Unlocking More Granular Control of Memory-Efficient LLM Finetuning

Yezhen Wang[1] , **Zhouhao Yang**[2] , **Fanyi Pu**[3] , **Kenji Kawaguchi**[1]
[1]NUS School of Computing
[2]JHU Applied Mathematics and Statistics
[3]NTU College of Computing and Data Science
yezhenw@comp.nus.edu.sg, zyang145@jh.edu, fpu001@e.ntu.edu.sg, kenji@nus.edu.sg

Abstract

Low-rank gradient projection (LoRP) has recently emerged as a memory-efficient alternative to low-rank adapters (LoRA) for finetuning large language models. Existing LoRP methods, however, implicitly fix the projection unit to a single gradient row, leaving the effect of grouping multiple rows (or subdividing a row) largely unexplored. In this work, we systematically investigate the impact of the projection unit on LoRP methods. Specifically, we extend existing LoRP approaches by introducing an additional degree of freedom, *projection granularity*, beyond the traditional rank hyperparameter. This enables a framework capable of performing **V**arious-grained **Lo**w-**R**ank **P**rojection of gradients, which we term **VLoRP**. Using VLoRP, we observe that, under an identical memory budget, fine-grained projections consistently deliver superior performance. Moreover, VLoRP requires no extra computation and minimal code changes, effectively providing a no-cost accuracy boost to LoRP. Finally, we provide convergence analysis on VLoRP with either SGD or an Adam-based memory-efficient optimizer, and extensive experiments are conducted to validate our findings, covering tasks such as Commonsense Reasoning, MMLU, and GSM8K.

1 Introduction

Large Language Models (LLMs) are powerful but costly to finetune due to their substantial memory requirements. For example, finetuning an LLM with 7 billion parameters in the bfloat16 format [Dean *et al.*, 2012] requires approximately 14 GB of memory for the parameters alone. The gradients demand a similar amount of memory,[1] and, with Adam/AdamW [Kingma and Ba, 2014; Loshchilov and 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.

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

To mitigate this training overhead, various parameter-efficient finetuning (PeFT) techniques [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; 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; 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 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.2):

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

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 equation 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 equation 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 with r to balance performance and memory efficiency.**

Accordingly, we propose **VLoRP** (**V**arious-Grained **Lo**w-**R**ank **P**rojection of Gradients), a LoRP framework that introduces a new degree of freedom, *projection granularity*, defined as the length of each row segment in the gradient matrix G. Unlike existing LoRP methods that implicitly fix the projection unit to one gradient row, VLoRP reshapes the gradient matrix before projection, thereby jointly controlling projection granularity and rank. This enables a more flexible memory-performance trade-off and can be implemented with only lightweight reshaping operations, incurring no additional computational overhead.

To support Adam-style optimization under projected-gradient storage, we further propose **ProjFactor**, a memory-efficient Adam-like optimizer that avoids access to full-rank gradients. Leveraging VLoRP, we systematically study different granularity-rank configurations under a fixed *memory budget*, and observe that finer-grained projections consistently outperform coarser alternatives. We also provide theoretical guarantees: VLoRP with SGD achieves an $O(1/T)$ convergence rate, and ProjFactor admits a monotonically decreasing Lyapunov function under the Hamiltonian descent framework [Maddison *et al.*, 2018; Chen *et al.*, 2023; Liang *et al.*, 2024].

Our contributions are summarized as follows:

1. We introduce VLoRP, which extends existing LoRP methods by jointly controlling projection granularity and rank.
2. We propose ProjFactor, a memory-efficient Adam-like optimizer for VLoRP.
3. We show that, under the same memory budget, finer projection granularity is more effective than increasing rank.
4. We provide convergence guarantees for VLoRP with both SGD and ProjFactor.

2 Exploring Various Granularities of Low-Rank Gradient Projection

2.1 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:

$$G^s := GP, \quad G^o := \gamma G^s P^\top = \gamma GPP^\top, \tag{2}$$

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 is usually set as 1. In practice, only the matrix $G^s \in \mathbb{R}^{n\times r}$ needs to be stored with $r \ll \min\{m, n\}$. Several strategies exist for constructing the projection matrix P. The classical choice is to extract the top singular vectors of the gradient matrix via Singular Value Decomposition (SVD) [Zhao *et al.*, 2024]. However, recent studies [Hao *et al.*, 2024] demonstrate that a matrix whose columns are sampled independently from a high-dimensional isotropic Gaussian distribution preserves the relevant geometric structure equally well—a result anticipated by the Johnson–Lindenstrauss lemma [Matousek, 2008; Dasgupta and Gupta, 2003; Indyk and Motwani, 1998] and corroborated by our experiments. Thus, we adopt Gaussian-random projections as the default. Nevertheless, the proposed framework readily accommodates projection matrices obtained by any other method.

