COAP: Memory-Efficient Training with Correlation-Aware Gradient Projection

Jinqi Xiao[1,2,*], Shen Sang[1], Tiancheng Zhi[1], Jing Liu[1],
Qing Yan[1], Linjie Luo[1], Bo Yuan[2]

[1] ByteDance Inc., [2] Rutgers University

Abstract

*Training large-scale neural networks in vision, and multimodal domains demands substantial memory resources, primarily due to the storage of optimizer states. While LoRA, a popular parameter-efficient method, reduces memory usage, it often suffers from suboptimal performance due to the constraints of low-rank updates. Low-rank gradient projection methods (e.g., GaLore, Flora) reduce optimizer memory by projecting gradients and moment estimates into low-rank spaces via singular value decomposition or random projection. However, they fail to account for inter-projection correlation, causing performance degradation, and their projection strategies often incur high computational costs. In this paper, we present **COAP** (COrrelation-Aware Gradient Projection), a memory-efficient method that minimizes computational overhead while maintaining training performance. Evaluated across various vision, language, and multimodal tasks, COAP outperforms existing methods in both training speed and model performance. For LLaMA-1B, it reduces optimizer memory by 61% with only 2% additional time cost, achieving the same PPL as AdamW. With 8-bit quantization, COAP cuts optimizer memory by 81% and achieves 4x speedup over GaLore for LLaVA-v1.5-7B fine-tuning, while delivering higher accuracy.* https://byteaigc.github.io/coap/

1. Introduction

Deep neural networks have achieved remarkable success across vision [10, 22, 35, 36, 41], language [1, 12, 46, 47], and multi-modality domains [11, 25, 26, 44], driven by the increasing scale of these models. While scaling up model size significantly contributes to these advancements [18], it also leads to considerable memory constraints, particularly due to the optimizer states. For instance, training an LLaVA-7B [25] model with the Adam [19] optimizer at BF16 numerical format requires at least 63.8GB of GPU memory, with optimizer states consuming 40% of the total, while the model and gradients each take up 20%. Combining engineering and system efforts, *e.g.*, activation checkpointing [4] and LOMO [32] can significantly reduce the memory usage of activations and gradients. This makes optimizer a critical bottleneck that needs to be addressed. To tackle this issue, low-rank training has proven effective in minimizing the memory footprint of optimizer states by performing training within a low-rank subspace [13, 16, 58].

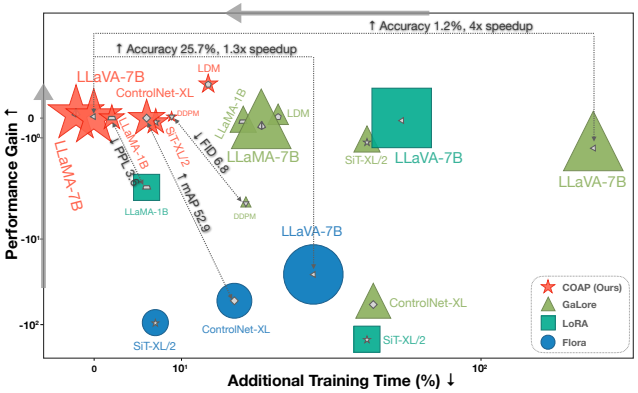


Figure 1. Comparison between COAP and other low-rank-based methods. The X-axis shows additional training time, with lower values being better. The Y-axis shows performance (*e.g.*, FID, PPL) changes compared to the original optimizer (*e.g.*, Adam [19], Adafactor [42]), with higher values indicating better performance.

Current low-rank training approaches can generally be categorized into two main types: parameter-efficient fine-tuning (PEFT) and training with low-rank gradients. The most common method in PEFT is Low-Rank Adaptation (LoRA) [16], which adds trainable low-rank matrices to a frozen pre-trained model's weights, allowing for efficient model adaptation with significantly fewer parameters than full fine-tuning. While low-rank updates can reduce memory consumption, extensive empirical evidence [2, 9, 28, 49, 50, 57] shows that the low-rank constraint diminishes the model's representation capacity, resulting in sub-optimal performance compared to full fine-tuning. In addition, its pre-training alternative ReLoRA [24], which periodi-

*Work done during internship at ByteDance.

Low-rank gradients 方法利用了 gradients 的 low-rank 特性，
而非限制 model param 到 low-rank space

cally updates the model using previously learned low-rank adapters, still requires a full-rank warm-up phase, thereby negating its memory-saving benefits. For training with low-rank gradient [13, 58], these methods leverage the inherent low-rank structure of the gradients rather than constraining model parameters to a low-rank space. Projecting gradients into low-rank space can significantly reduce memory usage during optimization, making it feasible for scenarios with limited memory resources or extremely large models. For example, GaLore [58] periodically applies Singular Value Decomposition (SVD) to decompose gradients, creating a fixed low-rank projection matrix for subsequent steps. However, the high computational cost of SVD can severely slow down training, especially for models with large weight matrices. In contrast, Flora [13] resamples a random projection matrix at each iteration to address the low-rank limitation of LoRA. Despite this, generating new random projection matrices at each iteration still incurs computational overhead, particularly for large models. Besides the significant time costs required to achieve memory savings, these methods also face the challenge of lacking inter-projection correlation. When the projection is determined solely by a single step's gradient or random sampling, it may disrupt the continuity of the optimization trajectory, leading to abrupt shifts and adversely affecting training dynamics.

To address these challenges, we propose a memory-efficient training method, **COAP** (COrrelation-Aware Gradient Projection), that minimizes additional computational overhead while retaining model performance (see Fig. 1). 1) Unlike methods that rely solely on a single step's gradient or random generation to determine the low-rank projection matrix, COAP introduces an *inter-projection correlation-aware update strategy*, by solving an optimization problem with gradient descent. It tracks the previous low-rank projection matrix and combines the direction of the first-order moments to continuously update the low-rank projection matrix, thereby avoiding abrupt shifts in training dynamics. 2) Additionally, we propose to use *occasional low-cost SVD* to recalibrate the low-rank projection matrices, preventing the gradient descent-based updates from getting trapped in local optima. Through the implementation of the criterion that prioritizes gradient reconstruction capability and training direction smoothness, combined with occasional low-cost SVD to continuously update the low-rank projection matrix, we achieve a significant reduction in the computational cost of vanilla SVD in large-scale models while simultaneously enhancing model performance.

We demonstrate the effectiveness of our method in both pre-training and fine-tuning across various vision, language, and multi-modal tasks. For pre-training, our method matches AdamW's performance while reducing optimizer memory by 61% for LLaMA-1B [46], with only 2% increase in time overhead. In 8-bit optimizers, our approach outperforms 8-bit Adam and GaLore in both training speed and performance on LLaMA-7B. For fine-tuning LLaVA-v1.5-7B [25], our method with 8-bit optimization reduces optimizer memory by 81%, achieves 4× speedup over GaLore, and increases accuracy by 1.2%.

2. Related Works

Momentum-based Optimizers, for instance Adam [19], AdamW [30] and Adadelta [55], have largely replaced traditional SGD due to their adaptiveness and faster convergence. However, this adaptiveness increases memory usage, as they require storing first and second moment estimates, effectively tripling the memory footprint. Adafactor [42] addresses this by employing low-rank approximations for the second moments, reducing memory requirements while retaining adaptive learning. In this work, we explore projecting both gradients and moments into a low-rank space during training, enabling efficient large-scale model training with commonly-used momentum-based optimizers.

Low-rank Adaptation (LoRA) [16] enables parameter-efficient fine-tuning by applying low-rank updates to pre-trained models, reducing memory and computation costs. DoRA [28] improves this by decoupling weights into magnitude and direction, using low-rank updates only for directional adjustments. ReLoRA [24] adapts LoRA to pre-training but requires a full-rank warm-up, increasing memory usage with model size. Variants such as LyCORIS [53], AdaLoRA [57], HALOC [52] and COMCAT [51] step further in efficiency and flexibility. However, these methods often result in suboptimal performance [50] due to the low-rank limit. Conversely, our approach preserves full-rank learning while leveraging low-rank gradient structures, supporting both pre-training and fine-tuning without compromising performance.

Training with Low-rank Gradient, inspired by projected gradient descent methods [3, 5], aims to reduce the memory usage of optimizers during training by applying low-rank gradient projections. GaLore [58] computes projection matrices via per-layer gradient decomposition using inefficient SVD, In contrast, Flora [13] proposes to generate projection matrices randomly on the fly to eliminate this overhead. However, a major limitation of these methods is the lack of inter-projection correlation. As the optimizer transitions to a new subspace, the projection matrix is either based on the current gradient or random selection, leading to abrupt shifts. In this work, we address this issue by incorporating prior optimization directions, ensuring smoother transitions in the projection space. This approach improves training dynamics and enhances overall optimization performance.

Memory-efficient Training Techniques are widely employed to reduce the overall memory footprint in large-scale model training, including activation checkpointing [4], 8-bit optimizers [8], and mixed-precision training [34], etc.

The popular approach, ZeRO [38], optimizes memory usage in multi-GPU settings. ZeRO contains three stages, progressively sharding optimizer states, gradients, and model weights across devices, synchronizing only when necessary to minimize memory usage. Our method specifically targets memory reduction for optimizer states and can be effectively combined with these techniques, offering additional memory savings and further improving training efficiency.

3. Method

3.1. Background

Notation. We use the following notation conventions: matrices and vectors are indicated with boldface capital and lowercase letters, such as $\boldsymbol{X}$ and $\boldsymbol{x}$, respectively. Non-boldface letters with indices, *e.g.*, $X(i,j)$, and $x(i)$, denote the entries of a matrix $\boldsymbol{X}$, and a vector $\boldsymbol{x}$, respectively.

