40 $\pm$ 2.08	77.97 $\pm$ 0.97	46.11 $\pm$ 1.13	69.46 $\pm$ 1.29	60.03
VLoRP									
- $c = 2^{-6}$, $r = 2^{14}$	42.92 $\pm$ 1.45	76.22 $\pm$ 0.87	79.27 $\pm$ 0.71	57.53 $\pm$ 0.49	32.60 $\pm$ 2.10	77.91 $\pm$ 0.97	46.72 $\pm$ 1.13	69.85 $\pm$ 1.29	60.38
- $c = 2^{-4}$, $r = 2^{12}$	43.34 $\pm$ 1.45	76.26 $\pm$ 0.87	79.54 $\pm$ 0.71	57.58 $\pm$ 0.49	32.00 $\pm$ 2.09	77.64 $\pm$ 0.97	46.72 $\pm$ 1.13	70.01 $\pm$ 1.29	60.39
- $c = 2^{-2}$, $r = 2^{10}$	43.34 $\pm$ 1.45	76.30 $\pm$ 0.81	79.45 $\pm$ 0.71	57.47 $\pm$ 0.49	32.20 $\pm$ 2.09	77.75 $\pm$ 0.97	46.78 $\pm$ 1.13	70.01 $\pm$ 1.29	60.41
- $c = 2^{0}$, $r = 2^{8}$	43.69 $\pm$ 1.45	77.02 $\pm$ 0.86	79.27 $\pm$ 0.71	57.49 $\pm$ 0.49	31.80 $\pm$ 2.08	78.07 $\pm$ 0.97	47.49 $\pm$ 1.13	69.77 $\pm$ 1.29	60.57
- $c = 2^{2}$, $r = 2^{6}$	44.03 $\pm$ 1.45	76.81 $\pm$ 0.87	79.17 $\pm$ 0.71	57.59 $\pm$ 0.49	31.80 $\pm$ 2.08	78.02 $\pm$ 0.97	47.19 $\pm$ 1.13	69.53 $\pm$ 1.29	60.53
- $c = 2^{4}$, $r = 2^{4}$	44.71 $\pm$ 1.45	77.27 $\pm$ 0.86	79.42 $\pm$ 0.71	57.50 $\pm$ 0.49	32.20 $\pm$ 2.09	77.86 $\pm$ 0.97	47.54 $\pm$ 1.13	70.09 $\pm$ 1.29	60.82
- $c = 2^{6}$, $r = 2^{2}$	44.97 $\pm$ 1.45	77.65 $\pm$ 0.85	80.46 $\pm$ 0.69	57.56 $\pm$ 0.49	33.60 $\pm$ 2.11	77.97 $\pm$ 0.97	48.06 $\pm$ 1.13	69.69 $\pm$ 1.29	61.25
$c = 2^{8}$, $r = 2^{0}$	45.56 $\pm$ 1.46	77.78 $\pm$ 0.85	80.58 $\pm$ 0.69	57.59 $\pm$ 0.49	34.00 $\pm$ 2.12	77.86 $\pm$ 0.97	48.16 $\pm$ 1.13	69.69 $\pm$ 1.29	61.40

(2) Any bounded solution $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)_t$ of (7) converges to a stationary point of $\mathcal{L}(W)$ as $t \to \infty$.

Theorem 4.1 presents a Hamiltonian interpretation of Projfactor, indicating that the Lyapunov function, namely, the "energy" of the system, decreases monotonically over time. In addition, it ensures that ProjFactor stabilizes at a local optimum, provided the step sizes are sufficiently small. We defer the detailed assumption and proof to Appendix C.3.

5. Experiments

In this section, we evaluate the effectiveness of VLoRP across multiple finetuning tasks, demonstrating its competitiveness with state-of-the-art baselines. More experimental studies and implementation details can be found in Appendix D.

Datasets We present a comprehensive evaluation of our proposed approaches using three benchmarks: (1) *Commonsense Reasoning*, which covers 8 reasoning tasks including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), Hellaswag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-e (Clark et al., 2018), ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018); (2) The *MMLU* benchmark (Hendrycks et al., 2020), which encompasses a wide range of subjects including Humanities, STEM, Social Sciences, and Other fields; (3) Finally, the *GSM8K* dataset (Cobbe et al., 2021), which is a dataset of 8.5K high-quality problems of mathematics.

Baselines We compare VLoRP with 7 state-of-the-art baselines, covering full-parameter finetuning, LoRA, and LoRP methods. Specifically, we compare with **Pretrain-Untuned**, which represents the basic performance of the pre-trained model without finetuning, **Adam** (Kingma & Ba, 2014), **Adafactor** (Shazeer & Stern, 2018), **LoRA** (Hu et al., 2022), **Galore** (Zhao et al., 2024), **fira** (Chen et al., 2024b), and **APOLLO** (Zhu et al., 2024).

Experimental Settings We use the LLaMA2-7B model with the bfloat16 data type as the primary testbed for all methods. Each method is guaranteed that every token in the training set is encountered at least once. Gradient accumulation and activation checkpointing (Chen et al., 2016) are enabled. VLoRP is optimized with ProjFactor by default.

Results and Analysis We report the performance of all methods on: (1) commonsense reasoning tasks in Table 1, (2) the MMLU benchmark in Table 2, and (3) the GSM8k task in Figure 3. In general, all finetuning methods yield significant performance improvements compared to the untuned model. Among the finetuning baselines, Adam and Adafactor generally achieve the highest performance across tasks. Notably, Adafactor delivers comparable even superior results to Adam, consistent with previous findings (Shazeer & Stern, 2018; Hao et al., 2024). Furthermore, the proposed VLoRP, optimized with ProjFactor, not only rivals the performance of the strongest baselines of efficient finetuning but in many cases surpasses them. On top of it, under a fixed memory budget, configurations of VLoRP with finer Projection Granularity (*i.e.*, larger factor c albeit smaller r) tend to achieve higher average accuracy. For instance, the finest-grained VLoRP configuration $\left(c = 2^8, r = 2^0\right)$ achieves the highest scores of 61.40, 55.83, and 29.42 on Commonsense Reasoning, MMLU, and GSM8k, respectively, among all tested configurations of c and r with the same memory budget. This suggests that Projection Granularity may play a more critical role than rank in balancing memory efficiency and performance.

Table 2: Performance Comparison on the MMLU benchmark. We also set the Memory Budget $\mathcal{M} = 256$ for VLoRP. The best-performing configurations are highlighted.

Methods	Hum.	STEM	S. Sci.	Other	ALL
Pretrain-Untuned	39.54	34.85	49.14	47.73	42.53
Adam	52.63	44.92	65.31	62.45	56.01
Adafactor	53.01	46.55	66.13	56.87	56.80
LoRA ($r = 256$)	50.74	43.61	60.93	59.73	53.55
Galore ($r = 256$)	50.22	42.61	60.25	59.52	52.93
fira ($r = 256$)	49.85	42.49	60.31	59.55	52.83
APOLLO ($r = 256$)	49.12	41.42	57.71	56.91	51.13
VLoRP					
- $c = 2^{-6}$, $r = 2^{14}$	47.06	39.54	56.33	56.71	49.65
- $c = 2^{-4}$, $r = 2^{12}$	49.88	42.50	58.44	58.69	52.22
- $c = 2^{-2}$, $r = 2^{10}$	50.04	41.21	58.74	58.63	52.01
- $c = 2^{0}$, $r = 2^{8}$	50.62	43.23	61.08	60.83	53.64
- $c = 2^{2}$, $r = 2^{6}$	50.93	44.31	61.04	60.51	53.91
- $c = 2^{4}$, $r = 2^{4}$	50.85	44.22	61.08	60.91	54.05
- $c = 2^{6}$, $r = 2^{2}$	51.94	44.53	63.53	62.85	55.42
$c = 2^{8}$, $r = 2^{0}$	52.29	46.18	64.05	62.24	55.83

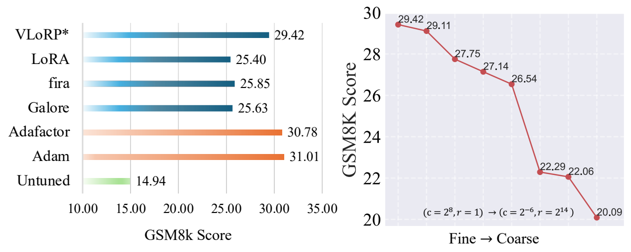


Figure 3: **Left:** Performance comparison of different methods on GSM8K. **Right:** Performance comparison among the configurations of VLoRP with $\mathcal{M} = 256$. The x-axis indicates configurations from fine to coarse (left to right).

Different Memory Budgets Besides, in Figure 4, we present the performance of VLoRP under different projection granularities across varying memory budgets. Overall, the results indicate that configurations with finer projection granularities consistently outperform other coarser configurations, regardless of the memory budget level. Furthermore, when the rank is fixed (the three subcolumns within each column), the performance improves with finer granularities.

Memory Analysis We illustrate the memory usage of various methods in Figure 5, emphasizing that VLoRP achieves the most significant memory reduction compared to baseline approaches. When incorporating gradient accumulation, certain memory-efficient methods, such as fira and APOLLO, fail to achieve memory usage below Adafactor. This limitation stems from their reliance on the full-rank gradient for update computations, which primarily reduces the memory associated with optimization states but not the gradient itself. In contrast, optimizing VLoRP with ProjFactor fundamentally reduces memory consumption by solely

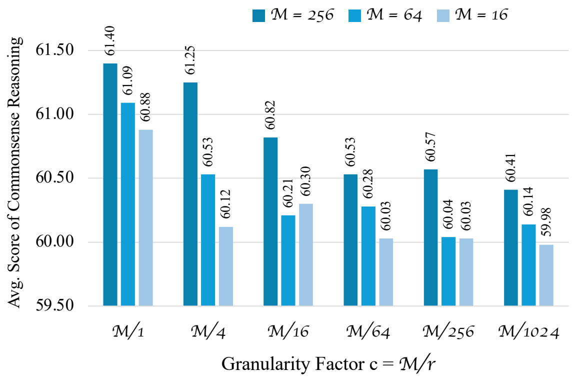


Figure 4: Performance Evaluation for Different Projection Granularities under Varying Memory Budgets on Commonsense. $\mathcal{M}$ denotes the memory budget. All subcolumns inside the same category $\mathcal{M}/r$ share the same rank r.

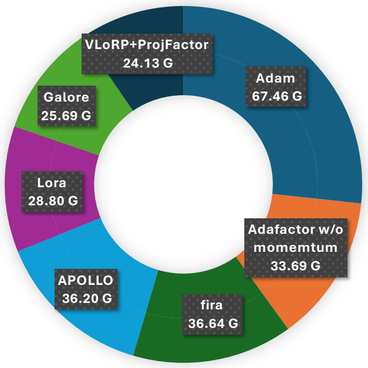


Figure 5: GPU Memory Usage Comparison on LLaMA2-7B Model with Batch Size 16 and Max Length 1024.

operating on the projected gradients. For a parameter matrix $W \in \mathbb{R}^{n\times m}$, the total memory required for VLoRP is $O(mn + 2n\mathcal{M} + n + m)$, where $\mathcal{M} = c \cdot r$ is the memory budget. On the other hand, LoRA requires significantly more storage, $O(mn + 4m\mathcal{M} + 4n\mathcal{M})$, due to its need for additional low-rank matrices and optimization states.

6. Conclusion

In this work, we introduce VLoRP, a memory-efficient fine-tuning method for LLMs, which implements low-rank gradient projections with varying granularities. By adjusting the projection granularity alongside rank, VLoRP offers a more nuanced control over the trade-off between memory efficiency and performance. Furthermore, we present ProjFactor, an adaptive optimizer that reduces memory consumption while maintaining competitive performance. Theoretical analysis and empirical results confirm the effectiveness of our approach, making it a promising solution for practical deployment in memory-constrained settings.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning by improving the memory efficiency and performance of LLM training. Specifically, our proposed method facilitates more efficient utilization of memory, which is particularly critical in the context of the current scarcity of computational resources. Furthermore, considering the significant energy consumption associated with training large models, particularly in terms of electricity and its environmental impact, a more efficient algorithm like ours represents a substantial step toward alleviating the energy costs of model training.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

Baydin, A. G., Pearlmutter, B. A., Syme, D., Wood, F., and Torr, P. Gradients without backpropagation. arXiv preprint arXiv:2202.08587, 2022.

Berahas, A. S., Cao, L., Choromanski, K., and Scheinberg, K. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. Foundations of Computational Mathematics, 22(2):507–560, 2022.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. PIQA: reasoning about physical commonsense in natural language. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pp. 7432–7439. AAAI Press, 2020.

Chen, A., Zhang, Y., Jia, J., Diffenderfer, J., Parasyris, K., Liu, J., Zhang, Y., Zhang, Z., Kailkhura, B., and Liu, S. Deepzero: Scaling up zeroth-order optimization for deep model training. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, 2024a.

Chen, L., Liu, B., Liang, K., and Liu, Q. Lion secretly solves constrained optimization: As lyapunov predicts. arXiv preprint arXiv:2310.05898, 2023.

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

Chen, X., Feng, K., Li, C., Lai, X., Yue, X., Yuan, Y., and Wang, G. Fira: Can we achieve full-rank training of llms under low-rank constraint? arXiv preprint arXiv:2410.01623, 2024b.

Clark, C., Lee, K., Chang, M., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pp. 2924–2936. Association for Computational Linguistics, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Dasgupta, S. and Gupta, A. An elementary proof of a theorem of johnson and lindenstrauss. Random Struct. Algorithms, 22(1):60–65, 2003.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. Advances in neural information processing systems, 25, 2012.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. In Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

Hao, Y., Cao, Y., and Mou, L. Flora: Low-rank adapters are secretly gradient compressors. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, 2024.