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

$$G^o = \frac{1}{r}\sum_{j=1}^{r} Gv_j v_j \top = \begin{pmatrix} \frac{1}{r}\sum\limits_{j=1}^{r} (G_{1,:}\cdot v_j)\, v_j^\top \\ \vdots \\ \frac{1}{r}\sum\limits_{j=1}^{r} (G_{n,:}\cdot v_j)\, v_j^\top \end{pmatrix}, \tag{3}$$

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. Equation (3) 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 [Wengert, 1964; Silver *et al.*, 2021; Baydin *et al.*, 2022; Ren *et al.*, 2022], belonging to the broader family of stochastic approximation methods [Robbins and Monro, 1951; Nevel'son and Has' minskii, 1976; Spall, 1992].

Observation 1. *Equation equation 3 indicates that LoRP can be interpreted as a row-wise application of forward gradient estimation with r samples for each gradient matrix.*

Motivation. 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) Alternatively, the estimation can be applied element-wise: for each one-dimensional parameter $\xi \in \theta$, the 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 cases demonstrate that stochastic approximation can be applied to parameters of varying sizes, leading to different methods. Naturally, according to Observation 1, the Projection Granularity, defined by the row size of G, can also be

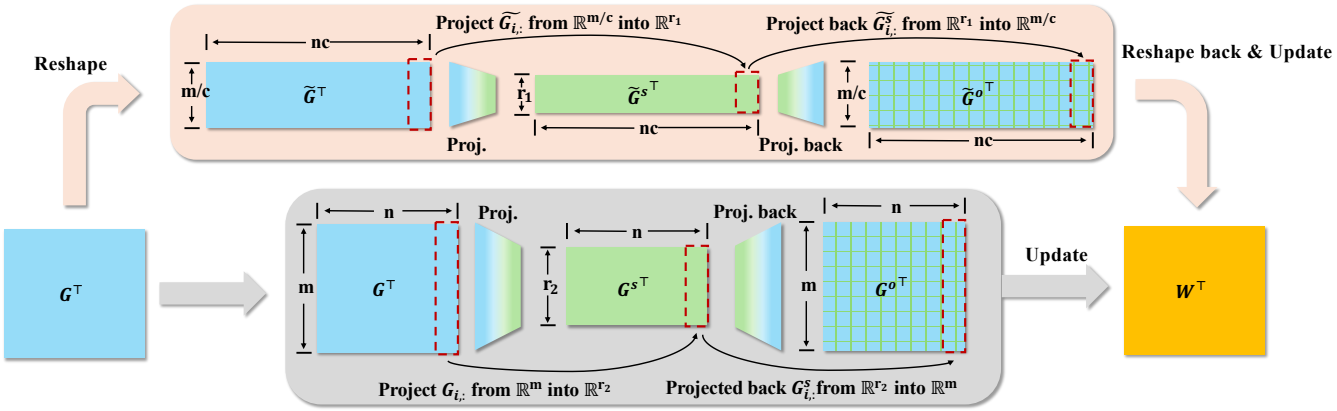

Figure 1: Overview of VLoRP versus standard LoRP. *Bottom (gray)* : In ordinary LoRP, the gradient matrix G is directly projected row-by-row from $\mathbb{R}^{n\times m}$ into $\mathbb{R}^{n\times r}$ and stored as G^s. *Top (pale beige)*: In contrast, VLoRP reshapes the original gradient matrix G first 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 $W \in \mathbb{R}^{n\times m}$.

adjusted in LoRPs. This insight motivates our exploration of low-rank gradient projections with varying granularities.

2.2 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 reshapes 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:

$$\begin{aligned} \tilde{G}^s &:= \underbrace{\text{Reshape}\left(G, [nc, \tfrac{m}{c}]\right)}_{\text{denoted as } \tilde{G}} \tilde{P}, \\ G^o &:= \text{Reshape}\Big(\underbrace{\tilde{G}^s \tilde{P}^\top}_{\text{denoted as } \tilde{G}^o}, [n, m]\Big), \end{aligned} \tag{4}$$

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, Equation (2) 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, 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 substantive distinction between VLoRP and existing LoRP variants is an additional pair of tensor-reshaping operations, which imposes *no extra computational overhead*.