Full-Rank Training. For a time-varying weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ with objective function as $\mathcal{L}(\boldsymbol{W})$, the corresponding gradient matrix at time step t can be denoted as $\boldsymbol{G}_t = \nabla_{\boldsymbol{W}}\mathcal{L}_t(\boldsymbol{W}_t) \in \mathbb{R}^{m\times n}$. Then, the general weight update process can be formulated as:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta\rho_t(\boldsymbol{G}_t), \tag{1}$$

where η is the learning rate, and ρ_t is the optimizer-dependent gradient regularizer that adaptively adjusts raw gradients $\boldsymbol{G}_t$ via incorporating additional optimizer states. For instance, Adam [19] uses bias-corrected first and second moment $\boldsymbol{M}_t$ and $\boldsymbol{V}_t$ to regularize the gradients as follows:

$$\begin{aligned}
\boldsymbol{M}_t &= \beta_1\boldsymbol{M}_{t-1} + (1-\beta_1)\boldsymbol{G}_t,\\
\boldsymbol{V}_t &= \beta_2\boldsymbol{V}_{t-1} + (1-\beta_2)\boldsymbol{G}_t^2,\\
\rho_t(\boldsymbol{G}_t) &= \frac{\boldsymbol{M}_t/(1-\beta_1^t)}{\sqrt{\boldsymbol{V}_t/(1-\beta_2^t)}+\epsilon},
\end{aligned} \tag{2}$$

where β_1 and β_2 are hyper-parameters, ϵ is a small constant to ensure numerical stability, and the matrix operations are element-wise. Considering Adam requires extra $2mn$ memory to store $\boldsymbol{M}_t$ and $\boldsymbol{V}_t$, Adafactor [42] proposes to use factorization to estimate $\boldsymbol{V}_t$ for higher memory efficiency:

$$\begin{aligned}
\boldsymbol{R}_t &= \beta_2\boldsymbol{R}_{t-1} + (1-\beta_2)\cdot \mathrm{Sum}(\boldsymbol{G}_t^2, 1),\\
\boldsymbol{C}_t &= \beta_2\boldsymbol{C}_{t-1} + (1-\beta_2)\cdot \mathrm{Sum}(\boldsymbol{G}_t^2, 0),\\
\hat{\boldsymbol{V}}_t &= (\boldsymbol{R}_t\boldsymbol{C}_t)/\mathrm{Sum}(\boldsymbol{R}_t, 0),
\end{aligned} \tag{3}$$

where $\mathrm{Sum}(\cdot)$ returns the sum of each row of the input matrix in the given dimension (0 or 1). Here by using $\boldsymbol{R}_t \in \mathbb{R}^{m\times 1}$ and $\boldsymbol{C}_t \in \mathbb{R}^{1\times n}$ to estimate $\boldsymbol{V}_t$ as $\hat{\boldsymbol{V}}_t \in \mathbb{R}^{m\times n}$, the memory consumption for storing second moments is reduced from mn to $m+n$.

Training with Low-rank Weight Update. Due to the high cost of updating the entire full-rank $\boldsymbol{W}_t$, LoRA [16] and its variants [24, 57], as the memory-efficient solutions that only adjust the change of $\boldsymbol{W}_t$ in the low-rank subspace, are popularly used in practice as follows:

$$\boldsymbol{W}_t = \boldsymbol{W}_0 + \boldsymbol{B}_t\boldsymbol{A}_t, \tag{4}$$

where $\boldsymbol{W}_0 \in \mathbb{R}^{m\times n}$ is the fixed pre-trained or initialized weights, and $\boldsymbol{A}_t \in \mathbb{R}^{r\times n}$ and $\boldsymbol{B}_t \in \mathbb{R}^{m\times r}$ are the learnable low-rank adapters with $r \ll \min(m,n)$.

Training with Low-rank Gradient. Motivated by the observations that 1) updating weight in the low-rank subspace may limit the representation capacity, and 2) gradient matrix tends to exhibit a certain degree of low-rankness during training, recent works [13, 58] propose to explore memory-efficient training via projecting gradient into low-rank format. For instance, given a projection matrix $\boldsymbol{P}_t$ that defines a low-rank subspace, the *projected* gradient, first order moments and second order moments in Adam optimizer are calculated as $\boldsymbol{G}_t^{\mathrm{proj}} = \boldsymbol{G}_t\boldsymbol{P}_t$, $\boldsymbol{M}_t^{\mathrm{proj}} = \beta_1\boldsymbol{M}_{t-1}^{\mathrm{proj}} + (1-\beta_1)\boldsymbol{G}_t^{\mathrm{proj}}$ and $\boldsymbol{V}_t^{\mathrm{proj}} = \beta_2\boldsymbol{V}_{t-1}^{\mathrm{proj}} + (1-\beta_2)(\boldsymbol{G}_t^{\mathrm{proj}})^2$, respectively. Then, the weight update is performed as follows:

$$\begin{aligned}
\boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - \eta\rho_t(\boldsymbol{G}_t^{\mathrm{proj}}),\\
\rho_t(\boldsymbol{G}_t^{\mathrm{proj}}) &= \frac{\boldsymbol{M}_t^{\mathrm{proj}}/(1-\beta_1^t)}{\sqrt{\boldsymbol{V}_t^{\mathrm{proj}}/(1-\beta_2^t)}+\epsilon}\boldsymbol{P}_t^\top.
\end{aligned} \tag{5}$$

Here, $\boldsymbol{W}_t$, $\boldsymbol{G}_t$, $\rho_t(\boldsymbol{G}_t^{\mathrm{proj}}) \in \mathbb{R}^{m\times n}$ $(m \geq n)$, $\boldsymbol{P}_t \in \mathbb{R}^{n\times r}$, and $\boldsymbol{G}_t^{\mathrm{proj}}$, $\boldsymbol{M}_t^{\mathrm{proj}}$, $\boldsymbol{V}_t^{\mathrm{proj}} \in \mathbb{R}^{m\times r}$.

3.2. Challenge Analysis of Gradient Projection

As discussed in Section 3.1 and demonstrated in [13, 58], projecting gradients onto low-rank subspace enables efficient memory-saving training while maintaining the essential full-rank structure of model weights, which is crucial for preserving the model's expressive capacity and avoiding information loss. However, despite these successes, we argue that state-of-the-art gradient projection methods still face some fundamental challenges as follows.

1) Lack of Inter-Projection Correlation Awareness. As highlighted in [13, 58], weight updates should not be restricted to a single low-rank subspace; therefore, periodically switching the projection subspace (i.e., recalculating $\boldsymbol{P}_t$) is essential in the training process. Specifically, GaLore [58] performs SVD on $\boldsymbol{G}_t$ every set period of batches, selecting either the truncated left or right singular vector matrix as $\boldsymbol{P}_t$. In contrast, Flora [13] proposes to directly use random matrix as $\boldsymbol{P}_t$ for gradient projection at each batch.

However, a fundamental limitation for these existing $\boldsymbol{P}_t$ update strategies is the lack of inter-projection correlation. Specifically, each time when the optimizer switches to a new low-rank subspace, the updated projection matrix $\boldsymbol{P}_t$ is determined either solely by $\boldsymbol{G}_t$ (as in GaLore) or through

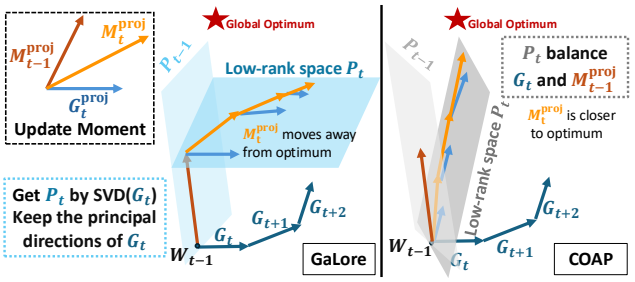


Figure 2. Comparison of optimization trajectories between COAP and GaLore. $\boldsymbol{P}_{t-1}$ represents the low-rank projection space from the previous cycle, and $\boldsymbol{P}_t$ represents the current low-rank projection space. The symbol "$\star$" denotes the global optimum. GaLore updates $\boldsymbol{P}_t$ based on a batch of stochastic data, which can lead to suboptimal updates if the data significantly deviates from the overall data distribution. In contrast, our method uses the more stable first-order moment $\boldsymbol{M}_{t-1}$ as guidance, mitigating this issue.

randomly selection (as in Flora), without leveraging directional information from previous projections. For instance, the first-order moment $\boldsymbol{M}_{t-1}^{proj}$, which reflects the moving average of projected gradients, is not incorporated into the $\boldsymbol{P}_t$ update. From an optimization trajectory perspective, this lack of continuity leads to abrupt shifts in the projection space without accounting for prior optimization direction (see Fig. 2). Such discontinuities, if not properly addressed, can destabilize the training dynamics, potentially compromising model performance.

To validate our analysis on the importance of inter-projection correlation, we perform an empirical experiment by pre-training DeiT-Base [45] model on the CIFAR-100 [20] dataset. Fig. 3 shows the *cumulative effective update* (CEU) as training progresses. Here CEU, defined as $\sum \|W_t - W_{t-1}\|_1 = \eta \sum \left\|\rho_t(\boldsymbol{G}_t^{\text{proj}})\right\|_1$, serves as a metric to assess optimization sufficiency, with a value close to that of the original optimizer indicating minimal loss due to the low-rank projection. From Fig. 3, it is seen that the inter-projection correlation-unaware GaLore and Flora yield a CEU that significantly deviates from that of the original optimizer, resulting in a notable drop in test accuracy. Particularly, because Flora selects $\boldsymbol{P}_t$ in a purely random way, it produces a CEU that is very different from Adam's, leading to severe performance degradation. On the other hand, our proposed inter-projection-aware approach achieves a higher CEU than the existing solutions (sometimes even surpassing that of Adam), bringing higher test accuracy. It suggests that the projection directions may have effectively eliminated noise or irrelevant information. This could potentially enable the model to converge more efficiently towards the optimal solution. Experiments in Section 4 further demonstrate that our approach empirically outperforms the state-of-the-art methods across various pre-training and fine-tuning tasks for different model types.