Hayou, S., Ghosh, N., and Yu, B. Lora+: Efficient low rank adaptation of large models. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, 2024.

He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. Towards a unified view of parameter-efficient transfer learning. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, 2022.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. Parameter-efficient transfer learning for NLP. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pp. 2790–2799, 2019.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, 2022.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E., Bing, L., Xu, X., Poria, S., and Lee, R. K. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In Bouamor, H., Pino, J., and Bali, K. (eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, pp. 5254–5276. Association for Computational Linguistics, 2023a.

Hu, Z., Yang, Z., Wang, Y., Karniadakis, G. E., and Kawaguchi, K. Bias-variance trade-off in physics-informed neural networks with randomized smoothing for high-dimensional pdes. arXiv preprint arXiv:2311.15283, 2023b.

Indyk, P. and Motwani, R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Vitter, J. S. (ed.), Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998, pp. 604–613. ACM, 1998.

Jaiswal, A., Yin, L., Zhang, Z., Liu, S., Zhao, J., Tian, Y., and Wang, Z. From galore to welore: How low-rank weights non-uniformly emerge from low-rank gradients. arXiv preprint arXiv:2407.11239, 2024.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. Vera: Vector-based random matrix adaptation. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, 2024.

LaSalle, J. Some extensions of liapunov's second method. IRE Transactions on circuit theory, 7(4):520–527, 1960.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021, pp. 3045–3059, 2021.

Li, B., Zhang, Y., Guo, D., Zhang, R., Li, F., Zhang, H., Zhang, K., Zhang, P., Li, Y., Liu, Z., et al. Llava-onevision: Easy visual task transfer. arXiv preprint arXiv:2408.03326, 2024.

Liang, K., Liu, B., Chen, L., and Liu, Q. Memory-efficient llm training with online subspace descent. arXiv preprint arXiv:2408.12857, 2024.

Liu, S., Wang, C., Yin, H., Molchanov, P., Wang, Y. F., Cheng, K., and Chen, M. Dora: Weight-decomposed low-rank adaptation. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, 2024.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019, 2019.

Maddison, C. J., Paulin, D., Teh, Y. W., O'Donoghue, B., and Doucet, A. Hamiltonian descent methods. arXiv preprint arXiv:1809.05042, 2018.

Mahabadi, R. K., Ruder, S., Dehghani, M., and Henderson, J. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021, pp. 565–576, 2021.

Malladi, S., Gao, T., Nichani, E., Damian, A., Lee, J. D., Chen, D., and Arora, S. Fine-tuning language models with just forward passes. Advances in Neural Information Processing Systems, 36:53038–53075, 2023.

Matousek, J. On variants of the johnson-lindenstrauss lemma. Random Struct. Algorithms, 33(2):142–156, 2008.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a suit of armor conduct electricity? A new dataset for open book question answering. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018, pp. 2381–2391. Association for Computational Linguistics, 2018.

Nevel'son, M. B. and Has' minskii, R. Z. Stochastic approximation and recursive estimation, volume 47. American Mathematical Soc., 1976.

Nguyen, S., Chen, L., Liu, B., and Liu, Q. H-fac: Memory-efficient optimization with factorized hamiltonian descent. arXiv preprint arXiv:2406.09958, 2024.

Pearlmutter, B. A. Fast exact multiplication by the hessian. Neural Comput., 6(1):147–160, 1994.

Razdaibiedina, A., Mao, Y., Khabsa, M., Lewis, M., Hou, R., Ba, J., and Almahairi, A. Residual prompt tuning: improving prompt tuning with residual reparameterization. In Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023, pp. 6740–6757, 2023.

Ren, M., Kornblith, S., Liao, R., and Hinton, G. Scaling forward gradient with local losses. arXiv preprint arXiv:2210.03310, 2022.

Robbins, H. and Monro, S. A stochastic approximation method. The annals of mathematical statistics, pp. 400–407, 1951.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. In The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pp. 8732–8740. AAAI Press, 2020.

Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions. arXiv preprint arXiv:1904.09728, 2019.

Shazeer, N. and Stern, M. Adafactor: Adaptive learning rates with sublinear memory cost. In Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, volume 80 of Proceedings of Machine Learning Research, pp. 4603–4611, 2018.

Shen, Q., Wang, Y., Yang, Z., Li, X., Wang, H., Zhang, Y., Scarlett, J., Zhu, Z., and Kawaguchi, K. Memory-efficient gradient unrolling for large-scale bi-level optimization. arXiv preprint arXiv:2406.14095, 2024.

Silver, D., Goyal, A., Danihelka, I., Hessel, M., and van Hasselt, H. Learning by directional gradient descent. In International Conference on Learning Representations, 2021.

Smith, S. L., Kindermans, P., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

Spall, J. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control, 37(3):332–341, 1992.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288, 2023.

Vyas, N., Morwani, D., and Kakade, S. M. Adamem: Memory efficient momentum for adafactor. In 2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024), 2024.

Wang, C., Chen, X., Smola, A. J., and Xing, E. P. Variance reduction for stochastic gradient optimization. In Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pp. 181–189, 2013.

Wang, S., Yu, L., and Li, J. Lora-ga: Low-rank adaptation with gradient approximation. arXiv preprint arXiv:2407.05000, 2024.

Wang, Y., Lin, Y., Zeng, X., and Zhang, G. Multilora: Democratizing lora for better multi-task learning. arXiv preprint arXiv:2311.11501, 2023.

Wengert, R. E. A simple automatic derivative evaluation program. Commun. ACM, 7(8):463–464, 1964.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. Neural Comput., 1(2):270–280, 1989.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? In Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pp. 4791–4800. Association for Computational Linguistics, 2019.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In The Eleventh International Conference on Learning Representations, 2023.

Zhao, J., Zhang, Z., Chen, B., Wang, Z., Anandkumar, A., and Tian, Y. Galore: Memory-efficient LLM training by gradient low-rank projection. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, 2024.

Zhu, H., Zhang, Z., Cong, W., Liu, X., Park, S., Chandra, V., Long, B., Pan, D. Z., Wang, Z., and Lee, J. Apollo: Sgd-like memory, adamw-level performance. arXiv preprint arXiv:2412.05270, 2024.

Table 3: A detailed table for notations used in this paper.

Symbol	Definition	Description
$(\cdot)^s$	Subspace tag	Distinguish variables in the projected low-dimensional space from those in the original space.
$(\cdot)^o$	Original Space tag	Distinguish variables in the original space from those in the projected low-dimensional space;
$(\cdot)_t$	Update step tag	Denotes the specific step of current variables, for example, the gradient G at the t-th update step can be denoted as G_t;
$(\cdot)^{\odot 2}$	Element-wise Squaring	The element-wise squaring of a matrix;
$\|\| \cdot \|\|$	Frobenius norm	Taking the square root of the summation of each squared element;
$\langle \cdot, \cdot \rangle$	Frobenius inner product	Inner product induced by Frobenius norm;
W	The Parameters Matrix	The shape is assumed as $n \times m$ with $n \geq m$;
$\mathcal{L}$	The loss function	The loss function of the training procedure;
$P, \tilde{P}$	Projection Matrix	Randomly sampled from a normal distribution $\mathcal{N}(0, \frac{1}{r})$ unless otherwise stated. The shape of P is $m \times r$ with $r \ll \min\{m, n\}$, while the shape of $\tilde{P}$ is $(m/c) \times r$ or $(m/c) \times (\mathcal{M}/c)$;
c	granularity factor	The parameter c is a hyperparameter of VLoRP that controls the granularity of projections. For instance, setting $c = 2$ reduces the granularity of projections by half for the entire model. For vanilla low-rank-based methods like LoRA or Galore, their c is equal to 1;
r	rank	The parameter r is a hyperparameter of VLoRP and other low-rank based memory-efficient methods, such as LoRA and Galore;
$\mathcal{M}$	Memory Budget	We introduce the memory budget, denoted as $\mathcal{M}$, for VLoRP to facilitate the comparison between different configurations of (c, r). The memory budget $\mathcal{M}$ is defined as the product of c and r, as both parameters jointly influence the memory requirements during LLM training. For other low-rank-based methods, where $c = 1$, the rank is set to $r = \mathcal{M}$;
G, G^s, G^o	Gradient	Gradient computed for a single forward-backward step. The shape is equal to W; G^s represents the projected gradient, *i.e.* $G^s = GP$, and the shape of G^s is $n \times r$; G^o represents the projected-back gradient, *i.e.* $G^o = GPP^\top$, where r is the rank, and the shape G^o is $n \times m$;
$\tilde{G}, \tilde{G}^s, \tilde{G}^o$	Reshaped Gradient	The reshaped version of gradients. The shape of $\tilde{G}$ is equal to $nc \times (m/c)$; The shape of G^s is $nc \times r$; The shape $\tilde{G}^o$ is $nc \times (m/c)$;
$\boldsymbol{G}, \boldsymbol{G}^s, \boldsymbol{G}^o$	Accumulated Gradient	Accumulation of gradients over multiple forward-backward steps, that is $\boldsymbol{G} = \sum_{i=1}^{k} G_i$ where k is number of accumulation steps; $\boldsymbol{G}^s$/ $\boldsymbol{G}^o$ represents the projected/projected-back gradient, *i.e.* $\boldsymbol{G}^s = \boldsymbol{G}P$; $\boldsymbol{G}^o = \boldsymbol{G}PP^\top$;
m, m^s, m^o	First moment of Adam	$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\boldsymbol{G}_t$, where β_1 represents the coefficient. m^s represents the states stored in the subspace, *i.e.* $m_t^s = \beta_1 m_{t-1}^s + (1 - \beta_1)\boldsymbol{G}_t^s$; m^o represents the states stored in the original space, *i.e.* $m_t^o = \beta_1 m_{t-1}^o + (1 - \beta_1)\boldsymbol{G}_t^o$;
v, v^s, v^o	Second moment of Adam	$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\boldsymbol{G}_t)^{\odot 2}$; $v_t^s = \beta_1 v_{t-1}^s + (1 - \beta_1)(\boldsymbol{G}_t^s)^{\odot 2}$; $v_t^o = \beta_1 v_{t-1}^o + (1 - \beta_1)(\boldsymbol{G}_t^o)^{\odot 2}$.

A. Notations

In Table 3, we provide the notations used in the main body and appendix of this paper. In case of any discrepancies between the definitions of the symbols in the table and those in the text, the definitions in the text should be followed.

Next, in Appendix B, we discuss our optimization scheme, specifically the algorithmic details of VLoRP with ProjFactor. Appendix C presents the proofs for all theoretical results and propositions introduced in the main text. Appendix D further provides several analytical experiments and ablation studies with specific implementation details. Finally, in Appendix E, we conduct an in-depth discussion of related works.

B. Algorithm of ProjFactor for VLoRP

In Algorithm 1, we present the final algorithm employed in our study—optimizing VLoRP with ProjFactor. Formally, given a learning rate η, a parameter matrix W, rank r, granularity factor c, and resampling gap τ, we first initialize $\tilde{m}^s$, $\tilde{v}_r^o$, and $\tilde{v}_c^o$, which serve as optimization states and need to be stored throughout training. Next, at the beginning of each update iteration, a zero matrix $\tilde{\boldsymbol{G}}^s \in \mathbf{0}^{nc\times r}$ is created to store the projected accumulated gradient. Subsequently, K substeps of forward-backward propagation are performed with each gradient $\nabla_W \mathcal{L}(\mathcal{B}_i)$ ($\mathcal{B}_i$ denotes the mini-batch data of the accumulation step i) projected, reshaped, and accumulated in $\tilde{\boldsymbol{G}}^s$. After the gradient projection and accumulation, in line 13, we update the state $\tilde{m}^s$ of the first moment, while in lines 14–16, we first project $\tilde{\boldsymbol{G}}_t^s$ back to the original space via $\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top$, and then perform the second moment update through factorization (Shazeer & Stern, 2018). It is important to note that $\tilde{m}^s$ is first projected back to the original space using $\tilde{m}_t^s \tilde{P}^\top$ prior to calculating Δ_t^o, in alignment with the **Original Space Scheme**. The following relation justifies this:

$$\tilde{m}_t^o = \beta_1\, \tilde{m}_{t-1}^o + (1-\beta_1)\tilde{\boldsymbol{G}}_t^o = \sum_{\tau=1}^{t} \beta_1^{t-\tau}(1-\beta_1)\tilde{\boldsymbol{G}}_\tau^o = \sum_{\tau=1}^{t} \beta_1^{t-\tau}(1-\beta_1)\left(\tilde{\boldsymbol{G}}_\tau^s \tilde{P}^\top\right) = \tilde{m}_t^s \tilde{P}^\top.$$

Additionally, before updating W in line 17, we multiply a bias correction term $\frac{1-\beta_2^t}{1-\beta_1^t}$, as in Adam (Kingma & Ba, 2014) and Adafactor (Shazeer & Stern, 2018). Besides, with the same ζ, the result of generation $\tilde{P}$ is equal.