2.3 Optimization: Adam-based Memory-Efficient Schemes

Since our problem setting focuses on memory-efficient fine-tuning, the original full-rank gradients are not stored in principle. However, the Adam optimizer requires access to the full-rank gradients to update its states, making it memory inefficient if directly applied in this context. Therefore, in this section, we first investigate two potential Adam-based optimization schemes for VLoRP, and subsequently introduce a memory-efficient variant, **ProjFactor**, for the superior one. We enable gradient accumulation [Smith *et al.*, 2018] for the projected gradient by default, which means at each update stage, we can only access $\tilde{\boldsymbol{G}}^s = \sum_{i=1}^K \tilde{G}^s_i/K$, 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 $\tilde{\Delta}^s_t = \tilde{m}^s_t/\sqrt{\tilde{v}^s_t}$ 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

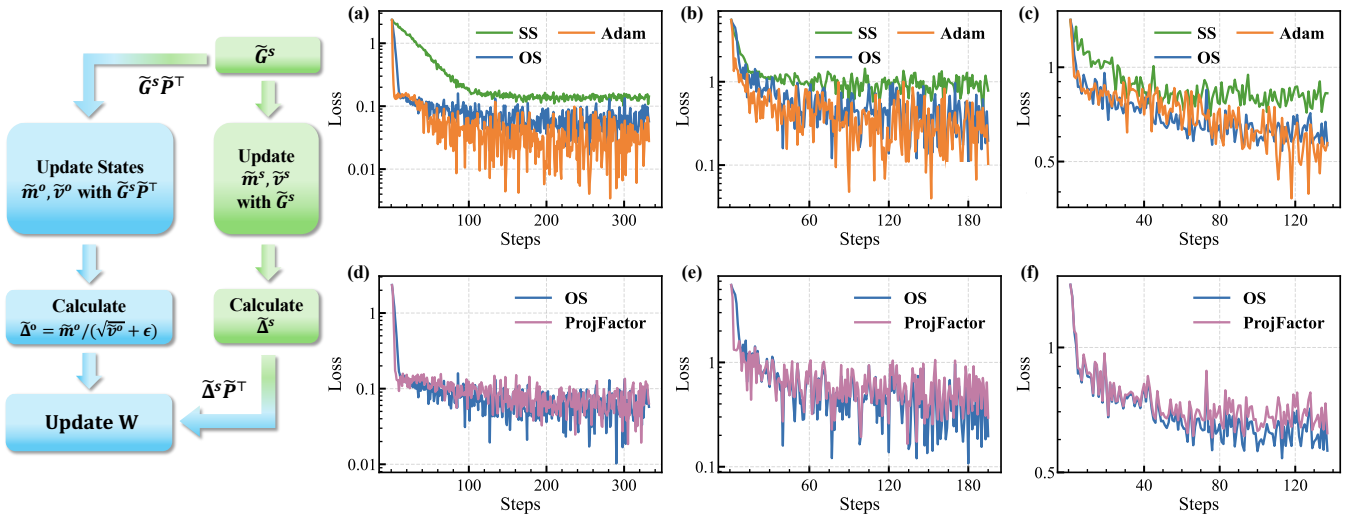

Figure 2: **Left:** Schematic illustration of the Subspace Scheme (SS, green) operating in a learned subspace, and the Original Scheme (OS, blue) operating in the original space. **Right (top row, panels a–c):** Fine-tuning loss curves of SS (green), OS (blue), and Adam (orange) on three tasks, showing that OS outperforms SS by a large margin while having 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 dynamics 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.

schemes are defined as follows[2]:

$$
\begin{aligned}
\textbf{SS:}\quad & \tilde{m}_t^s = \beta_1 \tilde{m}_{t-1}^s + (1-\beta_1)\tilde{\boldsymbol{G}}_t^s \\
& \tilde{v}_t^s = \beta_2 \tilde{v}_{t-1}^s + (1-\beta_2)(\tilde{\boldsymbol{G}}_t^s)^{\odot 2} \\
& \tilde{\Delta}_t^s = \tilde{m}_t^s / \sqrt{\tilde{v}_t^s} \\
& W_t = W_{t-1} - \eta\, \mathrm{Reshape}\left(\tilde{\Delta}_t^s \tilde{P}^\top, [n,m]\right) \\
\textbf{OS:}\quad & \tilde{m}_t^o = \beta_1 \tilde{m}_{t-1}^o + (1-\beta_1)(\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top) \\
& \tilde{v}_t^o = \beta_2 \tilde{v}_{t-1}^o + (1-\beta_2) \\
& \tilde{\Delta}_t^o = \tilde{m}_t^o / \sqrt{\tilde{v}_t^o} (\tilde{\boldsymbol{G}}_t^s \tilde{P}^\top)^{\odot 2} \\
& W_t = W_{t-1} - \eta\, \mathrm{Reshape}\left(\tilde{\Delta}_t^o, [n,m]\right).
\end{aligned}
\tag{5}
$$

OS Performs Better 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 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 $\tilde{\boldsymbol{G}}\tilde{P}\tilde{P}^\top$ is capable of preserving the primary information of $\tilde{\boldsymbol{G}}$. 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.