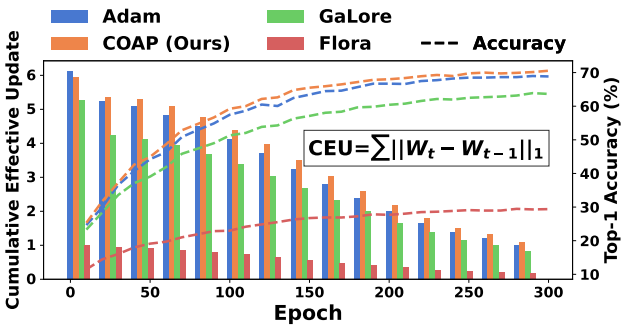


Figure 3. Cumulative effective update (CEU) and Top-1 accuracy over 300 epochs for different optimization methods on the CIFAR-100 dataset using the DeiT-Base model trained from scratch. The rank of $\boldsymbol{P}_t$ is 192, the learning rate is 5×10^{-5}, and all models are trained on a single A100 GPU with a batch size of 256. The magnitude of CEU indicates the extent to which the optimizer influences the model.

2) High SVD-incurred Projection Cost. Another challenge of current projection gradient methods, especially GaLore, is the high computational overhead caused by the costly SVD, which has a complexity of $O(mn^2)$ for an $m \times n$ matrix. Specifically, since each $\boldsymbol{P}_t$ update requires performing SVD on $\boldsymbol{G}_t$, this process can become prohibitively expensive, especially for large gradient matrices. For instance, when using GaLore to pre-train LLaVA-7B [25] model under a typical configuration – updating $\boldsymbol{P}_t$ every 200 batches with a batch size of 16 – it takes approximately 540 seconds on a single A100 GPU to compute all $\boldsymbol{P}_t$ with a rank of 512 for each projection update. On the other hand, the total time required for weight updates over these 200 batches is only 600 seconds. This means that calculating the $\boldsymbol{P}_t$ in GaLore incurs a 90% training overhead, making it extremely computationally intensive.

3.3. Proposed Method

To overcome these challenges and make the low-rank gradient projection solution more practical and feasible for model training, we propose COAP, which is detailed below.

Inter-projection Correlation-aware $\boldsymbol{P}_t$ Update. Considering the importance of incorporating the information from the previous projection into the current update, we propose calculating $\boldsymbol{P}_t$ via solving the optimization problem:

$$\min_{\boldsymbol{P}_t} \underbrace{\text{MSE}(\hat{\boldsymbol{G}}_t, \boldsymbol{G}_t)}_{\text{reconstruction term}} \cdot \underbrace{(1 - \text{CosSim}(\hat{\boldsymbol{M}}_{t-1}, \boldsymbol{G}_t))}_{\text{direction term}}, \quad (6)$$

While Flora avoids the intensive SVD, it still incurs significant overhead in large-scale model training due to regenerating random projection matrices each iteration.

GaLore claims that the total computational overhead induced by SVD is negligible ($< 10\%$) compared to other memory-efficient training techniques such as CPU-offload [39]. However, this overhead cannot be ignored when compared to methods that do not use these techniques

where $\mathrm{CosSim}(\cdot,\cdot)$ and $\mathrm{MSE}(\cdot,\cdot)$ return the cosine similarity and mean squared error, respectively. $\hat{\boldsymbol{G}}_t \in \mathbb{R}^{m\times n} = \boldsymbol{G}_t\boldsymbol{P}_{t-1}\boldsymbol{P}_{t-1}^\top$ and $\hat{\boldsymbol{M}}_{t-1} \in \mathbb{R}^{m\times n} = \boldsymbol{M}_{t-1}^{\text{proj}}\boldsymbol{P}_{t-1}^\top$ are the full-rank estimates of gradient and first-order moment projected back from the low-rank subspace, respectively. Notably, the *reconstruction term* is introduced to minimize the reconstruction error for gradients incurred by projection – achieving the similar goal that SVD essentially aims for, and *direction term* encourages the consistency of optimization direction after restoring from low-rank subspace. To solve Eqn. 6 as a non-convex optimization problem, we use stochastic gradient descent (SGD) to iteratively update $\boldsymbol{P}_t$ (see details in Appendix).

Occasional Low-cost SVD to recalibrate $\boldsymbol{P}_t$ Update. In principle, unlike GaLore, our proposed $\boldsymbol{P}_t$ update only requires to perform SGD-based iterative update without performing SVD on $\boldsymbol{G}_t$. However, considering 1) SGD may get stuck in local optimum in the optimization process; and 2) theoretically truncated SVD provides minimal approximation error for low-rank decomposition, we propose performing infrequent low-cost SVD to redirect the calculation of $\boldsymbol{P}_t$ as follows:

$$\begin{aligned}\boldsymbol{Q}_{\text{red}}, \_ &= \mathrm{QR}_{\text{red}}(\boldsymbol{G}_t\boldsymbol{P}_{t-1}),\\ \boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{Z}^\top &= \mathrm{SVD}(\boldsymbol{Q}_{\text{red}}^\top\boldsymbol{G}_t),\\ \boldsymbol{P}_t &= \boldsymbol{Z}\end{aligned} \tag{7}$$

where $\boldsymbol{U}, \boldsymbol{\Sigma} \in \mathbb{R}^{r\times r}, \boldsymbol{Z} \in \mathbb{R}^{n\times r}$ and $\mathrm{QR}_{\text{red}}(\cdot)$ is the reduced QR decomposition, which returns $\boldsymbol{Q}_{\text{red}} \in \mathbb{R}^{m\times r}$ consisting of orthogonal columns. Our key idea is to first project $\boldsymbol{G}_t$ into the $\boldsymbol{P}_{t-1}$-defined low-rank subspace and obtain $\boldsymbol{Q}_{\text{red}}$. Then, because all the columns of $\boldsymbol{Q}_{\text{red}}$ are orthogonal, we have $\boldsymbol{G}_t \approx \boldsymbol{Q}_{\text{red}}\boldsymbol{Q}_{\text{red}}^\top\boldsymbol{G}_t \approx \boldsymbol{Q}_{\text{red}}\boldsymbol{U}\Sigma\boldsymbol{Z}^\top$ to determine $\boldsymbol{P}_t$ as $\boldsymbol{Z}$. Notably, here the SVD on $\boldsymbol{Q}_{\text{red}}^\top\boldsymbol{G}_t$ can be viewed as the approximated version for SVD on $\boldsymbol{G}_t$, but the much smaller size of $\boldsymbol{Q}_{\text{red}}^\top$ brings significant reduction in computational complexity from $\mathcal{O}(mn^2)$ to $\mathcal{O}(mr^2)$. Evaluation on single A100 GPU shows that this low-cost method only consumes 23 seconds when updating all $\boldsymbol{P}_t$ for LLaVA-7B model, bringing over 20× speedup than the SVD operation in GaLore.

Overall Training Procedure. The proposed gradient projection update schemes, due to their generality, can be seamlessly integrated into existing optimizers such as Adam and Adafactor. Algorithm 1 outlines the overall Adam-based training procedure incorporating the proposed $\boldsymbol{P}_t$ update approach. In this setup, $\boldsymbol{P}_t$ is updated at regular intervals every T_{u} steps following Eqn. 6, with additional recalibrating based on Eqn. 7 every $\lambda \times T_{\text{u}}$ steps. Therefore, the weight update process can be formulated as:

$$\boldsymbol{W}_t = \boldsymbol{W}_0 + \sum_{i=0}^{\lambda-1}\sum_{j=1}^{T_{\text{u}}} \Delta\boldsymbol{W}_{i\times T_{\text{u}}+j}^{\text{proj}}\boldsymbol{P}_i^\top, \tag{8}$$

where $t = \lambda \times T_{\text{u}}$ and $\Delta\boldsymbol{W}_i^{\text{proj}}$ represent the weight update within the low-rank subspace defined by $\boldsymbol{P}_i$, with gradients over each T_{u} steps sharing the same $\boldsymbol{P}_i$. Notably, since T_{u} and λ are hyper-parameters that control the update interval of $\boldsymbol{P}_t$, the corresponding ablation study on their effects is discussed in Section 5.