Algorithm 1 ProjFactor for VLoRP

```
1:  **Input:** learning rate $\eta$, parameter $W \in \mathbb{R}^{n\times m}$, rank $r$, granularity factor $c$, resampling gap $\tau$;
2:  **Initialize:** $\tilde{m}^s \leftarrow \mathbf{0}^{nc\times r}$, $\tilde{v}_r^o \leftarrow \mathbf{0}^{nc\times 1}$, $\tilde{v}_c^o \leftarrow \mathbf{0}^{1\times \frac{m}{c}}$;
3:  **while** not converged **do**
4:      **if** $t \bmod \tau == 0$ **then**
5:          Resampling random seed $\zeta$;
6:      **end if**
7:      $\tilde{\boldsymbol{G}}_t^s \leftarrow \mathbf{0}^{nc\times r}$;
8:      **for** $i = 1, 2, \ldots, K$ **do**
9:          Sample a mini-batch $\mathcal{B}_i$, calculate $\mathcal{L}(B_i)$ and then generate $\tilde{P} \in \mathbb{R}^{\frac{m}{c}\times r}$ with $\tilde{p}_{ij} \sim \mathcal{N}_\zeta(0, 1/r)$
10:         $\tilde{\boldsymbol{G}}_t^s \leftarrow \tilde{\boldsymbol{G}}_t^s + \mathrm{Reshape}(\nabla_W \mathcal{L}(\mathcal{B}_i)/K, [nc, \frac{m}{c}])\tilde{P}$;
11:     **end for**
12:     Generate $\tilde{P} \in \mathbb{R}^{\frac{m}{c}\times r}$ with $\tilde{p}_{ij} \sim \mathcal{N}_\zeta(0, 1/r)$;
13:     $\tilde{m}^s \leftarrow \beta_1 \tilde{m}^s + (1-\beta_1)\tilde{\boldsymbol{G}}_t^s$;
14:     $\tilde{v}_r^o \leftarrow \beta_2 \tilde{v}_r^o + (1-\beta_2)\left(\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top\right)^{\odot 2} \mathbf{1}_m$;
15:     $\tilde{v}_c^o \leftarrow \beta_2 \tilde{v}_c^o + (1-\beta_2)\mathbf{1}_n^\top \left(\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top\right)^{\odot 2}$;
16:     $\Delta_t^o = \mathrm{Reshape}\left(\tilde{m}_t^s \tilde{P}^\top / \left(\sqrt{\frac{\tilde{v}_r^o \tilde{v}_c^o}{\mathbf{1}_n^\top \tilde{v}_r^o}} + \epsilon\right), [n, m]\right)$;
17:     $W \leftarrow W - \eta\, \frac{1-\beta_2^t}{1-\beta_1^t} \Delta_t^o$ ;
18:     $t \leftarrow t + 1$;
19: **end while**
20: **Output:** Optimized $W$.
```

C. Proof of Theorems

In this section, we provide the proof for Proposition C.2, Theorem 3.3, and Theorem 4.1. Throughout the proofs, we adopt the Frobenius norm and inner product as the primary metrics for matrices and vectors.

C.1. Proof of Proposition 3.2

We first concretely compute the variance of the forward gradient estimator with Gaussian samples for vector-input functions. A similar case is studied in Shen et al. (2024), where the samples are i.i.d. Rademacher distribution.

Lemma C.1. *Let $h : \mathbb{R}^N \to \mathbb{R}$ be a differentiable function, and fix any point $\phi \in \mathbb{R}^N$. Let $g := \nabla_\phi h(\phi) \in \mathbb{R}^{1\times N}$ be the gradient viewed as a row vector. Suppose we draw b i.i.d. samples $v_1, \dots, v_b \sim \mathcal{N}(0, I_N)$, with each v_i being an $N \times 1$ column vector. Define the forward gradient estimator of size b by*

$$\hat{g} = \frac{1}{b}\sum_{i=1}^{b} g v_i v_i^\top,$$

then its mean squared error is

$$\mathbb{E}\Big[\|\hat{g} - g\|^2\Big] = \frac{N+1}{b}\|g\|^2.$$

Proof. **Unbiasedness:** Consider a single random sample $v \sim \mathcal{N}(0, I_N)$. Since $\mathbb{E}[vv^\top] = I_N$, we have

$$\mathbb{E}\big[gvv^\top\big] = g\mathbb{E}[vv^\top] = g.$$

By linearity of expectation, averaging b such i.i.d. samples preserves unbiasedness:

$$\mathbb{E}\Bigg[\frac{1}{b}\sum_{i=1}^{b}\big(g v_i v_i^\top\big)\Bigg] = g.$$

Variance: First, consider one-sample estimator $gvv^\top$. Let $\alpha := gv \in \mathbb{R}$. Then $g(vv^\top) = (gv)v^\top = \alpha v^\top$. By moment identities, we have

$$\mathbb{E}\big[\|gvv^\top - g\|^2\big] = \mathbb{E}\big[\|\alpha v^\top - g\|^2\big] = \mathbb{E}\big[\|\alpha v^\top\|^2 - 2\langle \alpha v^\top, g\rangle + \|g\|^2\big] = (N+1)\|g\|^2.$$

With b i.i.d. samples,

$$\mathbb{E}\big[\|\hat{g} - g\|^2\big] = \frac{1}{b^2}\sum_{i=1}^{b}\mathbb{E}\big[\|g v_i v_i^\top - g\|^2\big] = \frac{1}{b^2}\big(b\cdot(N+1)\,\|g\|^2\big) = \frac{N+1}{b}\|g\|^2.$$

□

We consider a loss function $\mathcal{L} : \mathbb{R}^{n\times m} \to \mathbb{R}$ defined over matrix parameters $W \in \mathbb{R}^{n\times m}$. Fix a memory budget $\mathcal{M}$, a granularity factor c and rank r with $\mathcal{M} = cr$. Let $G = \nabla_W \mathcal{L}(W)$, and let $\tilde{G}$, $\tilde{G}^s$, $\tilde{G}^o$, and G^o follow the definitions in (5).

Proposition C.2. *The gradient estimator G^o satisfies the following properties:*

$$\mathbb{E}[G^o] = G, \tag{8}$$

$$\mathbb{E}\|G^o - G\|^2 = \frac{m+c}{\mathcal{M}}\|G\|^2. \tag{9}$$

Proof. Recall that G is reshaped into $\tilde{G} \in \mathbb{R}^{(nc)\times(m/c)}$, and then randomly approximated by $\tilde{G}^o = \tilde{G}\tilde{P}\tilde{P}^\top$, where $\tilde{P} \in \mathbb{R}^{\frac{m}{c}\times r}$ have i.i.d. Gaussian columns $v_i \in \mathbb{R}^{m/c}, i = 1, \dots, r$ with $r = \mathcal{M}/c$, such that

$$\tilde{P}\tilde{P}^\top = \sum_{i=1}^{r} v_i v_i^\top.$$

And finally, $\tilde{G}^o$ is reshaped back to size $n \times m$ to obtain G^o.

Unbiasedness: Row-by-row application of Lemma C.1 shows each row of $\tilde{G}^o$ is an unbiased estimator of the corresponding row of $\tilde{G}$. Hence $\mathbb{E}[\tilde{G}^o] = \tilde{G}$. Consequently, $\mathbb{E}[G^o] = G$, for reshaping does not affect the bias.

Variance: We first write

$$\tilde{G}^o = \frac{1}{r}\sum_{i=1}^{r} \tilde{G} v_i v_i^\top = \begin{pmatrix} \frac{1}{r}\sum_{i=1}^{r}(\tilde{G}_{1,:}v_i)v_i^\top \\ \vdots \\ \frac{1}{r}\sum_{i=1}^{r}(\tilde{G}_{nc,:}v_i)v_i^\top \end{pmatrix}.$$

By Lemma C.1, each row (dimension $d = \frac{m}{c}$) with $b = \frac{r}{c}$ samples has variance

$$\mathbb{E}\left[\left\|\frac{1}{r}\sum_{i=1}^{r}(v_i^\top \tilde{G}_{i,:}^\top)v_i^\top\right\|^2\right] = \frac{\frac{m}{c}+1}{r}\|\tilde{G}_{i,:}\|^2 = \frac{m+c}{\mathcal{M}}\|\tilde{G}_{i,:}\|^2,$$

where the last equality uses $\mathcal{M} = cr$. Subsequently, summing over all rows in $\tilde{G}$ yields

$$\mathbb{E}\big[\|\tilde{G}^o - \tilde{G}\|^2\big] = \frac{m+c}{\mathcal{M}}\|\tilde{G}\|^2.$$

Because reshaping does not change the Frobenius norm,

$$\mathbb{E}\big[\|G^o - G\|^2\big] = \frac{m+c}{\mathcal{M}}\,\|G\|^2.$$

Thus G^o is unbiased with the stated mean-squared error. □

C.2. Proof of Theorem 3.3

Theorem C.3. *Let $\mathcal{L}$ be an L-smooth function with respect to the matrix-shaped parameter W, i.e.,*

$$\mathcal{L}(W') \le \mathcal{L}(W) + \big\langle G, W' - W\big\rangle + \frac{L}{2}\|W' - W\|^2$$

for any $W, W' \in \mathbb{R}^{n\times m}$. Assume the parameter updates are given by:

$$W_{t+1} = W_t - \eta G_t^o,$$

where the step size is defined as $\eta = \frac{\mathcal{M}}{(m+c+\mathcal{M})L} \triangleq C$. Then, for any $T \ge 1$:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[\|G_t\|^2\big] \le \frac{2C}{T}\big(\mathcal{L}(W_0) - \mathcal{L}(W^*)\big),$$

where W^* is a global minimizer of $\mathcal{L}$.

Proof. By L-smoothness and the update rule $W_{t+1} = W_t - \eta G_t^o$, we have

$$\begin{aligned}\mathcal{L}(W_{t+1}) &\le \mathcal{L}(W_t) + \Big\langle G_t, W_{t+1} - W_t\Big\rangle + \frac{L}{2}\big\|W_{t+1} - W_t\big\|^2 \\ &= \mathcal{L}(W_t) - \eta\Big\langle G_t, G_t^o\Big\rangle + \frac{L\eta^2}{2}\big\|G_t^o\big\|^2 \\ &= \mathcal{L}(W_t) - \eta\Big\langle G_t, G_t^o\Big\rangle + \frac{L\eta^2}{2}\big\|G_t^o - G_t\big\|^2 + \frac{L\eta^2}{2}\big\|G_t\big\|^2 + L\eta^2\Big\langle G_t^o - G_t, G_t\Big\rangle.\end{aligned}$$

Taking the expectation over the randomness in G_t^o conditioning on W_t, and then using the unbiasedness of G_t^o, we get

$$\begin{aligned}\mathbb{E}\big[\mathcal{L}(W_{t+1})|W_t\big] &\le \mathbb{E}\big[\mathcal{L}(W_t)|W_t\big] - \left(\eta - \frac{L\eta^2}{2}\right)\mathbb{E}\big[\|G_t\|^2|W_t\big] + \frac{L\eta^2}{2}\mathbb{E}\big[\|G_t^o - G_t\|^2|W_t\big] \\ &= \mathcal{L}(W_t) - \left(\eta - \frac{L\eta^2}{2}\right)\|G_t\|^2 + \frac{L\eta^2}{2}\mathbb{E}\big[\|G_t^o - G_t\|^2|W_t\big].\end{aligned}$$

By Proposition C.2, $\mathbb{E}\big[\|G_t^o - G_t\|^2|W_t\big] = \frac{m+c}{\mathcal{M}}\|G_t\|^2$. Hence

$$\mathbb{E}\big[\mathcal{L}(W_{t+1})\big|W_t\big] \le \mathcal{L}(W_t) - \left(\eta - \frac{(m+c+\mathcal{M})L\eta^2}{2\mathcal{M}}\right)\|G_t\|^2.$$

Taking expectation over W_t, we further write

$$\mathbb{E}\big[\mathcal{L}(W_{t+1})\big] \le \mathbb{E}\big[\mathcal{L}(W_t)\big] - \left(\eta - \frac{(m+c+\mathcal{M})L\eta^2}{2\mathcal{M}}\right)\mathbb{E}\big[\|G_t\|^2\big]. \tag{10}$$

Summing (10) from $t = 0$ to $t = T-1$ and telescoping on the left-hand side:

$$\mathbb{E}\big[\mathcal{L}(W_T)\big] \le \mathbb{E}\big[\mathcal{L}(W_0)\big] - \sum_{t=0}^{T-1}\left(\eta - \frac{(m+c+\mathcal{M})L\eta^2}{2\mathcal{M}}\right)\mathbb{E}\big[\|G_t\|^2\big].$$

Rearrange to isolate the sum of gradient norms:

$$\sum_{t=0}^{T-1}\mathbb{E}\big[\|G_t\|^2\big] \le \frac{\mathcal{L}(W_0) - \mathbb{E}\big[\mathcal{L}(W_T)\big]}{\eta - \frac{(m+c+\mathcal{M})L\eta^2}{2\mathcal{M}}} \le \frac{\mathcal{L}(W_0) - \mathcal{L}(W^*)}{\eta - \frac{(m+c+\mathcal{M})L\eta^2}{2\mathcal{M}}},$$

where W^* is a minimizer for $\mathcal{L}$. Choosing $\eta = \frac{\mathcal{M}}{(m+c+\mathcal{M})L} := C$ yields

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\big[\|G_t\|^2\big] \le \frac{2(m+c+\mathcal{M})L}{\mathcal{M}T}\big(\mathcal{L}(W_0) - \mathcal{L}(W^*)\big) = \frac{2C}{T}\big(\mathcal{L}(W_0) - \mathcal{L}(W^*)\big).$$

Thus, on average, the norm of the true gradient converges to zero at a rate $O(1/T)$. □

C.3. Proof of Theorem 4.1

To analyze the convergence properties of ProjFactor, we leverage the Hamiltonian descent framework (Maddison et al., 2018; Chen et al., 2023; Liang et al., 2024; Nguyen et al., 2024), a powerful tool for understanding the behavior of optimizers in continuous time. This framework allows us to model ProjFactor's update rule as an ordinary differential equation (ODE), providing insights into its long-term stability and convergence.