[2]Appendix A provides a comprehensive explanation of notations.

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 and Stern, 2018] by applying rank-1 decomposition on $(\tilde{\boldsymbol{G}}_t^s\tilde{P}^\top)^{\odot 2}$, 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 $\tilde{\boldsymbol{G}}$ first before the squaring and decomposition. **The final algorithm for VLoRP optimized with Projfactor is outlined in Appendix B**. We showcase the performance comparison between ProjFactor and **OS** in Figure 2(d-f), which aligns with our expectations.

3 Convergence Analysis

Incorporating Projection Granularity into LoRP approaches offers clear benefits, but it is crucial to ensure that various-grained projections do not compromise the convergence of model training. This section provides such theoretical guarantees. The detailed proofs are deferred to Appendix D.

Convergence of VLoRP under SGD 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 $G = \nabla_W \mathcal{L}(W)$, we have

Theorem 1. *Let $\mathcal{L}$ be an L-smooth function with respect to the matrix-shaped parameter W. Assume the parameter up-*

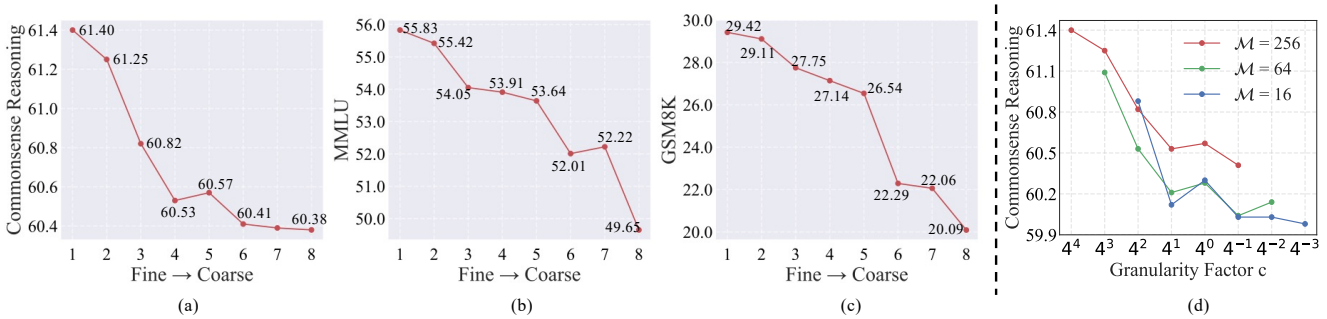

Figure 3: (a-c). Performance comparison of VLoRP configurations under a fixed memory budget $\mathcal{M} = 256$. The y-axis shows accuracy, whereas the x-axis indexes the projection-granularity level. Level "1" denotes the finest grain, $(c = 256, r = 1)$. Each unit increase in level divides c by 4 and multiplies r by 4; (d). Performance Evaluation for Different Projection Granularities under Varying Memory Budgets on Commonsense.

dates 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 1 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.

Convergence of VLoRP under ProjFactor To analyze the convergence of the proposed Adam-like memory-efficient optimization, ProjFactor, 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 are defined as follows:

$$\begin{aligned}
&\frac{d}{dt}\tilde{m}_t^s = a(\tilde{G}_t^s - \tilde{m}_t^s),\ \hat{v}_t^o = \frac{\tilde{v}_{rt}^o \tilde{v}_{ct}^o}{\mathbf{1}_n^T \tilde{v}_{rt}^o},\\
&\frac{d}{dt}\tilde{v}_{rt}^o = b((\tilde{G}_t^o)^{\odot 2}\mathbf{1}_m - \tilde{v}_{rt}^o),\ \frac{d}{dt}\tilde{v}_{ct}^o = b(\mathbf{1}_n^T(\tilde{G}_t^o)^{\odot 2} - \tilde{v}_{ct}^o),\\
&\frac{d}{dt}W_t = \mathrm{Reshape}\left(-\tilde{m}_t^s \tilde{P}^\top \Big/ \sqrt{\hat{v}_t^o},\ [n, m]\right),
\end{aligned} \tag{6}$$

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 2. *Suppose the functions in system equation 6 are continuously differentiable. Under mild assumptions (formally stated in Assumption 1 in Appendix D.2), we have*

(1) For $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)$ satisfying equation 6, we have $\frac{d}{dt}\mathcal{H}(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o) \leq 0$.