Algorithm 1: Adam with COAP

Input: Weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, Learning rate η, Rank r, Betas $[\beta_1, \beta_2]$, Update interval $[\lambda, T_{\text{u}}]$.
Initialize: $\boldsymbol{M}_0^{\text{proj}} \in \mathbb{R}^{m\times r} \leftarrow 0$, $\boldsymbol{V}_0^{\text{proj}} \in \mathbb{R}^{m\times r} \leftarrow 0$, $t \leftarrow 0$
Randomly Initialize: $\boldsymbol{P}_0 \in \mathbb{R}^{n\times r}$
Compute: $\boldsymbol{P}_0 \leftarrow (\boldsymbol{P}_0, \boldsymbol{G}_0)$ ▷ Eqn. 7
for t in $[1, 2, \cdots]$ **do**
  Compute: gradient $\boldsymbol{G}_t$ of $\boldsymbol{W}_t$ in the loss function.
  if $t \mod T_{\text{u}} = 0$ **then**
    if $t \mod (\lambda \times T_{\text{u}}) = 0$ **then**
      Compute: $\boldsymbol{P}_t \leftarrow (\boldsymbol{P}_{t-1}, \boldsymbol{G}_t)$ ▷ Eqn. 7
    else
      Update: $\boldsymbol{P}_t \leftarrow (\boldsymbol{P}_{t-1}, \boldsymbol{G}_t, \boldsymbol{M}_{t-1})$ ▷ Eqn. 6
  else
    $\boldsymbol{P}_t \leftarrow \boldsymbol{P}_{t-1}$
  ▷Project gradient and moments into low-rank space.
  $\boldsymbol{G}_t^{\text{proj}} \leftarrow \boldsymbol{G}_t\boldsymbol{P}_t$
  $\boldsymbol{M}_t^{\text{proj}} \leftarrow \beta_1\boldsymbol{M}_{t-1}^{\text{proj}} + (1-\beta_1)\boldsymbol{G}_t^{\text{proj}}$
  $\boldsymbol{V}_t^{\text{proj}} \leftarrow \beta_2\boldsymbol{V}_{t-1}^{\text{proj}} + (1-\beta_2)(\boldsymbol{G}_t^{\text{proj}})^2$
  ▷Calculate the bias correction term in low-rank space.
  $\Delta\boldsymbol{W}_t^{\text{proj}} \leftarrow \frac{\boldsymbol{M}_t^{\text{proj}}/(1-\beta_1^t)}{\sqrt{\boldsymbol{V}_t^{\text{proj}}/(1-\beta_2^t)}+\epsilon}$
  ▷Restore $\Delta\boldsymbol{W}_t^{\text{proj}}$ to original space and update $\boldsymbol{W}$.
  $\boldsymbol{W}_t \leftarrow \boldsymbol{W}_{t-1} - \eta\Delta\boldsymbol{W}_t^{\text{proj}}\boldsymbol{P}_t^\top$
Return: updated $\boldsymbol{W}$

4. Experiments

We evaluate COAP over various scales of datasets and models, including both pre-training and fine-tuning. The models assessed encompass architectures that contain convolutional layers, *e.g.*, Latent Diffusion Models (LDM) [40], ControlNet-XL [36, 56], as well as architectures that based only on transformers, including Scalable Interpolant Transformers (SiT) [33], LLaVA [25, 26] and LLaMA [46].

Rank Ratio Given the varying sizes of weight matrices in models, *e.g.*, LDM, and SDXL, we define the rank ratio as c to unify the calculation of ranks for different matrices. For an $m \times n$ matrix, the rank r is given by $r = \frac{\min(m,n)}{c}$.

In our experiments, we use various measures, and "↑" indicates that a higher value is better for this metric, while "↓" indicates that a lower value is better for this metric.

4.1. Pre-training LDM

Experimental Settings. We follow the configuration of LDM and train the U-Net model of LDM from scratch us-

https://github.com/CompVis/latent-diffusion

Table 1. Pre-training LDM on the ImageNet-1K dataset for 690K steps using 8×V100 GPUs. FID is reported, along with training time and GPU memory usage of optimizer states in FP32 format.

Method	Rank Ratio	Optimizer Mem. (GB)↓	Training Time↓	FID↓
AdamW	-	3.0	310.3 h	18.0
GaLore	2	2.0 (-33%)	+21%	17.8
COAP	2	**1.8 (-40%)**	**+13%**	**16.2**
Adafactor	-	2.2	336.9 h	38.7
GaLore	2	1.8 (-18%)	+18%	23.3
COAP	2.2	**1.3 (-41%)**	**+7%**	**18.3**

ing different optimizers on 8×V100 GPUs with a batch size of 32. We generate 50 images for each of the 1000 classes with 250 DDIM steps and compute the Fréchet Inception Distance (FID) [14] between the generated images and the validation dataset of ImageNet-1K [7].

Comparison Results. As shown in Table 1, our method reduces optimizer memory by 40%, and decreases the FID by 1.8 and 20.4 compared to the baseline. When integrated into AdamW, our method reduces the FID by 1.6 compared to GaLore. In Adafactor, our method requires only 70% of the memory that GaLore needs, reduces the extra training time by 60%, and decreases the FID score by 5.0.

4.2. Pre-training SiT-XL/2

Experimental Settings. We perform Sit-XL/2 [33] training with Representation Alignment (REPA) [54] on the ImageNet-1K [7] dataset, preprocessing each image to a resolution of 256×256 pixels. Both the training and evaluation settings follow those outlined in REPA. All models are trained for 400K iterations with CFG (classifier-free guidance). For FID computation, we generate 50K samples using the Euler-Maruyama sampler, with the number of inference steps set to 250.

Comparison Results. As shown in Table 2, our method outperforms other low-rank methods, achieving comparable performance to full-rank training while reducing optimizer memory by more than 40%. In contrast, using LoRA for pre-training reduces optimizer memory by 29%, but the adapter increases model size by 48%, leading to a 33% increase in training time and a significant FID increase of 149.6. Flora, which uses random projection, also incurs additional training time to generate low-rank mapping matrices per iteration, increasing FID by 113.3. Compared to GaLore, our method in AdamW achieves a lower FID with 58% less extra training time. In Adafactor, our FID is 0.9 lower than GaLore's, with 78% less extra training time.

4.3. Training ControlNet with Stable Diffusion XL

Experimental Settings. We train a ControlNet [56] based on Stable Diffusion XL [36] for text-to-image generation using human poses as additional control images. Single-person images are collected as training data, with RTM-Pose [17] used for pose estimation and BLIP2 [23] for generating text descriptions and prompts. The estimated poses serve as ground truth, and 1000 samples are randomly selected for evaluation. Specifically, we use human pose and

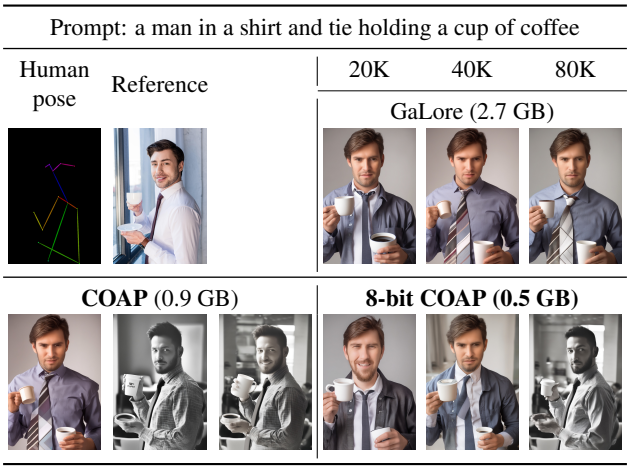
Table 2. Pre-training SiT-XL/2 with REPA on the ImageNet-1K dataset for 400K steps using 8×H100 GPUs. We set the rank as 512 for all methods. FID scores are reported, along with training time and GPU memory usage of optimizer states in FP32 format.

Method	Optimizer Mem. (GB)↓	Model Mem. (GB)↓	Training Time↓	FID↓
AdamW	5.1	2.5	20.8 h	1.9
GaLore	2.6 (-49%)	2.5	+38%	2.3
LoRA	3.6 (-29%)	3.7 (+48%)	+33%	151.9
ReLoRA	3.6 (-29%)	3.7 (+48%)	+33%	151.8
COAP	**2.6 (-49%)**	2.5	**+14%**	**2.2**
Adafactor	2.5	2.5	25.5 h	1.9
GaLore	1.5 (-40%)	2.5	+33%	3.0
Flora	1.6 (-36%)	2.5	+7%	115.2
COAP	**1.5 (-40%)**	2.5	**+7%**	**2.1**

Table 3. Training ControlNet based on SDXL for 80K steps using 8×H100 GPUs in BF16 format, conditioned on human poses.

Method	Rank Ratio	Optimizer Mem. (GB)↓	mAP↑ @ training steps 20K	40K	80K	Converged	Training Time (80K)↓
AdamW	-	9.3	18.3	19.1	19.9	✗	19.3 h
Adafactor	-	5.1	19.2	70.0	72.7	✓	22.3 h
Flora	2	3.9 (-24%)	18.0	18.9	19.6	✗	+16%
GaLore	2	4.7 (-8%)	18.6	67.0	72.7	✓	+39%
GaLore-8bit	2	3.1 (-39%)	19.6	20.8	20.9	✗	+49%
COAP	2	3.6 (-29%)	**66.6**	**71.6**	**73.4**	✓	**+4%**
8-bit COAP	2	**1.9 (-63%)**	18.7	66.9	72.2	✓	+17%
GaLore	4	3.5 (-31%)	18.9	19.7	19.5	✗	+39%
8-bit GaLore	4	3.1 (-39%)	18.8	19.7	19.8	✗	+50%
COAP	4	1.8 (-65%)	**50.9**	**70.4**	**72.1**	✓	**+5%**
8-bit COAP	4	**1.0 (-80%)**	19.4	19.2	71.5	✓	+15%
GaLore	8	2.7 (-47%)	18.6	18.2	19.7	✗	+35%
8-bit GaLore	8	2.3 (-55%)	18.6	18.2	19.7	✗	+45%
COAP	8	0.9 (-82%)	**25.8**	**70.2**	**72.6**	✓	**+6%**
8-bit COAP	8	**0.5 (-90%)**	19.3	18.9	69.9	✓	+13%

Table 4. Comparison of generated images at different training steps (20K, 40K, 80K). We use the DDIM with a guidance scale of 5.0 for generating images. The number of inference steps is 50.



BLIP2-generated prompts as inputs, perform pose estimation on the generated images using RTMPose, and calculate the mean Average Precision (mAP) to evaluate the match between the generated poses and the original poses. The mAP is computed by setting Intersection over Union (IoU) thresholds from 0.5 to 0.95 in increments of 0.05, calculating the IoU between predicted and ground truth key points to determine the Average Precision (AP), and averaging these AP values. We train the model for 80K steps using 8×H100 GPUs in BF16 format, with total batch size of 32, learning rate of 1×10^{-5}, and weight decay of 0.