The infinitesimal updates of Projfactor is defined as follows:

$$\begin{aligned}&\frac{d}{dt}\tilde{m}_t^s = a(\tilde{G}_t^s - \tilde{m}_t^s);\ \hat{v}_t^o = \frac{\tilde{v}_{rt}^o\tilde{v}_{ct}^o}{\mathbf{1}_n^T\tilde{v}_{rt}^o};\\ &\frac{d}{dt}\tilde{v}_{rt}^o = b((\tilde{G}_t^o)^{\odot 2}\mathbf{1}_m - \tilde{v}_{rt}^o);\\ &\frac{d}{dt}\tilde{v}_{ct}^o = b(\mathbf{1}_n^T(\tilde{G}_t^o)^{\odot 2} - \tilde{v}_{ct}^o);\\ &\frac{d}{dt}W_t = \text{Reshape}\left(-\tilde{m}_t^s\tilde{P}^\top\Big/\sqrt{\hat{v}_t^o},\ [n,m]\right)\end{aligned} \tag{11}$$

The corresponding Lyapunov function (Hamiltonian) is defined as

$$\mathcal{H}(W, \tilde{m}^s, \tilde{v}_r^o, \tilde{v}_c^o) = \mathcal{L}(W) + \frac{1}{2a}\left\langle \tilde{m}^s, \frac{\tilde{m}^s}{\sqrt{\hat{v}^o}}\right\rangle.$$

Subsequently, we make the following mild assumptions, consistent with prior works in this area, to establish the mathematical foundation for our analysis. (Maddison et al., 2018; Chen et al., 2023; Liang et al., 2024; Nguyen et al., 2024).

Assumption C.4. Assume the functions in system (11) are continuously differentiable, and

(1) $\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) = 0$ implies $\tilde{G}_t^s = 0$.

(2) For any $t > 0$, if $G_t \neq 0$, then $\tilde{G}_t^s \neq 0$ and $\tilde{G}_t^o \neq 0$.

(3) For any $t > 0$, $\frac{\|\tilde{G}_t^o\|^2}{\|\tilde{v}_{rt}^o\|} \leq R$.

Here, Assumption C.4 (1) claims that the system's Lyapunov function reaches a stationary point only when $\tilde{G}_t^s = 0$. This condition aligns with the behavior of widely used optimizers like SGD with momentum and Adam (Liang et al., 2024). Assumption C.4 (2) prevents the projection operator $\tilde{P}$ from annihilating nonzero gradients. Whenever $\tilde{G}^s = 0$ or $\tilde{G}^o = 0$, we have $G = 0$, which maintains consistency between projected and original spaces. Assumption C.4 (3) imposes a reasonable bound on the ratio of the squared gradient norm to the second moment. This bound can be intuitively derived by expanding the second-moment update rule in ProjFactor (Algorithm 1):

$$\tilde{v}_r^o \leftarrow \beta_2 \tilde{v}_r^o + (1-\beta_2)\left(\tilde{G}_t^s \tilde{P}^\top\right)^{\odot 2} \mathbf{1}_m.$$

By $\tilde{G}_t^o = \tilde{G}_t^s \tilde{P}^\top$ and the summation of geometric series, we further have

$$\tilde{v}_{rt}^o = \sum_{\tau=1}^{t} \beta_2^{t-\tau}(1-\beta_2)(\tilde{G}_\tau^o)^{\odot 2}\mathbf{1}_m.$$

Hence, if $\tilde{G}_t^o \to 0$ as $t \to \infty$, $\frac{\|\tilde{G}_t^o\|^2}{\|\tilde{v}_{rt}^o\|}$ will be bounded.

Now we present the following convergence analysis, which interprets Projfactor from the perspective of the Hamiltonian descent method.

Theorem C.5. *Suppose the functions in system (11) are continuously differentiable. Under Assumption C.4, we have*

1. *For $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)$ satisfying (11),*
$$\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) \leq 0.$$
2. *Any bounded solution $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)_t$ of (11) converges to a stationary point of $\mathcal{L}(W)$ as $t \to \infty$.*

Proof. First, we prove that $\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) \leq 0$. For simplicity, we denote that

$$R_t := \frac{1}{2a}\left\langle \tilde{m}^s, \frac{\tilde{m}^s}{\sqrt{\hat{v}^o}} \right\rangle.$$

By chain rule of derivatives, we have

$$\begin{aligned}\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) &= \frac{d}{dt}\mathcal{L}(W_t) + \frac{d}{dt}R_t \\ &= \left\langle \frac{d\mathcal{L}(W_t)}{dW_t}, \frac{dW_t}{dt} \right\rangle + \left\langle \frac{dR_t}{d\tilde{m}_t^s}, \frac{d\tilde{m}_t^s}{dt} \right\rangle + \left\langle \frac{dR_t}{d\tilde{v}_{rt}^o}, \frac{d\tilde{v}_{rt}^o}{dt} \right\rangle + \left\langle \frac{dR_t}{d\tilde{v}_{ct}^o}, \frac{d\tilde{v}_{ct}^o}{dt} \right\rangle.\end{aligned} \tag{12}$$

We compute the terms in (12) respectively. Firstly, by the dynamics in (11),

$$\begin{aligned}\left\langle \frac{d\mathcal{L}(W_t)}{dW_t}, \frac{dW_t}{dt} \right\rangle + \left\langle \frac{dR_t}{d\tilde{m}_t^s}, \frac{d\tilde{m}_t^s}{dt} \right\rangle &= \left\langle \nabla\mathcal{L}(W_t), -\operatorname{Reshape}\left(\frac{\tilde{m}_t^s \tilde{P}^\top}{\sqrt{\hat{v}_t^o}}, [n,m]\right) \right\rangle + \left\langle \frac{1}{2a}\frac{2\tilde{m}_t^s\sqrt{\mathbf{1}_n^\top \tilde{v}_{rt}^o}}{\sqrt{\tilde{v}_{rt}^o \tilde{v}_{ct}^o}}, a(\tilde{G}_t^s - \tilde{m}_t^s) \right\rangle \\ &= -\operatorname{tr}\left(\frac{\tilde{m}_t^s \tilde{P}^\top \operatorname{Reshape}\left(\nabla\mathcal{L}(W_t)^\top, [nc, \frac{m}{c}]\right)}{\sqrt{\hat{v}_t^o}}\right) + \operatorname{tr}\left(\frac{\tilde{m}_t^s (\tilde{G}_t^s)^\top}{\sqrt{\hat{v}_t^o}}\right) - \left\langle \frac{\tilde{m}_t^s}{\sqrt{\hat{v}_t^o}}, \tilde{m}_t^s \right\rangle \\ &= -\left\langle \frac{\tilde{m}_t^s}{\sqrt{\hat{v}_t^o}}, \tilde{m}_t^s \right\rangle,\end{aligned} \tag{13}$$

where the third equality is based on the fact that reshaping both matrices of Frobenius inner product does not change the result.

Secondly, notice that $\left\langle \frac{dR_t}{d\tilde{v}^o_{rt}}, \frac{d\tilde{v}^o_{rt}}{dt} \right\rangle$ is the inner product of two $nc \times 1$ vectors, we assume $(\tilde{v}^o_{rt})_k$ to be the k-th element of $\tilde{v}^o_{rt}$, and $(\tilde{G}^o_t)_{k,:}$ to be the k-th row of $\tilde{G}^o_t$. Recalling the ODE dynamics of $\tilde{v}^o_{rt}$ in (11), we further have

$$\begin{aligned}
\left\langle \frac{dR_t}{d\tilde{v}^o_{rt}}, \frac{d\tilde{v}^o_{rt}}{dt} \right\rangle &= \left(\frac{dR_t}{d\tilde{v}^o_{rt}}\right)^\top \frac{d\tilde{v}^o_{rt}}{dt} = \sum_{k=1}^{n} \frac{dR_t}{d(\tilde{v}^o_{rt})_k} \frac{d(\tilde{v}^o_{rt})_k}{dt} \\
&= \sum_{k=1}^{n} \left(\left(\frac{1}{2a} \sum_{i=1}^{n}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{ij}}{2\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})_j}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}} - \frac{1}{2a}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{kj}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}}{2\sqrt{(\tilde{v}^o_{rt})^3_k(\tilde{v}^o_{ct})_j}} \right) \cdot b\left((\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m - (\tilde{v}^o_{rt})_k \right) \right) \\
&= \frac{b}{4a}\sum_{k=1}^{n} \left(\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t \mathbf{1}_n^\top \tilde{v}^o_{rt}}} \right\rangle \left((\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m - (\tilde{v}^o_{rt})_k \right) \right) - \frac{b}{4a}\sum_{k=1}^{n}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{kj}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}(\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m}{\sqrt{(\tilde{v}^o_{rt})^3_k(\tilde{v}^o_{ct})_j}} \\
&\quad + \frac{b}{4a}\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}} \right\rangle \qquad (14) \\
&= \frac{b}{4a}\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t \mathbf{1}_n^\top \tilde{v}^o_{rt}}} \right\rangle \left(\|\tilde{G}^o_t\|^2 - \mathbf{1}_n^\top \tilde{v}^o_{rt} \right) - \frac{b}{4a}\sum_{k=1}^{n}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{kj}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}(\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m}{\sqrt{(\tilde{v}^o_{rt})_k(\tilde{v}^o_{ct})_j}\mathbf{1}_n^\top \tilde{v}^o_{rt}} + \frac{b}{4a}\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}} \right\rangle \\
&= \frac{b}{4a}\frac{1}{\mathbf{1}_n^\top \tilde{v}^o_{rt}} \left(\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}} \right\rangle \|\tilde{G}^o_t\|^2 - \sum_{k=1}^{n}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{kj}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}(\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m}{\sqrt{(\tilde{v}^o_{rt})_k(\tilde{v}^o_{ct})_j}} \right) \\
&\le \frac{bR}{4a}\left\langle \tilde{m}^s_t, \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}} \right\rangle,
\end{aligned}$$

where the inequality comes from Assumption C.4 (3) and the fact that $-\sum_{k=1}^{n}\sum_{j=1}^{m} \frac{(\tilde{m}^s_t)^2_{kj}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}(\tilde{G}^o_t)^{\odot 2}_{k,:}\mathbf{1}_m}{\sqrt{(\tilde{v}^o_{rt})_k(\tilde{v}^o_{ct})_j}} \le 0$.

Thirdly, since $\left\langle \frac{dR_t}{d\tilde{v}^o_{ct}}, \frac{d\tilde{v}^o_{ct}}{dt} \right\rangle$ is the inner product of two $1 \times m$ vectors, we assume $(\tilde{v}^o_{ct})_l$ to be the l-th element of $\tilde{v}^o_{ct}$, and $(\tilde{G}^o_t)_{:,l}$ to be the l-th column of $\tilde{G}^o_t$. With the ODE dynamics of $\tilde{v}^o_{ct}$ in (11), we deduce

$$\begin{aligned}
\left\langle \frac{dR_t}{d\tilde{v}^o_{ct}}, \frac{d\tilde{v}^o_{ct}}{dt} \right\rangle &= \frac{dR_t}{d\tilde{v}^o_{ct}} \left(\frac{d\tilde{v}^o_{ct}}{dt}\right)^\top = \sum_{l=1}^{m} \frac{dR_t}{d(\tilde{v}^o_{ct})_l}\left(\frac{d(\tilde{v}^o_{ct})_l}{dt}\right) \\
&= \sum_{l=1}^{m}\left(\sum_{i=1}^{n} \left(\frac{1}{2a} \frac{-(\tilde{m}^s_t)^2_{il}\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}}}{2\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})^3_l}} \right) \cdot b\left(\mathbf{1}_n^\top (\tilde{G}^o_t)^{\odot 2}_{:,l} - (\tilde{v}^o_{ct})_l \right) \right) \\
&= \frac{b}{4a}\sum_{l=1}^{m}\sum_{i=1}^{n} \left(-\frac{(\tilde{m}^s_t)^2_{il}\mathbf{1}_n^\top (\tilde{G}^o_t)^{\odot 2}_{:,l}}{\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})^3_l}} + \frac{(\tilde{m}^s_t)^2_{il}}{\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})_l}} \right)\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}} \qquad (15) \\
&\le \frac{b}{4a}\sum_{l=1}^{m}\sum_{i=1}^{n} \left(\frac{(\tilde{m}^s_t)^2_{il}}{\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})_l}} \right)\sqrt{\mathbf{1}_n^\top \tilde{v}^o_{rt}} \\
&= \frac{b}{4a}\left\langle \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}}, \tilde{m}^s_t \right\rangle,
\end{aligned}$$

where the inequality is due to $\frac{b}{4a}\sum_{l=1}^{m}\sum_{i=1}^{n} -\frac{(\tilde{m}^s_t)^2_{il}\mathbf{1}_n^\top (\tilde{G}^o_t)^{\odot 2}_{:,l}}{\sqrt{(\tilde{v}^o_{rt})_i(\tilde{v}^o_{ct})^3_l}} \le 0$.

Combining (12), (13), (14), (15), and setting $a \ge (R+1)b/4a$, we have

$$\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}^s_t, \tilde{v}^o_{rt}, \tilde{v}^o_{ct}) \le -\left(1 - \frac{(R+1)b}{4a}\right)\left\langle \frac{\tilde{m}^s_t}{\sqrt{\hat{v}^o_t}}, \tilde{m}^s_t \right\rangle \le 0, \qquad (16)$$

which demonstrates that the Lyapunov function $\mathcal{H}(W_t, \tilde{m}^s_t, \tilde{v}^o_{rt}, \tilde{v}^o_{ct})$ is monotonically decreasing along the ODE trajectory.