(2) Any bounded solution $(W_t, \tilde{m}_t^s, \tilde{v}_{rt}^o, \tilde{v}_{ct}^o)_t$ of equation 6 converges to a stationary point of $\mathcal{L}(W)$ as $t \to \infty$.

Theorem 2 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.

4 Experimental Results

4.1 Fine-Grained Gradient Projection Improves Performance

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

To address this question, it is essential to first establish a fair basis for comparison: for a fixed granularity factor c, increasing the rank typically enhances performance while incurring additional memory costs; Similarly, for a fixed rank r, employing a finer-grained projection (larger c) also increases memory requirements, since the shape of the projected gradient $\tilde{G}^s$ is $[nc, r]$. 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 conducted experiments on three representative benchmarks (Commonsense Reasoning, MMLU, GSM8K) to investigate how different configurations of projection granularity and rank affect finetuning performance under an identical memory budget.

The results are illustrated in Figure 3(a-c), where the y-axis represents performance (higher is better) and the x-axis indicates varying projection granularities defined by $c = 4^{5-i}$ and $r = 4^{i-1}$. Thus, an increase in i corresponds to a smaller c and a larger r, with a smaller c representing coarser projection granularity. Notably, varying i does not change the product of c and r, thereby maintaining a constant overall memory cost. We observed that VLoRP configurations with finer projection granularity (i.e., larger c and smaller r) always yield higher performances. Specifically, the finest-grained VLoRP configuration $(c = 4^4, r = 4^0)$ achieves the highest accuracy