Comparison Results. Table 3 presents the quantitative evaluation results of models trained with different optimizers. The convergence speed of Adafactor is significantly faster than AdamW, so we chose Adafactor for our experiments. Our method achieves results comparable to Adafactor while reducing optimizer memory usage by 80%, with only a 6% increase in additional training time. In contrast, methods like Flora and GaLore fail to converge at the same compression rate. Additionally, our method is also applicable to 8-bit, reducing optimizer memory usage by 90%. Table 4 compares images generated by models trained with different methods. Our method is able to generate human poses that align with the input at just 20K steps, while GaLore fails to converge even after 80K steps, leading to noticeable mismatches between the generated and input poses.

4.4. Pre-training LLaMA-1B and LLaMA-7B

Experimental Settings. For large language models, we follow the training settings described in [58] to train LLaMA-1B from scratch on the C4 dataset. Our training employs an initial learning rate of 0.01 and a total batch size of 512.

Comparison Results. As shown in Table 5, our method outperforms other methods. In LLaMA-1B, compared to other low-rank methods, we reduce the optimizer memory by 61% with only a 2% increase in additional training time, which is significantly lower than the training time increase of other methods, and achieve performance comparable to AdamW. In contrast, LoRA and ReLoRA require an additional 36% model size and result in perplexity (PPL) increases of 3.65 and 2.77, respectively. In LLaMA-7B using 8-bit optimizers, our method outperforms 8-bit Adam [8], reducing training time by 2% and lowering PPL by 0.11. Meanwhile, GaLore increases training time by 19% and suffers a PPL increase of 0.12.

4.5. Fine-tuning LLaVA-7B

Experimental Settings. We apply our method to a recent state-of-the-art Large Multimodal Model (LMM): LLaVA-v1.5-7B [25]. LLaVA connects the pre-trained CLIP ViT-L/14 [37] visual encoder and the large language model Vicuna [59] using a simple projection matrix for feature alignment. We perform fine-tuning on LLaVA based on this setup. We train the model using task-specific fine-tuning on the ScienceQA [31] dataset for 12 epochs with a batch size of 16 and a learning rate of 2×10^{-5} on 1×A100 GPU.

Table 5. Pre-training LLaMA-1B and LLaMA-7B on the C4 dataset using 8×H100 GPUs. PPL is reported along with GPU memory usage of optimizer states and the model in BF16 format. The rank is 512 for LLaMA-1B and 1024 for LLaMA-7B.

Model	Method	Optimizer Mem. (GB)↓	Model Mem. (GB)↓	Training Time↓	PPL↓
LLaMA 1B (100K)	AdamW	4.99	2.49	28.50 h	15.56
	GaLore	1.94 (-61%)	2.49	+17%	15.64
	LoRA	2.27 (-55%)	3.38 (+36%)	+6%	19.21
	ReLoRA	2.27 (-55%)	3.38 (+36%)	+6%	18.33
	COAP	**1.94 (-61%)**	2.49	**+2%**	**15.56**
LLaMA 7B (80K)	8-bit Adam	12.55	12.55	52.01 h	15.39
	8-bit GaLore	5.25 (-58%)	12.55	+19%	15.47
	8-bit COAP	**5.25 (-58%)**	12.55	**-2%**	**15.28**

Table 6. Fine-tuning LLaVA-v1.5-7B on the ScienceQA dataset using 1×A100. "OOM" means out-of-memory.

Method	Training Time↓	Optimizer Mem. (GB)↓	Model Mem. (GB)↓	ScienceQA IMG-Acc(%)↑
AdamW	-	OOM	OOM	-
DeepSpeed	47.1 h	26.4	13.2	82.4
GaLore	30.2 h	8.1 (-49%)	13.2	91.1
LoRA	11.1 h	8.1 (-49%)	17.2 (+30%)	**92.3**
Flora	9.5 h	8.1 (-49%)	13.2	66.6
COAP	**7.6 h**	8.1 (-49%)	13.2	**92.3**
8-bit GaLore	30.0 h	4.9 (-81%)	13.2	90.7
8-bit COAP	**7.8 h**	4.9 (-81%)	13.2	92.0

Comparison Results. Table 6 compares the results of fine-tuning LLaVA-7B on the ScienceQA dataset using different optimizers. In a single GPU environment, training large models directly with the AdamW optimizer can lead to Out-of-memory (OOM) errors due to the significant GPU memory consumption by optimizer states. Although DeepSpeed's CPU-offload [39] feature can alleviate GPU memory pressure by migrating optimizer states to CPU memory, it significantly increases training latency due to frequent data transfers. Compared to DeepSpeed and GaLore, our training speed is improved by 6× and 4×, respectively, with accuracy improvements of 9.9% and 1.2%. Compared to LoRA, our training speed is 1.4× faster while achieving the same performance without increasing the model size.

5. Ablation Study

5.1. Hyper-parameters of COAP

In algorithm 1, we introduce three key hyper-parameters: r, λ, and T_u. Rank r represents the compression level of the optimizer states, with a smaller rank indicating a higher compression rate. T_u denotes the update interval of the SGD-based iterative update (Eqn. 6), while $\lambda \times T_u$ repre-

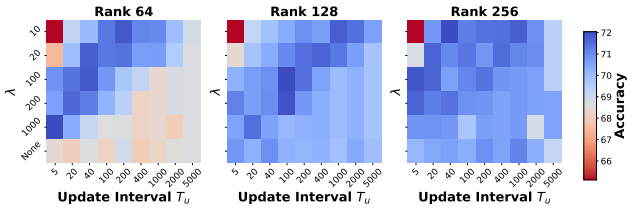


Figure 4. Ablation study on hyper-parameters λ, r, and T_u for DeiT-Base on CIFAR-100 over 300 epochs. Here, λ = None means that occasional low-cost SVD does not participate in updating the low-rank projection matrix.

Table 7. Ablation study of low-rank projection matrix updates in DeiT-Base model on 1×A100 GPU.

Method	w/ Eqn.7	w/ Eqn.6 CosSim.	w/ Eqn.6 MSE	Top-1 Acc Fine-tuning	Top-1 Acc Pre-training
AdamW (655.2 MB)	-	-	-	90.41	68.75
GaLore (169.2 MB)	-	-	-	91.09	65.58
COAP Rank=192 (169.2 MB)	✓	✓	✓	**91.42**	**70.39**
	✗	✓	✓	91.25	63.28
	✗	✓	✗	91.29	63.31
	✗	✗	✓	91.30	63.27
	✓	✗	✗	90.04	69.83
	✓	✓	✗	91.29	70.23
	✓	✗	✓	91.26	69.90

sents the update interval of Occasional Low-cost SVD (Eqn. 7).

As shown in Fig. 4, it is evident that all listed ranks can achieve baseline performance (68.75%). For the DeiT-Base model with a full-rank matrix of 768, when the rank is 256 or 128, the performance difference is minimal when $\lambda \geq 100$. However, when λ is too small, *e.g.*, 10, excessively small T_u values can lead to a drop in model performance. When the compression rate exceeds 10×, *i.e.*, $r = 64$, the model becomes highly sensitive to hyper-parameters. Additionally, it can be observed that selecting λ and T_u near the diagonal yields better performance.

5.2. Low-rank Projection Matrix Updates

Our proposed method for updating the low-rank projection matrix involves two key components: Inter-projection Correlation-aware $\boldsymbol{P}_t$ Update (Eqn. 6) and Occasional Low-cost SVD (Eqn. 7). To evaluate the benefits of these two update strategies on model performance, we conduct experiments using the DeiT-Base model. We train the model on the CIFAR-100 dataset using two strategies: training from scratch and fine-tuning a pre-trained model from ImageNet-1K, both for 300 epochs.

As shown in Table 7, for the task of training from scratch, the Occasional Low-cost SVD provides the most significant benefits. In contrast, the gains are minimal for the fine-tuning task. Regarding the reconstruction term and direction term in Eqn. 6, the direction term offers greater benefits. This indicates that during training from scratch, the model has not yet found an optimal update direction, thus

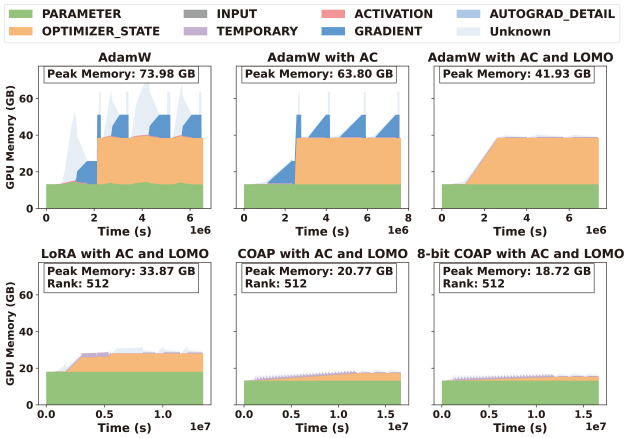


Figure 5. Profiling the GPU memory usage during the training stage of LLaVA-v1.5-7B on 1×A100.

requiring frequent adjustments through Occasional Low-cost SVD. On the other hand, for the fine-tuning task, the model is likely already in a good update direction, making the Inter-projection Correlation-aware update more beneficial. Combining both update strategies yields the best results for both training from scratch and fine-tuning.