Furthermore, define

$$\mathcal{I} = \left\{ \text{the union of complete trajectories satisfying } \frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) = 0, \, \forall t \right\}.$$

Subsequently, by LaSalle's invariance principle (LaSalle, 1960), as $t \to +\infty$, the accumulation points of any trajectory $\{(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)\}_t$ lies in $\mathcal{I}$. By Assumption C.4 (1), the points in the limit set $\mathcal{I}$ should satisfy that for any t, $\tilde{G}_t^s = 0$. Hence, by Assumption C.4 (2), we further have $G_t = \nabla\mathcal{L}(W_t) = 0$ for any t. This indicates that any trajectory will converge to the local optimum. □

Theorem C.5 demonstrates the Lyapunov function's monotonic descent and the infinitesimal ODE system's convergence to local optimum. These results imply that ProjFactor stabilizes at a local optimum, provided the step sizes are sufficiently small.

D. More Empirical Analysis

In this section, we present additional empirical results. Specifically, we provide descriptions of the baseline methods used for comparison in Appendix D.1. The loss curves corresponding to different projection granularities are reported in Appendix D.2. Further details on the numerical error testing procedure are elaborated in Appendix D.3. In Appendix D.4, we explore alternative approaches for generating projection matrices. The impact of warm-up steps on our optimization scheme is examined in Appendix D.6, while the effect of projection matrix update frequency is analyzed in Appendix D.5. Additionally, we evaluate the throughput efficiency of different methods in Appendix D.7. Further experimental results are provided in Appendix D.8 and Appendix D.9. A detailed overview of the hyperparameters used in our experiments is given in Appendix D.10. Finally, sample prompts utilized during training are presented in Appendix D.11.

We use the implementation from HuggingFace for the Adam, Adafactor, and LoRA approaches while all other methods were implemented on our own by referencing their respective open-source code repositories. The complete code for our implementations and experiments will be released.

D.1. Description of Baselines

In this section, we provide a brief overview of the finetuning methods compared in our study.

- **Adam** (Kingma & Ba, 2014) is an adaptive gradient-based optimization method that maintains exponentially moving averages of both the first and second moments of gradients. By adjusting step sizes for each parameter individually, Adam often converges faster and requires less finetuning of hyperparameters compared to non-adaptive methods.
- **Adafactor** (Shazeer & Stern, 2018) generalizes the adaptive learning-rate principles of Adam but employs a factored approximation of second-order moments. This factorization reduces memory overhead, making Adafactor particularly suitable for large-scale training while preserving performance benefits similar to Adam.
- **LoRA** (Hu et al., 2022) (Low-Rank Adaptation) provides an efficient approach to finetuning large language models by introducing a trainable, low-rank decomposition into selected layers. This design substantially reduces both memory usage and training time, all while maintaining competitive performance levels.
- **Galore** (Zhao et al., 2024) is a recent optimizer that extends low-rank adaptation techniques by exploiting the low-rank structure in the update gradient rather than in the parameters themselves. Specifically, it applies singular value decomposition (SVD) to construct a projection matrix that projects the original gradient into a subspace.
- **fira** (Chen et al., 2024b) improves upon Galore by combining both the projected gradient and the residual component in the original space. Nonetheless, this design necessitates retaining the original gradient for each update step, which increases memory consumption under gradient accumulation.
- **APOLLO** (Zhu et al., 2024) is a newly introduced, memory-efficient optimization algorithm that approximates channel-wise learning-rate scaling through an auxiliary low-rank optimizer state derived from random projections.

Besides, **FLoRA** (Hao et al., 2024) is equivalent to our methods by setting the granularity factor $c = 1$, which also generates the gradient from Gaussian distribution.

D.2. Loss Curves of Different Projection Granularities

Figure 6 presents the loss curves of two distinct-grained configurations of projections, Fine-Grained Projection ($c = 256, r = 1$), and Ordinary-Grained Projection ($c = 1, r = 256$), evaluated on the Commonsense170k dataset under a fixed Memory Budget $\mathcal{M} = 256$. Figure 6(a) shows the performance when trained LLaMA2-7B with ProjFactor, where both projection configurations rapidly reduce the loss and perform comparably in the initial training phase. However, Fine-Grained Projection demonstrates a clear advantage over Ordinary-Grained Projection as the training continues. In contrast, Figure 6(b) illustrates the results obtained when LLaMA2-7B is trained with the Subspace Scheme (see (6) and right of Figure 2). Here, Fine-Grained Projection consistently outperforms Ordinary-Grained Projection, even in the earlier stages of training.

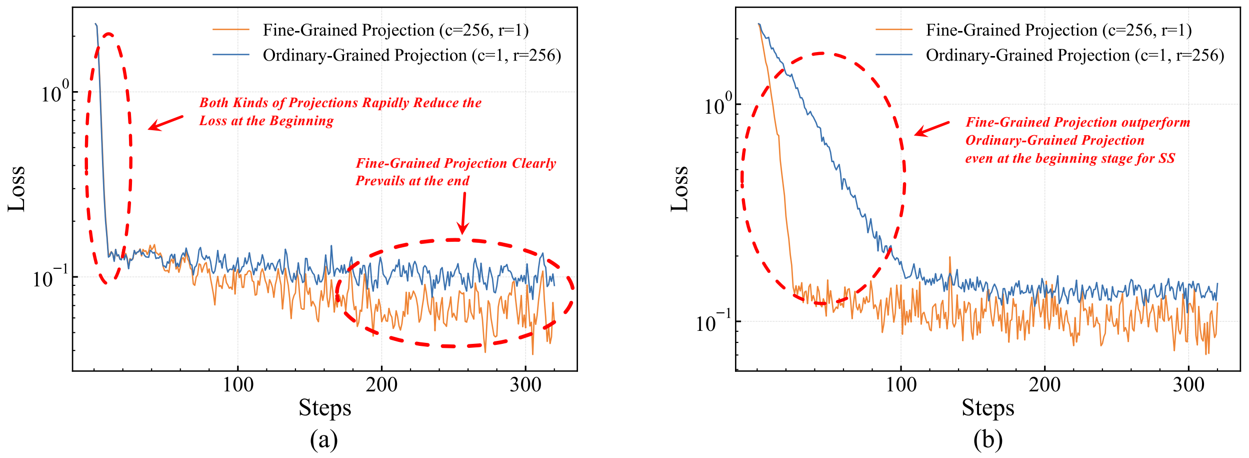


Figure 6: Loss curves of different grained projections on the Commonsense170k dataset: (a) Training LLaMA2-7B with **ProjFactor**; (b) Training LLaMA2-7B with the Subspace Scheme (as described in Section 4). The performance of two types of grained projections is compared under the same memory budget of 256.

D.3. Numerical Error Testing for the Projection Operation

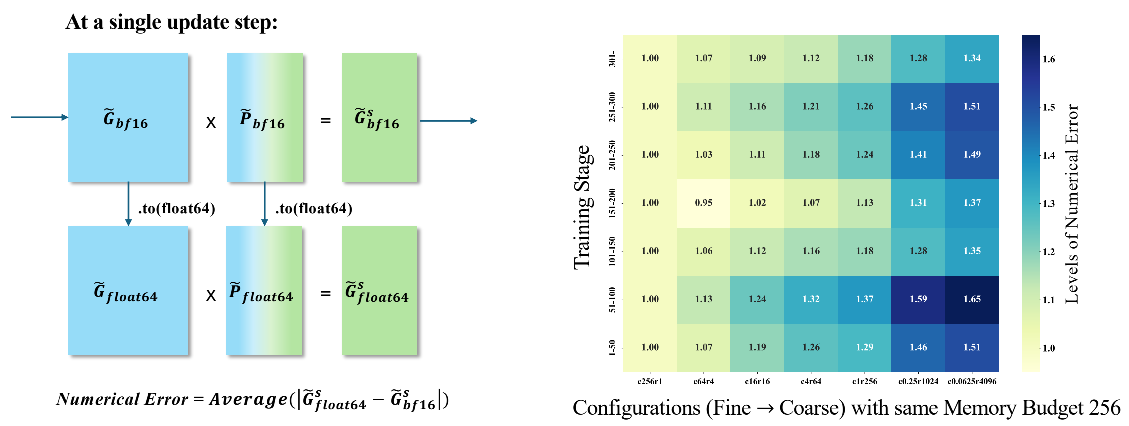


Figure 7: **Left:** Illustration of computational numerical error for a single parameter matrix during an update step. The numerical error of the projection operator is defined as the absolute difference between $\tilde{\boldsymbol{G}}_{bf16}\tilde{P}_{bf16}$ and $\tilde{\boldsymbol{G}}_{float64}\tilde{P}_{float64}$, averaged across all parameter matrices applied low-rank gradient projection. **Right:** Comparison of projections' numerical errors for configurations under a constant memory budget $\mathcal{M} = 256$. The y-axis denotes the training steps, which is divided into 7 stages, while the x-axis is 7 different-grained configurations.

In this section, we elaborate on the numerical error testing discussed in Section 3.3. We evaluate numerical errors introduced by projection operations under varying granularities with a fixed memory budget $\mathcal{M} = 256$. The procedure is shown in the LHS of Figure 7. Specifically, given a VLoRP configuration, at each update step, we compute $\tilde{\boldsymbol{G}}^s = \tilde{\boldsymbol{G}}\tilde{P}$ using bfloat16, denoted as $\tilde{\boldsymbol{G}}^s_{bf16} = \tilde{\boldsymbol{G}}_{bf16}\tilde{P}_{bf16}$. Simultaneously, we create double-precision copies $\tilde{\boldsymbol{G}}_{float64}$ and $\tilde{P}_{float64}$, which are numerically equivalent to their bfloat16 counterparts but retain higher precision (*e.g.*, 0.1100 vs. 0.11). Using these, we compute $\tilde{\boldsymbol{G}}^s_{float64} = \tilde{\boldsymbol{G}}_{float64}\tilde{P}_{float64}$. The discrepancy between $\tilde{\boldsymbol{G}}^s_{float64}$ and $\tilde{\boldsymbol{G}}^s_{bf16}$ reflects numerical errors introduced by low-precision computation. For example, in low precision, $0.11 \times 0.11 = 0.01$, while in high precision, $0.1100 \times 0.1100 = 0.0121$, yielding an error of 0.0021. The optimization is performed using the $\tilde{\boldsymbol{G}}^s_{bf16}$ datatype.

To quantify this error, we compute the absolute element-wise difference between $\tilde{\boldsymbol{G}}^s_{float64}$ and $\tilde{\boldsymbol{G}}^s_{bf16}$, averaging across all elements and parameter matrices subject to low-rank projections. This yields a numerical error metric $\delta_{(c,r)}$ for a given configuration (c, r):

$$\delta_{(c,r)} = \text{Average}_{\forall \tilde{\boldsymbol{G}}^s} \left(\text{Average}_{\forall (\tilde{\boldsymbol{G}}^s)_{ij} \in \tilde{\boldsymbol{G}}^s} \left| ((\tilde{\boldsymbol{G}}^s)_{ij})_{float64} - ((\tilde{\boldsymbol{G}}^s)_{ij})_{bf16} \right| \right) \tag{17}$$

The results, shown on the RHS of Figure 7, examine seven configurations (c, r), ranging from the finest $(c = 256, r = 1)$ to the coarsest $(c = 0.0625, r = 4096)$ under the constraint $\mathcal{M} = 256$. We normalize $\delta_{(c,r)}$ by $\delta_{(c=256,r=1)}$ and compute a moving average over every 50 steps. The numerical error increases almost monotonically as the configuration shifts from fine-grained $(c = 256, r = 1)$ to coarse-grained $(c = 0.0625, r = 4096)$, a trend consistent throughout training. This observation partly explains why finer-grained configurations typically yield better performance given a fixed memory budget $\mathcal{M}$—fine-grained projections have lower numerical errors introduced by using the low-precision datatype which makes the finer granularity of projection (larger c albeit smaller r) a better choice among all the configurations shared a same $\mathcal{M}$.

D.4. Different Ways of Projection Matrix Generation

In this section, we investigate three approaches for generating the projection matrix. The first method, "normal," involves sampling the projection matrix from a standard normal distribution. The second, "Rademacher," samples the projection matrix from the Rademacher distribution, which consists of entries drawn from $\{-1, +1\}$ with equal probability. The third approach, "SVD," constructs the projection matrix using singular value decomposition (SVD), following the methodology proposed in Galore (Zhao et al., 2024). These methods are evaluated on the Commonsense Reasoning task using the LLaMA2-7B model as the testbed. For all experiments, the maximum input sequence length is set to 1024, and the effective batch size is configured to 512. We finetune the model for 1 epoch in total.

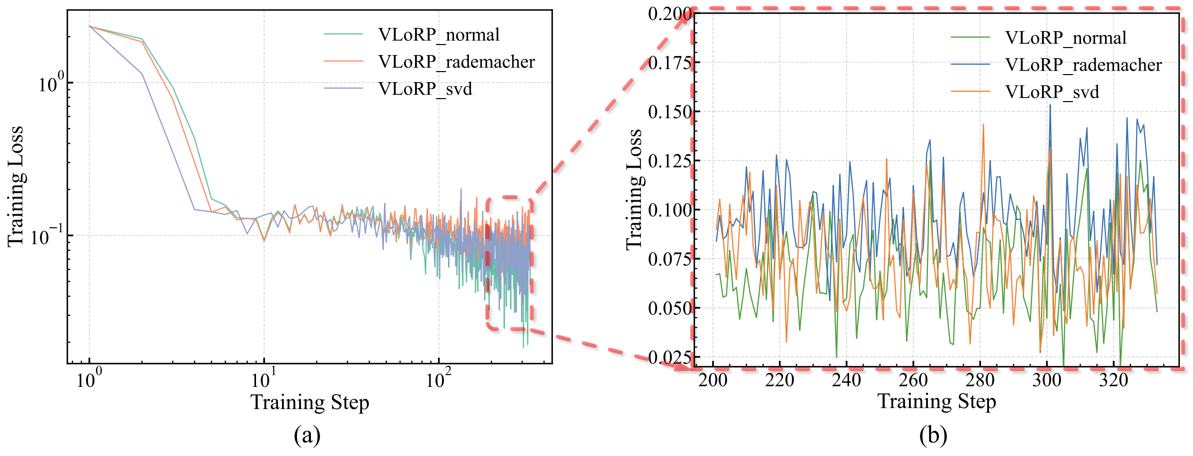


Figure 8: Comparative Analysis of Projection Matrix Generations in LLaMA2-7B Training on the Commonsense170k Dataset: (a). Training loss curves across all training steps; (b). Zoomed-in view of convergence behavior highlighting loss variability among projection methods.