Methods	ARC_C	ARC_E	BoolQ	HellaSwag	OBQA	PIQA	SIQA	winogrande	Avg.
Llama2-7B									
Untuned	42.26 ± 1.45	75.26 ± 0.87	76.86 ± 0.73	56.17 ± 0.49	30.40 ± 2.08	77.91 ± 0.97	45.11 ± 1.13	58.78 ± 1.30	58.84
Adam	47.12 ± 1.46	78.55 ± 0.83	82.27 ± 0.65	56.08 ± 0.49	33.60 ± 2.13	77.45 ± 0.96	52.53 ± 1.13	71.69 ± 1.25	62.41
Adafactor	48.06 ± 1.46	79.22 ± 0.82	80.50 ± 0.68	56.24 ± 0.49	34.20 ± 2.14	77.56 ± 0.96	52.02 ± 1.13	71.22 ± 1.26	62.38
LoRA	44.23 ± 1.46	76.66 ± 0.85	80.21 ± 0.68	55.79 ± 0.49	33.00 ± 2.13	76.57 ± 0.97	47.94 ± 1.13	69.69 ± 1.27	60.51
Galore	44.13 ± 1.45	76.65 ± 0.87	78.23 ± 0.72	57.59 ± 0.49	32.00 ± 2.09	77.95 ± 0.97	46.01 ± 1.13	69.71 ± 1.29	60.28
fira	44.06 ± 1.45	76.63 ± 0.87	78.12 ± 0.73	57.69 ± 0.49	32.40 ± 2.09	77.78 ± 0.97	46.21 ± 1.13	69.89 ± 1.29	60.35
APOLLO	44.28 ± 1.45	76.26 ± 0.87	77.74 ± 0.73	57.00 ± 0.49	31.40 ± 2.08	77.97 ± 0.97	46.11 ± 1.13	69.46 ± 1.29	60.03
VLoRP	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**
Llama3.1-8B									
Untuned	50.86 ± 1.46	81.44 ± 0.80	82.20 ± 0.67	59.93 ± 0.49	33.00 ± 2.10	79.94 ± 0.93	46.84 ± 1.13	73.76 ± 1.24	63.18
Adam	51.96 ± 1.44	80.85 ± 0.84	83.44 ± 0.70	61.36 ± 0.49	34.80 ± 2.10	81.94 ± 0.98	49.32 ± 1.13	76.72 ± 1.25	65.05
Adafactor	51.89 ± 1.45	80.94 ± 0.84	83.12 ± 0.71	61.37 ± 0.49	34.00 ± 2.09	81.03 ± 0.94	49.31 ± 1.13	76.79 ± 1.26	64.91
LoRA	51.54 ± 1.46	81.65 ± 0.85	82.54 ± 0.73	60.21 ± 0.49	33.20 ± 2.07	80.14 ± 0.96	46.72 ± 1.13	73.80 ± 1.29	63.73
Galore	51.62 ± 1.45	81.69 ± 0.86	82.48 ± 0.72	60.40 ± 0.49	33.80 ± 2.11	79.87 ± 0.95	46.98 ± 1.13	74.35 ± 1.28	63.90
fira	51.29 ± 1.45	81.81 ± 0.85	82.01 ± 0.72	60.37 ± 0.49	33.60 ± 2.08	80.76 ± 0.96	47.66 ± 1.12	75.06 ± 1.28	64.07
APOLLO	51.09 ± 1.46	81.29 ± 0.83	82.41 ± 0.62	59.74 ± 0.50	33.60 ± 2.11	80.91 ± 0.97	48.32 ± 1.12	74.82 ± 1.22	64.02
VLoRP	52.65 ± 1.46	82.15 ± 0.87	83.39 ± 0.71	60.00 ± 0.49	34.00 ± 2.12	81.07 ± 0.98	48.57 ± 1.13	76.24 ± 1.28	**64.76**
Qwen3-4B									
Untuned	49.53 ± 1.46	79.78 ± 0.81	84.31 ± 0.62	51.32 ± 0.50	29.80 ± 2.05	74.91 ± 1.01	50.10 ± 1.13	65.92 ± 1.33	60.71
Adam	53.09 ± 1.46	82.29 ± 0.83	85.41 ± 0.62	52.74 ± 0.50	33.60 ± 2.11	74.91 ± 0.97	53.32 ± 1.12	73.82 ± 1.22	63.65
Adafactor	53.10 ± 1.45	82.38 ± 0.77	85.70 ± 0.59	51.67 ± 0.50	34.60 ± 2.14	74.87 ± 0.97	53.55 ± 1.13	73.91 ± 1.25	63.72
LoRA	50.43 ± 1.46	80.26 ± 0.82	83.61 ± 0.65	52.16 ± 0.50	30.60 ± 2.05	75.14 ± 1.01	49.47 ± 1.13	71.23 ± 1.33	61.60
Galore	50.03 ± 1.46	79.94 ± 0.87	83.86 ± 0.66	52.17 ± 0.50	31.00 ± 2.07	75.27 ± 0.97	49.79 ± 1.13	69.38 ± 1.27	61.43
fira	50.49 ± 1.46	80.33 ± 0.76	83.97 ± 0.59	51.53 ± 0.49	30.80 ± 2.10	75.61 ± 0.99	50.41 ± 1.13	69.22 ± 1.30	61.54
APOLLO	50.44 ± 1.46	80.74 ± 0.82	84.09 ± 0.59	51.41 ± 0.49	31.40 ± 2.08	75.66 ± 0.99	50.13 ± 1.13	68.82 ± 1.30	61.58
VLoRP	51.25 ± 1.46	81.77 ± 0.80	85.12 ± 0.63	52.60 ± 0.50	31.60 ± 2.08	75.39 ± 1.01	50.51 ± 1.13	69.07 ± 1.31	**62.15**

Table 1: Performance Comparison on Commonsense Reasoning. All models are first finetuned on the Commonsense170k [Hu *et al.*, 2023] dataset and then evaluated separately on 8 reasoning tasks (0-shot in the prompt). We set the Memory Budget $\mathcal{M} = 256$ for all models of VLoRP. For a fair comparison, we set the rank of all other low-rank-based methods as 256. We used the finest-grained projection for VLoRP as the benefits observed in section 4.1.

Methods	Hum.	STEM	S. Sci.	Other	ALL
Untuned	39.54	34.85	49.14	47.73	42.53
Adafactor	53.01	46.55	66.13	56.87	56.80
Adam	52.63	44.92	65.31	62.45	56.01
LoRA	50.74	43.61	60.93	59.73	53.55
Galore	50.22	42.61	60.25	59.52	52.93
fira	49.85	42.49	60.31	59.55	52.83
APOLLO	49.12	41.42	57.71	56.91	51.13
VLoRP	**52.29**	**46.18**	**64.05**	**62.24**	**55.83**

Table 2: Performance Comparison on the MMLU benchmark with Llama2-7B. We also set the Memory Budget $\mathcal{M} = 256$ for VLoRP and $r = 256$ for other low-rank based methods. The best-performing configurations are highlighted.

scores of 61.40, 55.83, and 29.42 on three benchmarks, respectively, among all tested configurations sharing the same memory budget. These results suggest that **projection granularity may have a more critical role than rank in balancing memory efficiency and performance**. Furthermore, in Figure 3(d), we present the performance of VLoRP across different granularity factors under varying memory budgets. It is evident that finer-grained projections generally outperform coarser ones.