5.3. GPU Memory Profiling

To validate the effectiveness of our method on GPU memory usage, we utilize PyTorch's Memory Profiler [43] to analyze the memory consumption of LLaVA-v1.5-7B during training. We use the AdamW optimizer as the baseline with batch size of 4 and categorize the memory usage according to PyTorch's convention. Since our method aims to reduce the optimizer states, we combined it with other complementary memory-efficient training techniques, *e.g.*, activation checkpointing (AC) [4], LOMO [32], and quantization to reduce the overall memory usage. As shown in the memory profile of AdamW, the optimizer states occupy a significant portion (36%) of the memory during training. with the remaining memory being used by gradients, activations, and model parameters. By enabling LOMO and AC, the memory used by gradients and activations can be reduced, but these techniques alone only reduce the peak memory usage to 41.93 GB. Finally, by incorporating our 8-bit COAP, peak memory usage is further reduced to 18.72 GB, achieving a total reduction of 75%. As shown in Table 6, this significant memory reduction has minimal impact on performance.

6. Conclusion

We introduce COAP, a memory-efficient approach that integrates seamlessly with momentum-based optimizers (*e.g.*, AdamW, Adafactor) for large-scale training across language, vision, and multimodal domains. We analyze the limitations of current low-rank gradient projection methods, and develop a correlation-aware projection update rule

that reduces computational costs and minimizes the performance gap with standard optimizers, while significantly cutting memory usage. Extensive experiments show that COAP outperforms competing methods in both training efficiency and model performance. We believe that, when combined with other memory-efficient techniques, COAP has strong potential to advance large-scale model training.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1

[2] Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*, 2024. 1

[3] Han Chen, Garvesh Raskutti, and Ming Yuan. Non-convex projected gradient descent for generalized low-rank tensor regression. *Journal of Machine Learning Research*, 20(5): 1–37, 2019. 2

[4] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 1, 2, 8

[5] Yudong Chen and Martin J Wainwright. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*, 2015. 2

[6] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000. 2

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 6

[8] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*, 2021. 2, 7

[9] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023. 1

[10] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[11] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023. 1

[12] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1

[13] Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors. In *Forty-first International Conference on Machine Learning*, 2024. 1, 2, 3

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 6

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 3

[16] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 1, 2, 3

[17] Tao Jiang, Peng Lu, Li Zhang, Ningsheng Ma, Rui Han, Chengqi Lyu, Yining Li, and Kai Chen. Rtmpose: Real-time multi-person pose estimation based on mmpose. *arXiv preprint arXiv:2303.07399*, 2023. 6

[18] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 1

[19] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1, 2, 3

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4, 3

[21] Pieter M Kroonenberg and Jan De Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika*, 45:69–97, 1980. 2

[22] Black Forest Labs. Announcing black forest labs, 2024. 1

[23] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023. 6

[24] Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. Relora: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*, 2023. 1, 2, 3

[25] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26296–26306, 2024. 1, 2, 4, 5, 7

[26] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024. 1, 5

[27] Ji Liu, Przemyslaw Musialski, Peter Wonka, and Jieping Ye. Tensor completion for estimating missing values in visual data. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):208–220, 2012. 1

[28] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024. 1, 2

[29] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015. 3

[30] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 2

[31] Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022. 7

[32] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. Full parameter fine-tuning for large language models with limited resources. *arXiv preprint arXiv:2306.09782*, 2023. 1, 8

[33] Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024. 5, 6

[34] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 2

[35] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. 1

[36] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 1, 5, 6

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 7

[38] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020. 3

[39] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021. 4, 7

[40] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021, 2021. 5

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1

[42] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR, 2018. 1, 2, 3

[43] Aaron Shi and Zachary DeVito. Understanding gpu memory usage in pytorch, 2024. 8

[44] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 1

[45] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021. 4

[46] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1, 2, 5

[47] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 1

[48] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 2

[49] Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*, 2024. 1

[50] Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning. *arXiv preprint arXiv:2401.04151*, 2024. 1, 2

[51] Jinqi Xiao, Miao Yin, Yu Gong, Xiao Zang, Jian Ren, and Bo Yuan. COMCAT: Towards efficient compression and customization of attention-based vision models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 38125–38136. PMLR, 2023. 2

[52] Jinqi Xiao, Chengming Zhang, Yu Gong, Miao Yin, Yang Sui, Lizhi Xiang, Dingwen Tao, and Bo Yuan. Haloc: hardware-aware automatic low-rank compression for compact neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10464–10472, 2023. 2

[53] SHIH-YING YEH, Yu-Guan Hsieh, Zhidong Gao, Bernard B W Yang, Giyeong Oh, and Yanmin Gong. Navigating text-to-image customization: From lyCORIS fine-tuning to model evaluation. In *The Twelfth International Conference on Learning Representations*, 2024. 2

[54] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffu-

sion transformers is easier than you think. *arXiv preprint arXiv:2410.06940*, 2024. 6

[55] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. 2

[56] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023. 5, 6

[57] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023. 1, 2, 3

[58] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024. 1, 2, 3, 7

[59] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023. 7

COAP: Memory-Efficient Training with Correlation-Aware Gradient Projection

Supplementary Material

1. Detailed Proposed Method

Notation. In this paper, we use the following notation conventions: Matrices and vectors are indicated with boldface capital and lowercase letters, *e.g.*, $\boldsymbol{X}$ and $\boldsymbol{x}$, respectively. Tensors are represented using boldface calligraphic script, denoted as $\boldsymbol{\mathcal{X}}$.

1.1. Inter-projection Correlation-aware P_t Update.

Considering the importance of incorporating the information from the previous projection into the current update, we propose calculating $\boldsymbol{P}_t$ via solving the optimization problem:

$$\min_{\boldsymbol{P}_t} \underbrace{\text{MSE}(\hat{\boldsymbol{G}}_t, \boldsymbol{G}_t)}_{\text{reconstruction term}} \underbrace{(1 - \text{CosSim}(\hat{\boldsymbol{M}}_{t-1}, \boldsymbol{G}_t))}_{\text{direction term}}, \quad (1)$$

where $\text{CosSim}(\cdot,\cdot)$ and $\text{MSE}(\cdot,\cdot)$ return the cosine similarity and mean squared error, respectively. To simplify notation, the notation without the subscript t represents a general form of the optimization problem, *i.e.*,

$$\min_{\boldsymbol{P}} \underbrace{\text{MSE}(\hat{\boldsymbol{G}}, \boldsymbol{G})}_{\text{reconstruction term}} \underbrace{(1 - \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G}))}_{\text{direction term}}, \quad (2)$$

where $\boldsymbol{P} \in \mathbb{R}^{n\times r}$, $\hat{\boldsymbol{G}} \in \mathbb{R}^{m\times n} = \boldsymbol{G}\boldsymbol{P}\boldsymbol{P}^\top$ and $\hat{\boldsymbol{M}} \in \mathbb{R}^{m\times n} = \boldsymbol{M}^{\text{proj}}\boldsymbol{P}^\top$ are the full-rank estimates of gradient and first-order moment projected back from the low-rank subspace, respectively. Notably, the *reconstruction term* is introduced to minimize the reconstruction error for gradients incurred by projection – achieving the similar goal that SVD essentially aims for, and *direction term* encourages the consistency of optimization direction after restoring from low-rank subspace.

To solve Eqn. 2 as a non-convex optimization problem, we propose using stochastic gradient descent to iteratively update $\boldsymbol{P}_t$ as follows:

$$\begin{aligned}\boldsymbol{P} :=& \boldsymbol{P} - \eta\Big(\frac{\partial \text{MSE}(\hat{\boldsymbol{G}}, \boldsymbol{G})}{\partial \boldsymbol{P}}(1 - \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G}) + \\ & \frac{\partial \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G})}{\partial \boldsymbol{P}}\text{MSE}(\hat{\boldsymbol{G}}, \boldsymbol{G})\Big).\end{aligned} \quad (3)$$

Here, η represents learning rate, set to 0.1 by default. The gradient expressions for the reconstruction term and the direction term are derived as follows:

Gradient of Reconstruction term (MSE).

$$\begin{aligned}\frac{\partial \text{MSE}(\hat{\boldsymbol{G}}, \boldsymbol{G})}{\partial \boldsymbol{P}} &= \frac{\partial}{\partial \boldsymbol{P}}\left(\frac{1}{mn}\text{tr}((\hat{\boldsymbol{G}} - \boldsymbol{G})^\top(\hat{\boldsymbol{G}} - \boldsymbol{G}))\right) \\ &= \frac{2}{mn}(\hat{\boldsymbol{G}}^\top\boldsymbol{G}\boldsymbol{P} - 2\boldsymbol{G}^\top\boldsymbol{G}\boldsymbol{P} + \boldsymbol{G}^\top\hat{\boldsymbol{G}}\boldsymbol{P}),\end{aligned} \quad (4)$$

where $\text{tr}(\cdot)$ represents the trace of a matrix.

Gradient of Direction Term (CosSim).