As illustrated in Figure 8(a) and the zoomed-in view in Figure 8(b), the three projection matrix generation methods—normal, Rademacher, and SVD—demonstrate broadly similar training dynamics. Each approach effectively reduces the training loss during the initial steps and ultimately converges to nearly equivalent final loss values. The zoomed-in view highlights a subtle difference, with the normal-based projection slightly outperforming the others in achieving a lower training loss after convergence. These results align with the Johnson–Lindenstrauss lemma, which asserts that in high-dimensional spaces, random projections can effectively preserve geometric properties. Therefore, we adopt the sampling from normal distribution as the method for generating the projection matrix, as it provides computational and storage efficiency without compromising performance.

D.5. Ablation Study on Update Frequency of Projection Matrix

According to recent research on low-rank projection in memory-efficient LLM training (Hao et al., 2024; Zhao et al., 2024; Jaiswal et al., 2024), it is advantageous to keep the vectors $\boldsymbol{v}_i$ constant over several training steps before resampling or reconstructing them, in order to balance the trade-off between variance and biases during training.

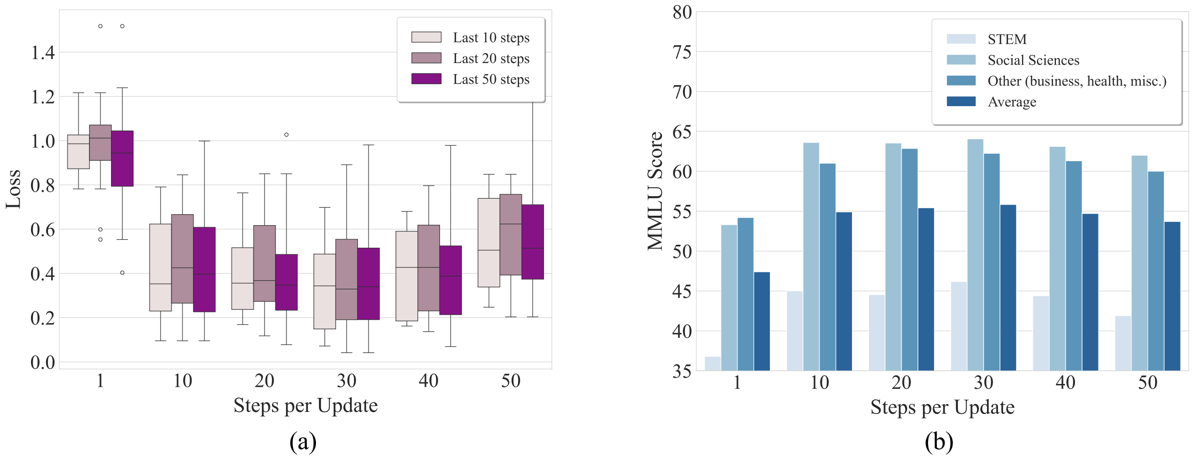


Figure 9: Ablation study on the update frequency of the projection matrix: (a). Training loss statistics across different update frequencies of the projection matrix. "Last n steps" represents the number of final steps used to calculate the statistics; (b). MMLU scores for different categories.

Figure 9 presents an ablation study evaluating the effect of varying the update frequency of the projection matrix on training loss and MMLU performance. Figure 9(a) shows the training loss statistics calculated over the last 10, 20, and 50 steps of training for different update frequencies, while Figure 9(b) illustrates the MMLU scores across STEM, social sciences, other (e.g., business, health), and the overall average for the same frequencies.

In Figure 9(a), the box plots highlight that a lower update frequency, such as 1, results in higher loss values with greater variability. As the update frequency increases, the loss decreases and stabilizes, with frequencies of 20-30 demonstrating relatively consistent performance. This suggests that updating the projection matrix too frequently may introduce high variance leading to the instability of training while a larger interval for updating the projection matrix can cause the model's updates to become constrained within fixed subspaces, leading to a degradation in performance. Figure 9(b) reveals the impact of update frequency on MMLU scores. A similar trend emerges, where frequencies of 30 yield the highest average scores though the performance gain is slight. However, a very low-frequency 1, which updates the projection matrix at each update step results in significantly lower scores across all categories, particularly in STEM and social sciences, indicating the negative impact of high variance on the model's ability to generalize.

D.6. Study on the Warmup Steps

Figure 10 illustrates the effect of varying warm-up steps on the training loss when training our VLoRP framework with ProjFactor. The experiments evaluate five configurations: no warm-up, and warm-up steps set to 10, 20, 50, and 100. The x-axis represents the training steps on a logarithmic scale, while the y-axis shows the training loss.

The results demonstrate that warm-up steps make the early loss curve smoother. Specifically, the configuration without warm-up exhibits a sharp spike in training loss during the initial steps, suggesting instability in optimization at the start of training. In contrast, introducing a warm-up phase helps to mitigate this instability, as evidenced by smoother loss curves for all configurations with warm-up steps. While early-stage differences are prominent, the training loss for all configurations

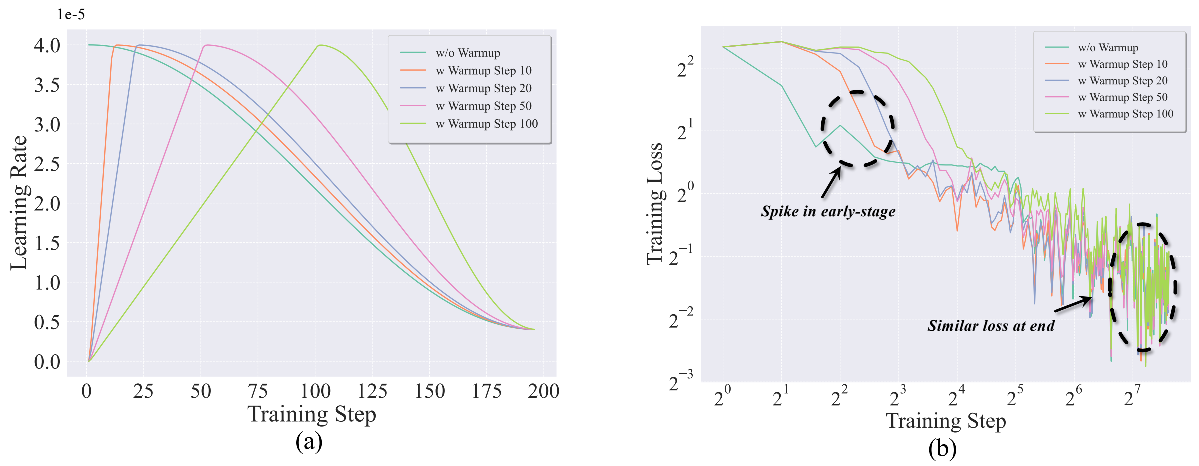


Figure 10: Effect of different warm-up steps on training loss for VLoRP: (a). Learning rate schedules with varying warm-up steps; (b). Impact of different warm-up steps on the model's convergence.

converges to similar values by the end of training. This suggests that the choice of warm-up steps primarily impacts the transient phase of training without significantly affecting the final model performance. Notably, longer warm-up periods incur a trade-off, as they delay the convergence but enhance the stability of training in the initial phase.

D.7. Throughput Analysis

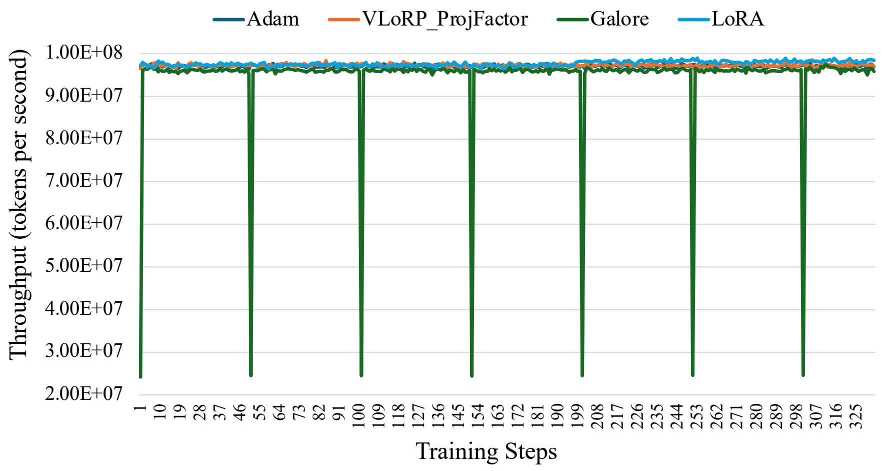


Figure 11: Throughput Analysis of our method, LoRA, Galore, and Adam. Throughput, plotted on the y-axis, is defined as the number of tokens (including padding tokens) processed per second, while the x-axis represents the training step. Please note that the value of throughput would be influenced by factors such as the input sentence length, effective batch size, GPU hardware, and the number of padding tokens.

This section presents a comparative analysis of the throughput performance of VLoRP, LoRA, Galore, and the baseline Adam optimizer during training. For LoRA and Galore, the rank r is set to 256. In the case of VLoRP, the projection matrix is generated by sampling from a standard normal distribution, and the ProjFactor optimization algorithm is employed. For VLoRP, the granularity factor c is set to 256, and the rank r is fixed at 1. These methods are evaluated on the Commonsense Reasoning task using the LLaMA2-7B model as the testbed. All experiments are conducted with a maximum input sequence length of 1024 and an effective batch size of 512.

Figure 11 illustrates the throughput of various methods over the course of 200 training steps. Overall, our method, demonstrates comparable throughput to both LoRA and the baseline Adam optimizer, while outperforming the Galore method. A notable observation is the periodic plunges in throughput experienced by Galore, occurring approximately every 50 iterations. This behavior can be attributed to the singular value decomposition (SVD) operation (Zhao et al., 2024) used to regenerate the projection matrix in the GaLore algorithm, with the regeneration interval explicitly set to 50 iterations, which, according to our experiments, is the optimal update interval for projection matrices in GaLore.

D.8. Experiments with Different Models

Apart from LLaMA2-7B (Touvron et al., 2023), we also evaluated the effectiveness of our VLoRP approach on both a weaker model, GPT2-XL, and a more powerful model, LLaMA3.2-3B (Dubey et al., 2024). The results are presented in Table 4 and Figure 12.

Table 4: Performance Comparison on Commonsense Benchmark Tasks with GPT2-XL. The table presents the results for several baseline methods and different configurations of our proposed VLoRP approach across eight commonsense reasoning tasks. All models are first finetuned on the Commonsense170k (Hu et al., 2023a) dataset and then evaluated separately on different tasks. We set the Memory Budget as 64.