Beyond the performance advantages, finer-grained projection also offers several additional benefits: 1. **Efficient Matrix Multiplication.** 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 $O\left(\frac{nm\mathcal{M}}{c}\right)$; 2. **Faster Generation 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. The training runtimes reported in Table 4 empirically validate these, showing that the configuration with the finer granularity achieves the faster training speed. This improvement is the result of the two benefits described above; 3. **Lower Numerical Error.** As c increases, the numerical stability of the random projection tends to improve. As discussed above, a larger c reduces the number of floating-point arithmetic operations in matrix multiplications, which can help mitigate the accumulation of numerical errors. This

is particularly important in low-precision training scenarios such as float16 or bfloat16, where limited mantissa precision can lead to instability. Furthermore, this numerical stability advantage is further amplified when using gradient accumulation or Adam-based optimizers, as these methods involve iteratively accumulating optimization states. We provide an analytical experiment in Appendix C.3 to support this claim.

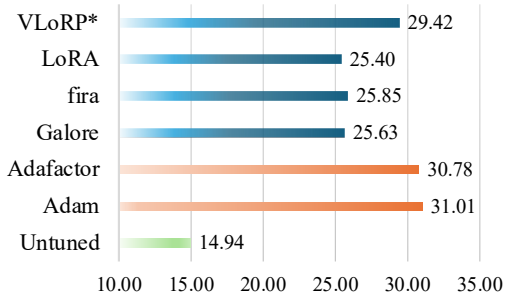

Figure 4: Performance comparison of different methods on GSM8K with Llama2-7B. The rank is uniformly fixed at 256 for all other low-rank-based methods. VLoRP* denotes the configuration $(c = 256, r = 1)$.

4.2 Comprehensive 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 C.

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.*, 2019], ARC-e and 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 consists of 8.5K high-quality math problems.

Baselines We compare VLoRP with 7 state-of-the-art baselines, covering full-parameter finetuning, LoRA, and LoRP methods. Specifically, we compare with Untuned, which represents the basic performance of the pre-trained model without finetuning, Adam [Kingma and Ba, 2014], Adafactor [Shazeer and 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 test Llama2-7B, Llama3-3B, Qwen3-4B models with the bfloat16 data type. We ensure that for each tested method, every token in the training set is encountered at least once. Gradient accumulation and activation checkpointing [Chen *et al.*, 2016] are enabled.

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 4. Generally, every fine-tuning strategy confers substantial performance gains over the untuned model. Among the fine-tuning baselines, full-parameter updates with Adam or Adafactor consistently achieve the strongest results across tasks; nevertheless, the inherent nature of full-parameter fine-tuning precludes memory efficiency. Notably, Adafactor delivers comparable even superior results to Adam, consistent with previous findings [Shazeer and Stern, 2018; Hao *et al.*, 2024]. Moreover, the proposed VLoRP not only outperforms all memory-efficient fine-tuning baselines but in many instances achieves results comparable to full-parameter fine-tuning. Given the difficulty of the benchmarks experimented with, this result is decidedly non-trivial.

Methods	**GPU Mem.**
Adam	67.46 GB
LoRA($r = 256$)	28.80 GB
GaLore($r = 256$)	25.69 GB
Fira($r = 256$)	36.64 GB
APOLLO($r = 256$)	36.20 GB
VLoRP($\mathcal{M} = 256$)	**24.13** GB

Table 3: GPU memory consumption of different finetuning methods on Commonsense Reasoning. Llama2-7B is used, and the data type is bfloat16 (batch size = 16, sequence length = 1024).

Configurations	**Runtime**
$c = 4^{-2}, \quad r = 4^6$	183s/iter
$c = 4^0, \quad r = 4^4$	165s/iter
$c = 4^2, \quad r = 4^2$	161s/iter
$c = 4^4, \quad r = 4^0$	157s/iter

Table 4: Training speed for different configurations. We tested with Llama2-7B model, bfloat16 data type, 512 batch size, and 1024 input length.

Memory Analysis We illustrate the memory usage of various methods in Table 3, emphasizing that VLoRP achieves the most significant memory reduction compared to baseline approaches. Once gradient accumulation is enabled, certain LoRP-based schemes—most notably Fira and APOLLO—require substantially more VRAM than other low-rank approaches. This overhead stems from their reliance on the full-rank gradient during weight update: they compress only the optimiser states, leaving the gradient tensor itself untouched. By contrast, VLoRP intrinsically lowers memory usage by operating exclusively on projected gradients. Specifically, 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.