Given the cosine similarity between matrices $\hat{\boldsymbol{M}}$ and $\boldsymbol{G}$, defined as:

$$\begin{aligned}\text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G}) &= \frac{1}{m}\sum_{i=1}^{m}\text{CosSim}(\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i) \\ &= \frac{1}{m}\sum_{i=1}^{m}\frac{\langle\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i\rangle}{\|\hat{\boldsymbol{M}}_i\|\|\boldsymbol{G}_i\|}.\end{aligned} \quad (5)$$

Applying the chain rule to compute the gradient of $\text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G})$ with respect to $\boldsymbol{P}$:

$$\begin{aligned}\frac{\partial \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G})}{\partial \boldsymbol{P}} &= \Big(\frac{\partial \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G})}{\partial \hat{\boldsymbol{M}}}\Big)^\top\frac{\partial \hat{\boldsymbol{M}}}{\partial \boldsymbol{P}} \\ &= \frac{1}{m}\sum_{i=1}^{m}\Big(\frac{\partial \text{CosSim}(\hat{\boldsymbol{M}}, \boldsymbol{G})}{\partial \hat{\boldsymbol{M}}_i}\Big)^\top\boldsymbol{M}_i^{\text{proj}} \\ &= \frac{1}{m}\sum_{i=1}^{m}\Big(\frac{\boldsymbol{G}_i}{\|\hat{\boldsymbol{M}}_i\|\|\boldsymbol{G}_i\|} - \frac{\hat{\boldsymbol{M}}_i\langle\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i\rangle}{\|\hat{\boldsymbol{M}}_i\|^3\|\boldsymbol{G}_i\|}\Big)^\top\boldsymbol{M}_i^{\text{proj}},\end{aligned} \quad (6)$$

where $\|\cdot\|$ represents the Euclidean norm, $\langle\cdot,\cdot\rangle$ denotes the inner product, $\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i, \boldsymbol{M}_i^{\text{proj}}$ denotes the i-th row of $\hat{\boldsymbol{M}}, \boldsymbol{G}, \boldsymbol{M}^{\text{proj}}$.

Incorporating the gradient expressions above, we derive the final update formula for $\boldsymbol{P}$ as follows:

$$\begin{aligned}\boldsymbol{P} :=& \boldsymbol{P} - \eta\Big(\frac{2}{mn}(\hat{\boldsymbol{G}}^\top\boldsymbol{G}\boldsymbol{P} - 2\boldsymbol{G}^\top\boldsymbol{G}\boldsymbol{P} + \boldsymbol{G}^\top\hat{\boldsymbol{G}}\boldsymbol{P}) \\ & (1 - \frac{1}{m}\sum_{i=1}^{m}\frac{\langle\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i\rangle}{\|\hat{\boldsymbol{M}}_i\|\|\boldsymbol{G}_i\|}) + \\ & \frac{1}{m}\sum_{i=1}^{m}\Big(\frac{\boldsymbol{G}_i}{\|\hat{\boldsymbol{M}}_i\|\|\boldsymbol{G}_i\|} - \frac{\hat{\boldsymbol{M}}_i\langle\hat{\boldsymbol{M}}_i, \boldsymbol{G}_i\rangle}{\|\hat{\boldsymbol{M}}_i\|^3\|\boldsymbol{G}_i\|}\Big)^\top\boldsymbol{M}_i^{\text{proj}} \\ & \frac{1}{mn}\text{tr}((\hat{\boldsymbol{G}} - \boldsymbol{G})^\top(\hat{\boldsymbol{G}} - \boldsymbol{G}))\Big).\end{aligned} \quad (7)$$

1.2. Extension to CONV Layer.

To enhance the generality and applicability of our algorithm, it is essential to extend support to higher-dimensional weight tensors, which are prevalent in architectures such as Convolutional Neural Networks (CNNs). While the most straightforward approach would be to reshape CNN weights into matrices and apply the same low-rank space construction method used for matrix operations, this naive strategy would inevitably result in the loss of intrinsic spatial characteristics inherent to CNNs [27].

For a convolutional layer with a weight tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{O\times I\times K_1\times K_2}$, where I and O are the number of input

channels and output channels, respectively, and K_1 and K_2 are the kernel sizes, we use Tucker-2 decomposition [48] as the factorization method. In this scenario, $\mathcal{W}$ can be represented with a core tensor $\mathcal{C}$ and two factor matrices (U_1 and U_2) along each mode as follows:

$$\mathcal{W} = \mathcal{C} \times_1 U_1 \times_2 U_2, \tag{8}$$

where "$\times_n$" denotes the n-mode product. Specifically, $U_1 \in \mathbb{R}^{O \times r_O}$ represents the left singular vectors of the mode-1 unfolding of $\mathcal{W}$, denoted as $W_{(1)}$, *i.e.*, $W_{(1)} = U_1 \Sigma_1 V_1^\top$. Similarly, $U_2 \in \mathbb{R}^{I \times r_I}$ represents the left singular vectors of the mode-2 unfolding of $\mathcal{W}$, denoted as $W_{(2)}$, *i.e.*, $W_{(2)} = U_2 \Sigma_2 V_2^\top$. The core tensor $\mathcal{C}$ is obtained by projecting $\mathcal{W}$ onto the subspaces spanned by U_1 and U_2, i.e.,

$$\mathcal{C} = \mathcal{W} \times_1 U_1^\top \times_2 U_2^\top, \tag{9}$$

where $\mathcal{C} \in \mathbb{R}^{r_O \times r_I \times K_1 \times K_2}$. Here, r_O and r_I are the Tucker-2 tensor ranks, determining the dimensionality of the factor matrices and the core tensor. Initially, the values for U_1, U_2, and $\mathcal{C}$ are typically obtained using Higher-Order Singular Value Decomposition (HOSVD) [6]. To refine these initial values and achieve the final decomposition, the Alternating Least Squares (ALS) [21] method is employed. This iterative optimization technique alternates between updating the core tensor $\mathcal{C}$ and the factor matrices U_1 and U_2 to minimize the reconstruction error.

Algorithm 2 outlines the Adam-based training procedure, integrating the proposed P_t update method for convolutional layers.

Typically, for a convolutional layer with a weight tensor $\mathcal{W} \in \mathbb{R}^{O \times I \times K_1 \times K_2}$, I and O are significantly larger than the kernel sizes K_1 and K_2 (i.e., $I, O \gg K_1, K_2$). Therefore, we propose using the format of Tucker-2 decomposition to handle CNNs while employing our own decomposition method for the actual factorization. According to Eq. 8 and Eq. 9, the low-rank projection space of $\mathcal{G}_t$ becomes $[P_{O_t} \in \mathbb{R}^{O \times r_O}, P_{I_t} \in \mathbb{R}^{I \times r_I}]$. The gradient $\mathcal{G}_t$ in the low-rank space is $\mathcal{G}_t^{\text{proj}} = \mathcal{G}_t \times_1 P_{O_t}^\top \times_2 P_{I_t}^\top$, and the restored tensor from the low-rank space is $\hat{\mathcal{G}}_t = \mathcal{G}_t^{\text{proj}} \times_1 P_{O_t} \times_2 P_{I_t}$. Here, P_{O_t} and P_{I_t} can be updated according to Eqn. 7 and Occasional Low-cost SVD on the mode-1 and mode-2 unfolding of tensor $\mathcal{G}$, respectively.

1.3. Impact of Low-rank Matrix Projection Formats on CNN Models Performance.

We compare different formats of Tucker decomposition, *i.e.*, Tucker-1, Tucker-2, and Tucker. In this context, the default Tucker format applies projections along all dimensions of a tensor. For instance, if the tensor is 4-dimensional, it requires 4 projection matrices. Tucker-2, on the other hand, uses only two projection matrices, while Tucker-1 requires just one, making it a variant of SVD.

As shown in the Fig. 1, Tucker-2 achieves performance

Algorithm 2: Adam with COAP (CONV)

Input: Weight tensor $\mathcal{W} \in \mathbb{R}^{O \times I \times K_1 \times K_2}$, Learning rate η, Rank ratio α, Betas $[\beta_1, \beta_2]$, Update interval $[\lambda, T_u]$.

Initialize: $r_O = O\frac{1}{\sqrt{\alpha}}$, $r_I = I\frac{1}{\sqrt{\alpha}}$, $t \leftarrow 0$,
$\mathcal{M}_0^{\text{proj}} \in \mathbb{R}^{r_O \times r_I \times K_1 \times K_2} \leftarrow 0$,
$\mathcal{V}_0^{\text{proj}} \in \mathbb{R}^{r_O \times r_I \times K_1 \times K_2} \leftarrow 0$

Randomly Initialize: $P_O \in \mathbb{R}^{O \times r_O}$, $P_I \in \mathbb{R}^{I \times r_I}$

Define: $G_{O_t} \leftarrow \text{reshape}(\mathcal{G}_t, [O, IK_1K_2])$,
$G_{I_t} \leftarrow \text{reshape}(\mathcal{G}_t, [I, OK_1K_2])$

Compute: $P_{O_0} \leftarrow (P_O, G_{O_0}), P_{I_0} \leftarrow (P_I, G_{I_0})$
▷ Occasional Low-cost SVD

for t in $[1, 2, \cdots]$ **do**
  Compute: gradient G_t of W_t in the loss function.
  if $t \mod T_u = 0$ **then**
    if $t \mod (\lambda \times T_u) = 0$ **then**
      Compute: $P_{O_t} \leftarrow (P_{O_{t-1}}, G_{O_t})$,
      $P_{I_t} \leftarrow (P_{I_{t-1}}, G_{I_t})$
      ▷ Occasional Low-cost SVD
    else
      Update: $P_{O_t} \leftarrow (P_{O_{t-1}}, G_{O_t}, M_{O_{t-1}})$ ▷ Eqn. 7
      Update: $P_{I_t} \leftarrow (P_{I_{t-1}}, G_{I_t}, M_{I_{t-1}})$ ▷ Eqn. 7
  else
    $P_{O_t} \leftarrow P_{O_{t-1}}, P_{I_t} \leftarrow P_{I_{t-1}}$
  ▷ Project gradient and moments into low-rank space.
  $\mathcal{G}_t^{\text{proj}} \leftarrow \mathcal{G}_t \times_1 P_{O_t}^\top \times_2 P_{I_t}^\top$
  $\mathcal{M}_t^{\text{proj}} \leftarrow \beta_1 \mathcal{M}_{t-1}^{\text{proj}} + (1 - \beta_1)\mathcal{G}_t^{\text{proj}}$
  $\mathcal{V}_t^{\text{proj}} \leftarrow \beta_2 \mathcal{V}_{t-1}^{\text{proj}} + (1 - \beta_2)(\mathcal{G}_t^{\text{proj}})^2$
  ▷ Calculate the bias correction term in low-rank space.
  $\Delta\mathcal{W}_t^{\text{proj}} \leftarrow \frac{\mathcal{M}_t^{\text{proj}}/(1-\beta_1^t)}{\sqrt{\mathcal{V}_t^{\text{proj}}/(1-\beta_2^t)}+\epsilon}$
  ▷ Restore $\Delta\mathcal{W}_t^{\text{proj}}$ to original space and update $\mathcal{W}$.
  $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} - \eta\Delta\mathcal{W}_t^{\text{proj}} \times_1 P_{O_t} \times_2 P_{I_t}$