Methods	**ARC_C**	**ARC_E**	**BoolQ**	**HellaSwag**	**OBQA**	**PIQA**	**SIQA**	**winogrande**	**Avg.**
Adam	25.51 $\pm$ 1.27	57.87 $\pm$ 1.01	61.19 $\pm$ 0.85	40.15 $\pm$ 0.49	22.40 $\pm$ 1.87	71.22 $\pm$ 1.06	40.23 $\pm$ 1.11	58.64 $\pm$ 1.38	47.15
Adafactor	25.09 $\pm$ 1.27	57.53 $\pm$ 1.01	59.63 $\pm$ 0.86	39.82 $\pm$ 0.49	23.00 $\pm$ 1.88	71.11 $\pm$ 1.06	40.02 $\pm$ 1.11	59.19 $\pm$ 1.38	46.92
LoRA(r=64)	24.83 $\pm$ 1.26	57.11 $\pm$ 1.02	58.59 $\pm$ 0.86	39.98 $\pm$ 0.49	22.80 $\pm$ 1.88	70.89 $\pm$ 1.06	40.28 $\pm$ 1.11	59.12 $\pm$ 1.38	46.70
Galore(r=64)	25.09 $\pm$ 1.27	57.83 $\pm$ 1.01	59.08 $\pm$ 0.86	40.20 $\pm$ 0.49	22.80 $\pm$ 1.88	71.00 $\pm$ 1.06	40.48 $\pm$ 1.11	57.38 $\pm$ 1.39	46.73
fira(r=64)	25.09 $\pm$ 1.27	57.87 $\pm$ 1.01	59.17 $\pm$ 0.86	40.13 $\pm$ 0.49	22.80 $\pm$ 1.88	70.89 $\pm$ 1.06	40.43 $\pm$ 1.11	58.01 $\pm$ 1.39	46.80
APOLLO(r=64)	24.74 $\pm$ 1.26	58.04 $\pm$ 1.01	59.69 $\pm$ 0.86	39.99 $\pm$ 0.49	22.20 $\pm$ 1.86	70.51 $\pm$ 1.06	40.43 $\pm$ 1.11	58.33 $\pm$ 1.39	46.74
VLoRP									
- $\boldsymbol{c = 2^{-6}, r = 2^{12}}$	24.91 $\pm$ 1.26	57.49 $\pm$ 1.01	58.84 $\pm$ 0.86	40.07 $\pm$ 0.49	23.20 $\pm$ 1.89	71.00 $\pm$ 1.06	40.48 $\pm$ 1.11	59.04 $\pm$ 1.38	46.88
- $\boldsymbol{c = 2^{-4}, r = 2^{10}}$	25.00 $\pm$ 1.27	57.41 $\pm$ 1.01	59.20 $\pm$ 0.86	40.04 $\pm$ 0.49	23.20 $\pm$ 1.89	70.73 $\pm$ 1.06	40.38 $\pm$ 1.11	58.48 $\pm$ 1.38	46.81
- $\boldsymbol{c = 2^{-2}, r = 2^{8}}$	25.00 $\pm$ 1.27	57.37 $\pm$ 1.01	59.51 $\pm$ 0.86	40.01 $\pm$ 0.49	23.00 $\pm$ 1.88	71.00 $\pm$ 1.06	40.48 $\pm$ 1.11	58.56 $\pm$ 1.38	46.87
- $\boldsymbol{c = 2^{0}, r = 2^{6}}$	25.51 $\pm$ 1.27	57.45 $\pm$ 1.01	59.33 $\pm$ 0.86	40.18 $\pm$ 0.49	23.20 $\pm$ 1.89	70.67 $\pm$ 1.06	40.28 $\pm$ 1.11	58.25 $\pm$ 1.39	46.86
- $\boldsymbol{c = 2^{0}, r = 2^{8}}$	25.17 $\pm$ 1.27	57.62 $\pm$ 1.01	59.54 $\pm$ 0.86	40.08 $\pm$ 0.49	22.60 $\pm$ 1.87	71.22 $\pm$ 1.06	40.38 $\pm$ 1.11	58.72 $\pm$ 1.38	46.92
- $\boldsymbol{c = 2^{2}, r = 2^{4}}$	25.09 $\pm$ 1.27	57.62 $\pm$ 1.01	59.79 $\pm$ 0.86	40.08 $\pm$ 0.49	22.80 $\pm$ 1.88	70.89 $\pm$ 1.06	40.33 $\pm$ 1.11	58.96 $\pm$ 1.38	46.94
- $\boldsymbol{c = 2^{4}, r = 2^{2}}$	25.17 $\pm$ 1.27	57.45 $\pm$ 1.01	59.72 $\pm$ 0.86	40.02 $\pm$ 0.49	23.40 $\pm$ 1.90	70.84 $\pm$ 1.06	40.53 $\pm$ 1.11	58.96 $\pm$ 1.38	47.01
$\boldsymbol{c = 2^{6}, r = 2^{0}}$	25.51 $\pm$ 1.27	57.87 $\pm$ 1.01	60.67 $\pm$ 0.85	40.11 $\pm$ 0.49	23.00 $\pm$ 1.88	71.06 $\pm$ 1.06	40.23 $\pm$ 1.11	58.56 $\pm$ 1.38	47.13

Specifically, Table 4 shows the performance of VLoRP on GPT2-XL across eight commonsense reasoning benchmarks. VLoRP consistently outperforms or remains competitive with other PeFT methods such as LoRA, GaLore, and fira. Besides, among different configurations of VLoRP, finer-grained projections ($c = 2^6, r = 1$) achieve the highest average score (47.13), demonstrating the effectiveness of fine-grained low-rank projections.

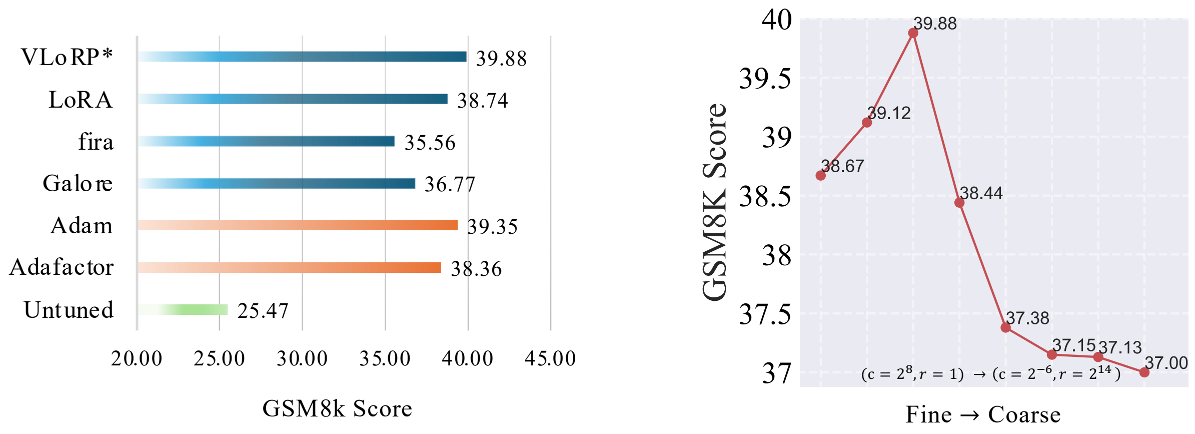


Figure 12: **Left:** Performance comparison of different methods on GSM8K with LLaMA3.2-3B. **Right:** Performance comparison among the configurations of VLoRP with $\mathcal{M} = 256$. The x-axis indicates configurations from fine to coarse (left to right)

Figure 12 further evaluates VLoRP on LLaMA3.2-3B using the GSM8K benchmark. The left subfigure compares VLoRP to other adaptation methods, where VLoRP achieves the highest GSM8K score (39.88). The right subfigure examines the impact of projection granularity on VLoRP's performance. Instead of achieving the best performance at the finest granularity $(c = 2^8, r = 1)$, we find that the slightly coarser configuration $(c = 2^4, r = 2^4)$ reaches the highest GSM8K score of 39.88. However, a general trend where finer-grained configurations yield better results still holds.

D.9. Specific Statistics of Figure 4

In this section, we present the detailed data for Figure 4, which displays the average performance across eight commonsense reasoning tasks. From Table 5, Table 6, and Table 7, it is evident that the finer configuration (larger c and smaller r) consistently outperforms coarser configurations across nearly all tasks when evaluated under the same memory budget.

Table 5: Performance Comparison on Commonsense Benchmark Tasks (Memory Budget 256).

Configurations	ARC_C	ARC_E	BoolQ	HellaSwag	OBQA	PIQA	SIQA	winogrande	Avg.
- $\boldsymbol{c = 2^{-6}, r = 2^{14}}$	42.92 ± 1.45	76.22 ± 0.87	79.27 ± 0.71	57.53 ± 0.49	32.60 ± 2.10	77.91 ± 0.97	46.72 ± 1.13	69.85 ± 1.29	60.38
- $\boldsymbol{c = 2^{-4}, r = 2^{12}}$	43.34 ± 1.45	76.26 ± 0.87	79.54 ± 0.71	57.58 ± 0.49	32.00 ± 2.09	77.64 ± 0.97	46.72 ± 1.13	70.01 ± 1.29	60.39
- $\boldsymbol{c = 2^{-2}, r = 2^{10}}$	43.34 ± 1.45	76.30 ± 0.81	79.45 ± 0.71	57.47 ± 0.49	32.20 ± 2.09	77.75 ± 0.97	46.78 ± 1.13	70.01 ± 1.29	60.41
- $\boldsymbol{c = 2^{0}, r = 2^{8}}$	43.69 ± 1.45	77.02 ± 0.86	79.27 ± 0.71	57.49 ± 0.49	31.80 ± 2.08	78.07 ± 0.97	47.49 ± 1.13	69.77 ± 1.29	60.57
- $\boldsymbol{c = 2^{2}, r = 2^{6}}$	44.03 ± 1.45	76.81 ± 0.87	79.17 ± 0.71	57.59 ± 0.49	31.80 ± 2.08	78.02 ± 0.97	47.19 ± 1.13	69.53 ± 1.29	60.53
- $\boldsymbol{c = 2^{4}, r = 2^{4}}$	44.71 ± 1.45	77.27 ± 0.86	79.42 ± 0.71	57.50 ± 0.49	32.20 ± 2.09	77.86 ± 0.97	47.54 ± 1.13	70.09 ± 1.29	60.82
- $\boldsymbol{c = 2^{6}, r = 2^{2}}$	44.97 ± 1.45	77.65 ± 0.85	80.46 ± 0.69	57.56 ± 0.49	33.60 ± 2.11	77.97 ± 0.97	48.06 ± 1.13	69.69 ± 1.29	61.25
$\boldsymbol{c = 2^{8}, r = 2^{0}}$	45.56 ± 1.46	77.78 ± 0.85	80.58 ± 0.69	57.59 ± 0.49	34.00 ± 2.12	77.86 ± 0.97	48.16 ± 1.13	69.69 ± 1.29	61.40

Table 6: Performance Comparison on Commonsense Benchmark Tasks (Memory Budget 64).

Configurations	ARC_C	ARC_E	BoolQ	HellaSwag	OBQA	PIQA	SIQA	winogrande	Avg.
- $\boldsymbol{c = 2^{-6}, r = 2^{12}}$	42.83 ± 1.45	76.09 ± 0.87	78.99 ± 0.71	57.54 ± 0.49	31.80 ± 2.08	77.86 ± 0.97	46.32 ± 1.13	69.77 ± 1.29	60.15
- $\boldsymbol{c = 2^{-4}, r = 2^{10}}$	43.00 ± 1.45	76.14 ± 0.87	78.99 ± 0.71	57.55 ± 0.49	31.80 ± 2.08	77.97 ± 0.97	46.16 ± 1.13	69.53 ± 1.29	60.14
- $\boldsymbol{c = 2^{-2}, r = 2^{8}}$	43.09 ± 1.45	76.30 ± 0.87	78.78 ± 0.72	57.57 ± 0.49	31.40 ± 2.08	77.97 ± 0.97	46.16 ± 1.13	69.06 ± 1.30	60.04
- $\boldsymbol{c = 2^{0}, r = 2^{6}}$	43.52 ± 1.45	76.56 ± 0.87	79.02 ± 0.71	57.46 ± 0.49	32.00 ± 2.09	78.07 ± 0.97	46.21 ± 1.13	69.38 ± 1.30	60.28
- $\boldsymbol{c = 2^{2}, r = 2^{4}}$	43.43 ± 1.45	76.43 ± 0.87	79.11 ± 0.71	57.53 ± 0.49	31.40 ± 2.08	78.02 ± 0.97	46.26 ± 1.13	69.46 ± 1.29	60.21
- $\boldsymbol{c = 2^{4}, r = 2^{2}}$	44.03 ± 1.45	76.56 ± 0.87	79.36 ± 0.71	57.51 ± 0.49	32.40 ± 2.10	77.86 ± 0.97	46.88 ± 1.13	69.61 ± 1.29	60.53
$\boldsymbol{c = 2^{6}, r = 2^{0}}$	45.39 ± 1.45	77.19 ± 0.87	79.91 ± 0.71	57.46 ± 0.49	33.20 ± 2.11	77.91 ± 0.97	47.70 ± 1.13	69.93 ± 1.29	61.09

Table 7: Performance Comparison on Commonsense Benchmark Tasks (Memory Budget 16).

Configurations	ARC_C	ARC_E	BoolQ	HellaSwag	OBQA	PIQA	SIQA	winogrande	Avg.
- $\boldsymbol{c = 2^{-6}, r = 2^{10}}$	42.41 ± 1.44	76.18 ± 0.87	78.69 ± 0.72	57.50 ± 0.49	31.60 ± 2.08	78.07 ± 0.97	46.11 ± 1.13	69.30 ± 1.30	59.98
- $\boldsymbol{c = 2^{-4}, r = 2^{8}}$	43.00 ± 1.45	76.14 ± 0.87	78.90 ± 0.71	57.33 ± 0.49	31.20 ± 2.07	78.40 ± 0.96	45.85 ± 1.13	69.46 ± 1.29	60.03
- $\boldsymbol{c = 2^{-2}, r = 2^{6}}$	42.75 ± 1.45	76.35 ± 0.87	78.65 ± 0.72	57.37 ± 0.49	31.80 ± 2.08	78.18 ± 0.96	45.85 ± 1.13	69.30 ± 1.30	60.03
- $\boldsymbol{c = 2^{0}, r = 2^{4}}$	43.43 ± 1.45	76.22 ± 0.87	79.08 ± 0.71	57.56 ± 0.49	32.00 ± 2.09	78.02 ± 0.97	46.37 ± 1.13	69.69 ± 1.29	60.30
- $\boldsymbol{c = 2^{2}, r = 2^{2}}$	43.26 ± 1.45	76.39 ± 0.87	78.62 ± 0.72	57.49 ± 0.49	31.60 ± 2.08	77.97 ± 0.97	46.57 ± 1.13	69.06 ± 1.30	60.12
$\boldsymbol{c = 2^{4}, r = 2^{0}}$	44.88 ± 1.45	76.94 ± 0.86	79.39 ± 0.71	57.63 ± 0.49	33.20 ± 2.11	78.07 ± 0.97	47.49 ± 1.13	69.46 ± 1.29	60.88

D.10. Implementation Details

For all datasets, experiments were conducted on an **NVIDIA A100 GPU (80GB)** using the **bfloat16** datatype. By default, we applied our proposed ProjFactor method to VLoRP. For other low-rank-based PeFT methods, such as LoRA and GaLore, we set the rank to $r = \mathcal{M}$ for a fair comparison, where $\mathcal{M}$ is the memory budget defined in Section 3. The training process follows a **batch size of 16** with **gradient accumulation over 32 steps**, yielding an **effective batch size of 512**. The **maximum input sequence length** is **1024 tokens**, and **activation checkpointing** is enabled to optimize memory usage. Learning rates (η) are set to the best values found through empirical testing: 2×10^{-5} for Commonsense Reasoning, 4×10^{-5} for MMLU, and 10^{-4} for GSM8K. Notably, for GSM8K, we observed that Galore and fira require relatively larger learning rates, while Apollo performs badly on this benchmark. Regarding training duration, models are trained for **one epoch** on Commonsense Reasoning (333 iterations) and MMLU (196 iterations). For GSM8K, all models are trained for **three epochs** (138 iterations) due to: (1) the higher difficulty of GSM8K questions compared to the other benchmarks and (2) the significantly smaller training dataset size.