5 Conclusion

In this work, we introduce VLoRP, a memory-efficient finetuning 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. Convergence analysis and extensive empirical results confirm the effectiveness of our approach, making it a promising solution for practical deployment in memory-constrained settings.

Acknowledgements

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA2386-24-1-4011, and this research is partially supported by the Singapore Ministry of Education Academic Research Fund Tier 1 (Award No: T1 251RES2509).

References

[Baydin *et al.*, 2022] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

[Berahas *et al.*, 2022] Albert S Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, 22(2):507–560, 2022.

[Bisk *et al.*, 2020] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[Chen *et al.*, 2016] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost.(2016). *arXiv preprint arXiv:1604.06174*, 2016.

[Chen *et al.*, 2023] Lizhang Chen, Bo Liu, Kaizhao Liang, and Qiang Liu. Lion secretly solves constrained optimization: As lyapunov predicts. *arXiv preprint arXiv:2310.05898*, 2023.

[Chen *et al.*, 2024a] Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Konstantinos Parasyris, Jiancheng Liu, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. 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*, 2024.

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

[Clark *et al.*, 2018] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[Clark *et al.*, 2019] Christopher Clark, Kenton Lee, Ming Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

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

[Dasgupta and Gupta, 2003] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Struct. Algorithms*, 22(1):60–65, 2003.

[Dean *et al.*, 2012] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[Dettmers *et al.*, 2023] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 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.

[Hao *et al.*, 2024] Yongchang Hao, Yanshuai Cao, and Lili Mou. 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 *et al.*, 2024] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. 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.

[Hendrycks *et al.*, 2020] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[Hu *et al.*, 2022] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

[Hu *et al.*, 2023] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5254–5276. Association for Computational Linguistics, 2023.

[Indyk and Motwani, 1998] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613. ACM, 1998.

[Kingma and Ba, 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Liang *et al.*, 2024] Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. Memory-efficient llm training with online subspace descent. *arXiv preprint arXiv:2408.12857*, 2024.

[Loshchilov and Hutter, 2019] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[Maddison *et al.*, 2018] Chris J Maddison, Daniel Paulin, Yee Whye Teh, Brendan O'Donoghue, and Arnaud Doucet. Hamiltonian descent methods. *arXiv preprint arXiv:1809.05042*, 2018.

[Malladi *et al.*, 2023] Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. *Advances in Neural Information Processing Systems*, 36:53038–53075, 2023.

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

[Mihaylov *et al.*, 2018] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 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*, pages 2381–2391. Association for Computational Linguistics, 2018.

[Nevel'son and Has' minskii, 1976] Mikhail Borisovich Nevel'son and Rafail Zalmanovich Has' minskii. *Stochastic approximation and recursive estimation*, volume 47. American Mathematical Soc., 1976.

[Nguyen *et al.*, 2024] Son Nguyen, Lizhang Chen, Bo Liu, and Qiang Liu. H-fac: Memory-efficient optimization with factorized hamiltonian descent. *arXiv preprint arXiv:2406.09958*, 2024.

[Razdaibiedina *et al.*, 2023] Anastasia Razdaibiedina, Yuning Mao, Madian Khabsa, Mike Lewis, Rui Hou, Jimmy Ba, and Amjad Almahairi. 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*, pages 6740–6757, 2023.

[Ren *et al.*, 2022] Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. *arXiv preprint arXiv:2210.03310*, 2022.

[Robbins and Monro, 1951] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[Sakaguchi *et al.*, 2019] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*, 2019.

[Sap *et al.*, 2019] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[Shazeer and Stern, 2018] Noam Shazeer and Mitchell Stern. 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*, pages 4603–4611, 2018.

[Shen *et al.*, 2024] Qianli Shen, Yezhen Wang, Zhouhao Yang, Xiang Li, Haonan Wang, Yang Zhang, Jonathan Scarlett, Zhanxing Zhu, and Kenji Kawaguchi. Memory-efficient gradient unrolling for large-scale bi-level optimization. *arXiv preprint arXiv:2406.14095*, 2024.

[Silver *et al.*, 2021] David Silver, Anirudh Goyal, Ivo Danihelka, Matteo Hessel, and Hado van Hasselt. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2021.

[Smith *et al.*, 2018] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. 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, 1992] J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[Wang *et al.*, 2024] Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*, 2024.

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

[Zellers *et al.*, 2019] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 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*, pages 4791–4800. Association for Computational Linguistics, 2019.

[Zhao *et al.*, 2024] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 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 *et al.*, 2024] Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z. Pan, Zhangyang Wang, and Jinwon Lee. Apollo: Sgd-like memory, adamw-level performance. *arXiv preprint arXiv:2412.05270*, 2024.