Besides, the reported memory usage in Section 5 refers to the actual GPU memory allocated once the model has stabilized, rather than the maximum memory reserved by `PyTorch`. The latter one is typically displayed by commands such as `watch nvidia-smi` and is often larger than the former counterpart.

D.11. Showcase of Training Prompts

✓ Example 1: CommonSense 170K

### Instruction:
Please choose the correct ending to complete the given sentence: Getting a tattoo: Man is kneeling in front ofa woman and is making a tattoo on her right foot. woman wearing a jean skirt

Ending1: is puting a tattoo and making it in her cheek. Ending2: is sitting in front of a man getting a tattoo. Ending3: is sitting in a living room of a house, in front of is wall painted black and there is pullerous dotted weeblt on the walls and the floor. Ending4: is in the middle of a room and c tool is in her left hand and in her left hand she is holding a candy and a black razor is on her right knee.

Answer format: *ending1/ending2/ending3/ending4*

### Response:
the correct answer is ending2

✓ Example 2: CommonSense 170K

### Instruction:
Please choose the correct answer to fill in the blank to complete the given sentence: In dire need of a new kidney, Ryan soug ht out Jason so _______ could agree to donate him one.

Option1: Ryan Option2: Jason Answer format: option1/option2

### Response:
the correct answer is option2

✓ Example 3: MMLU

The following are multiple choice questions (with answers).

Question: Isabella Stewart was born in New York City in 1840. Her father made a great deal of money in trade. During school,her parents took her to Italy to explore the country's many cultural treasures. One of the private art collections Isabella visit ed in Milan had a deep influence on her. She wrote to her friends about her dream of owning a house one day with an art collection like the one she had seen in Italy. In Paris, Isabella became a close friend of one of her classmates, Julia Gardner, whose family was from Boston . Julia would later introduce Isabella to her brother, Jack. In 1860, Isabella Stewart married Jack Gardner. The couple had too much art to fit inside their home. So they decided to start planning a museum. Mrs. Gardner didn't like the cold and empty spaces of many museums during her time. She wanted a warm museum filled with light. She once said that she decided years ago that _ . America was a young country developing quickly in other areas. But the country needed more chances for people to see beautiful examples of art. A fter her husband's death in 1898, Isabella knew she had no time to lose in building her museum. She bought land, hired a building desi gner, and supervised every detail of her museum's construction. Mrs. Gardner opened her museum on January 1,1903. The museum was then called Fenway Court. She invited her friends that night for a special musical performance. The next month, she opened the mus eum to the public. At first, visits were limited to twenty days out of the year. Visitors paid one dollar to enter. Isabella Stewart Gardner died in 1924 in Boston. In her will, she left the museum a million dollars and a series of requirements about how it should be manage d. One requirement is that the permanent collection cannot be changed. From the passage, we can learn that the museum _ .

A. helps earn much money for its collections of art
B. is called Fenway Court by the visitors
C. was opened to the public on January 1st, 1903
D. is still affected by Isabella Gardner in management now
Answer:*D*

Example 4: MMLU

The following are multiple choice questions (with answers).

Question: Where do you like to live? For this question, different people have different answers. Some people like to live in a city because there are many shops and supermarkets. They think it is convenient to buy things. But some people think it is good to live in a quiet town because they don't like the dirty air in the big city. They dislike pollution in the city. Today, some people like travelling, so they would like to buy house cars. House car is both a house and a car. You can't buy it with a little money. There is a driv ing area in the car. You can do lots of things in the car. There is a bed and a lamp in the bedroom. You can make dinner in the kitchen. You can also find a fridge and a sink in it. You can listen to music and watch TV in the sitting room. If you are tired, you can have a shower or a bath in the bathroom. You can do most things you want to do. Life is travelling. Do you want to live in this kind of car? What's the advantage of living in the town?

A. There are no cars and buses.
B. There isn't much pollution.
C. There aren't any places to buy things.
D. The air is dirty there.
Answer:*B*

Example 5: GSM8K

Question: Kimiko watches four YouTube videos. The first video is 2 minutes long, the second video is 4 minutes and 30 seconds, and the last two videos are equal in length. If she spends a total of 510 seconds watching YouTube, how many seconds long was eac h of the last two videos?
Answer: *First convert the length of the first video to seconds: 2 minutes * 60 seconds/minute = <<2*60=120>>120 seconds*
*Then convert the length of the second video to seconds: 4 minutes * 60 seconds/minute + 30 seconds = 240 seconds + 30 seconds = <<4*60+30=270>>270 seconds*
Now subtract the length of the first two videos from the total time Kimiko spent watching to find the combined length of the last two videos: 510 seconds - 120 seconds - 270 seconds = <<510-120-270=120>>120 seconds
Now divide the combined length by the number of videos to find each video's length: 120 seconds / 2 = <<120/2=60>>60 seconds
#### 60

Example 6: GSM8K

Question: A business executive is going on a four day vacation where he will be unable to answer emails. The first day he is gone, he receives 16 new emails. On each of the following days, he receives half as many new emails as he received on the prior day. At the end of his four day vacation, how many new emails will he have received in total?
Answer: *On the second day, the executive receives 16/2 = <<16/2=8>>8 new emails.*
On the third day, he receives 8/2 = <<8/2=4>>4 new emails.
On the fourth day, he receives 4/2 = <<4/2=2>>2 new emails.
Therefore, during the entire trip he will have received 16 + 8 + 4 + 2 = <<16+8+4+2=30>>30 new emails.
#### 30

Example 7: GSM8K

Question: At the end of a circus act, there are 12 dogs on stage. Half of the dogs are standing on their back legs and the other hal f are standing on all 4 legs. How many dog paws are on the ground?
Answer: *There are 12 dogs and half are standing on their back legs so that means 12/2 = <<12/2=6>>6 are standing on their back legs*
*A dog has 2 back legs and only 6 are standing on their back legs meaning that there are 2*6 = <<2*6=12>>12 paws on the ground*
*The other 6 dogs are standing on all 4 legs which means these 6 dogs have 6*4 = <<6*4=24>>24 paws on the ground*
When you add them together 12+24 = <<12+24=36>>36 paws on the ground
#### 36

E. Related Works

E.1. Parameter-Efficient Finetuning

To mitigate the substantial costs associated with finetuning large-scale models, Parameter-Efficient finetuning (PEFT) methods have been introduced. These techniques adapt models to downstream tasks by training only a small fraction of the total parameters. Existing PEFT approaches can be broadly categorized into three primary families. The first category comprises adapter-based methods (Houlsby et al., 2019; He et al., 2022; Mahabadi et al., 2021), which integrate additional trainable modules into the otherwise frozen backbone network. For instance, Houlsby et al. (2019) proposes the sequential addition of linear modules to existing layers, while He et al. (2022) introduces the integration of these modules in parallel with the original layers to enhance performance. The second category includes prompt-based methods (Lester et al., 2021; Razdaibiedina et al., 2023; Wang et al., 2023), which augment the initial input with extra soft tokens, focusing solely on finetuning these trainable vectors. However, prompt-based methods often face challenges stemming from sensitivity to initialization, which can impede their overall effectiveness. Notably, both adapter-based and prompt-based methods, whether modifying the model's input or architecture, tend to increase inference latency compared to the baseline model. The third category is the low-rank-based methods (*e.g.* LoRA) which exploit low-rank properties inside the training procedure, which we will discuss comprehensively in the next.

E.2. Low-Rank Based Memory-Efficient Finetuning

By using two low-rank matrices to estimate the increment of pre-trained weights without incurring additional inference overhead, LoRA (Hu et al., 2022) and its improved variants have achieved remarkable success in the field of PeFT. For example, QLoRA (Dettmers et al., 2023) combines low-bit quantization with LoRA to facilitate the finetuning of LLMs. AdaLoRA (Zhang et al., 2023) dynamically allocates the parameter budget across weight matrices based on importance scores, optimizing the use of trainable parameters. Additionally, methods such as VeRA (Kopiczko et al., 2024) reduce the number of trainable parameters by employing a single pair of low-rank matrices shared across all layers, learning small scaling vectors instead. Wang et al. (2024) align the gradients of the low-rank matrix product with those from full finetuning at the initial step, achieving competitive results. Furthermore, Hayou et al. (2024) explore adjusting the learning rates of the LoRA adapter matrices independently, enhancing feature learning efficiency.

Recently, another line of research (Zhao et al., 2024; Hao et al., 2024; Jaiswal et al., 2024) has re-implemented LoRA methods from the perspective of low-rank gradient projection. Hao et al. (2024) investigated the training dynamics of LoRA methods and found that vanilla LoRA (Hu et al., 2022) can be approximated by a process of randomly projecting gradients to a low-rank subspace and then projecting back. Zhao et al. (2024) followed a similar idea but chose to obtain the projection matrix by performing Singular Value Decomposition (SVD) on the gradients to implement Galore, instead of using a randomly sampled matrix as in FloRA (Hao et al., 2024). Chen et al. (2024b) extends Galore by incorporating the residual error between the full-rank gradient and its low-rank approximation, effectively simulating full-rank updates. APOLLO (Zhu et al., 2024) approximates channel-wise learning-rate scaling through an auxiliary low-rank optimizer state derived from random projections.

E.3. The Equivalence between LoRA and LoRP

Hao et al. (2024) has shown that LoRA is approximately equivalent to an approach that compresses the gradient updates by down-projecting them onto a lower-dimensional space, and then projecting back to the original space:

Theorem E.1 (Hao et al. (2024)). *Consider a weight matrix $W \in \mathbb{R}^{n\times m}$ with the low-rank factorization $\Delta W = BA$ where we initially have $B_0 = \mathbf{0}^{n\times r}$ and $A_0 \in \mathbb{R}^{r\times m}$ randomly sampled from a standard Gaussian distribution. Assuming that the learning rate η is sufficiently small, then the LoRA training procedure effectively restricts weight updates to the column space of A_0. Moreover, after T gradient-based update steps, the resulting weight matrix W_T satisfies*

$$W_T \approx W_0 - \eta \sum_{t=0}^{T-1} G_t A_0 A_0^\top / r, \tag{18}$$

where G_t denotes the gradient of W at the t-th step.

The rationale of Theorem E.1 is grounded in the Johnson–Lindenstrauss lemma and its extensions (Dasgupta & Gupta, 2003; Matousek, 2008; Indyk & Motwani, 1998), which assert that random projections via Gaussian matrices approximately preserve the geometry with high probability. Moreover, Hao et al. (2024) quantifies the reconstruction error, demonstrating that the rank r needs only to scale logarithmically to maintain low element-wise error, thus ensuring computational and memory efficiency.

E.4. Stochastic Approximation

Stochastic approximation methods (Robbins & Monro, 1951; Nevel'son & Has' minskii, 1976; Spall, 1992) are a family of iterative methods primarily used to solve root-finding or optimization problems. These methods address functions of the form $f(\theta) = \mathbb{E}_{\boldsymbol{z}}[F(\theta, \boldsymbol{z})]$, which represent the expected value of a function F that depends on a random variable z. The goal is to infer properties of f without directly evaluating it. We focus primarily on its applications involving gradient estimation.

Forward Gradient. FG methods (Wengert, 1964; Silver et al., 2021; Baydin et al., 2022; Ren et al., 2022) update model parameters using directional gradients along multiple random perturbation directions and belong to the family of stochastic approximation techniques. More formally, given a differentiable function $f : \mathbb{R}^N \to \mathbb{R}$, the gradient at a given input $\theta \in \mathbb{R}^N$ can be approximated as

$$\hat{\nabla} f(\theta) := \left(\nabla f(\theta)^\top z\right) z. \tag{19}$$

There are multiple choices for the random variables z. All distributions satisfying $\mathbb{E}[z] = 0$ and $\mathbb{E}[zz^\top] = I_N$ are qualified, such as the standard Gaussian distribution and the Rademacher distribution. This way, for any given θ, $\hat{\nabla} f(\theta)$ is also an unbiased estimator of $\nabla f(\theta)$ since

$$\begin{aligned} \mathbb{E}[\hat{\nabla} f(\theta)] &= \mathbb{E}\left[\left(\nabla f(\theta)^\top z\right) z\right] = \mathbb{E}\left[zz^\top\right] \nabla f(\theta) \\ &= I_N \nabla f(\theta) = \nabla f(\theta). \end{aligned} \tag{20}$$

In practice, to reduce variance, Monte Carlo gradient estimation can be performed by averaging forward gradients over multiple random directions (Baydin et al., 2022; Hu et al., 2023b). Utilizing forward-mode automatic differentiation techniques (Williams & Zipser, 1989; Pearlmutter, 1994), the Jacobian-vector product $\nabla_\theta f(\theta)^\top z$ can be computed efficiently with a single forward pass. This enables forward gradient learning (Wengert, 1964; Silver et al., 2021; Baydin et al., 2022; Ren et al., 2022), which updates model parameters based on the directional gradient along a random perturbation direction, enabling backpropagation-free training.