Return: updated $\mathcal{W}$

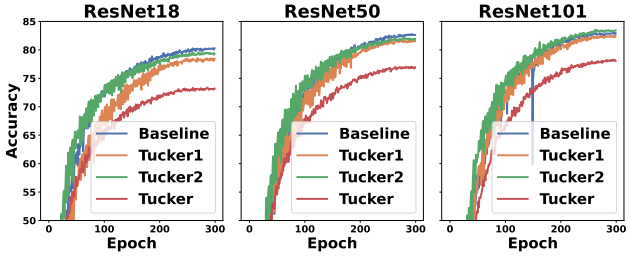


Figure 1. The comparison of Top-1 accuracy for ResNet under different low-rank projection formats is conducted with a rank ratio of 4. The models are trained for 100 epochs on the CIFAR-100 dataset.

Table 1. Pre-training DDPM on CIFAR-10 and CelebA-HQ datasets on 8×V100. FID scores are reported, along with the GPU memory usage of optimizer states in FP32 format.

Dataset	Method	Rank Ratio	Optimizer Mem. (MB)↓	FID↓
CIFAR-10 (32×32) 800K steps	AdamW	-	272.72	5.42
	GaLore	1.5	302.43	6.09
	COAP	1.5	**214.66**	5.66
	Adafactor	-	222.71	5.43
	GaLore	1.5	196.11	7.14
	COAP	1.5	**180.87**	**5.41**
CelebA-HQ (256×256) 460K steps	AdamW	-	867.26	12.82
	GaLore	2	562.56	27.95
	COAP	2	**525.18**	17.37
	Adafactor	-	714.64	12.38
	GaLore	2	549.27	19.12
	COAP	2	**447.59**	**12.30**

closest to the baseline across different scales of ResNet models. Thus, we select Tucker-2 as the primary format for computing convolution projection matrices.

2. Experimental Results

2.1. Pre-training DDPM

Experimental Settings. We implement DDPM based on the Diffusers from Hugging Face and conduct experiments on 8×V100 GPUs following the training and evaluation settings in [15]. For CIFAR-10 [20], the model is trained with a batch size of 128 for 800K steps. For CelebA-HQ [29], the model is trained with a batch size of 64 for 460K steps. We generate 50K images to compute the FID (Frechet Inception Distance) with respect to the training dataset and the images generated with 1000 DDPM steps.

Comparison Results. Table 1 presents the performance of our method and GaLore when compressing AdamW and Adafactor optimizers on the DDPM model. Our approach consistently outperforms GaLore on CIFAR-10 and CelebA-HQ datasets. Specifically, using the Adafactor optimizer, our method reduces FID by 1.7 and 6.8 compared to GaLore. Additionally, at compression rates of 1.2× and 1.6×, our approach surpasses the baseline performance.

2.2. Qualitative Comparisons

We present qualitative results, showcasing images generated by models trained with COAP and other optimizers. These results provide a visual comparison that complements the quantitative analysis in the main paper. Comparisons for DDPM (CIFAR-10), DDPM (CelebA-HQ), LDM, SiT-XL/2, and ControlNet-XL are detailed in Tables 2, 3, 4, 5, and 6, respectively.

https://github.com/huggingface/diffusers/tree/main/examples/unconditional_image_generation

Table 2. Comparison of images generated by DDPM trained on CIFAR-10 with different Adam-based optimizers.

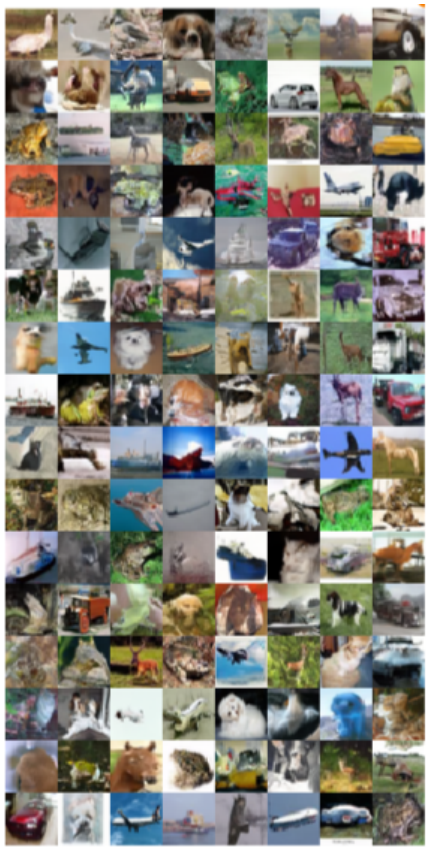
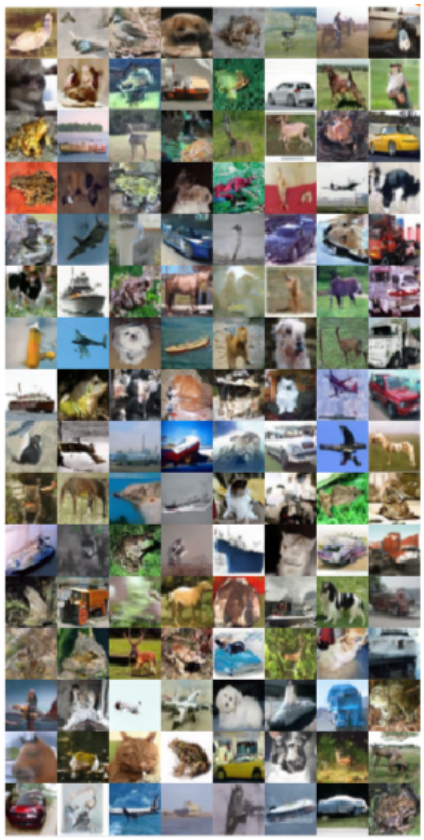
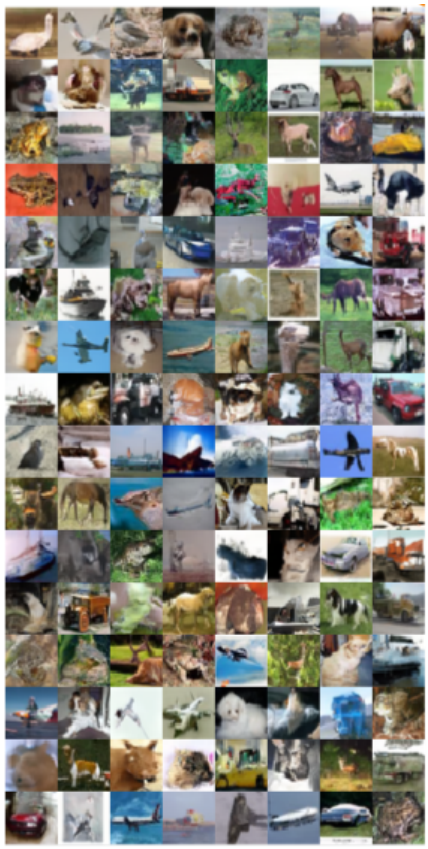
Adam (272.7 MB)	GaLore (302.4 MB)	**COAP (214.7 MB)**
		

Table 3. Comparison of images generated by DDPM trained on CelebA-HQ with different Adafactor-based optimizers.

Adafactor (714.6 MB)	
GaLore (549.3 MB)	
COAP (447.6 MB)	

Table 4. Random class-conditional samples generated by LDM trained on the ImageNet dataset using the COAP optimizer.

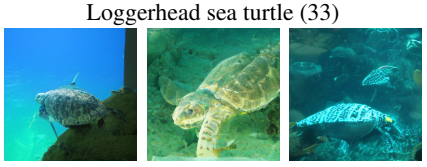
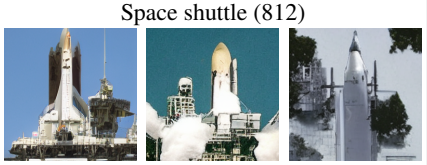
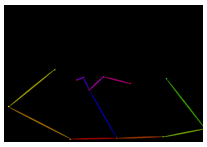
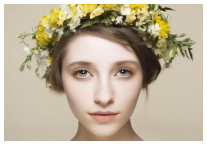
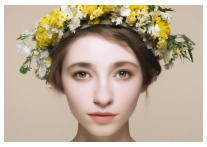
Loggerhead sea turtle (33)	Sulphur-crested cockatoo (89)	Golden retriever (207)
		
Husky (250)	Panda (388)	Balloon (417)
		
Baseball (429)	Space shuttle (812)	Volcano (980)
		

Table 5. Random class-conditional samples generated by SiT-XL/2 trained on the ImageNet dataset using the COAP optimizer.

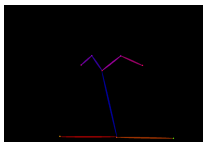
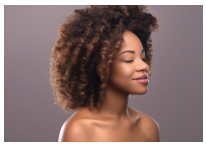
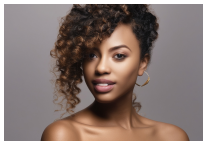
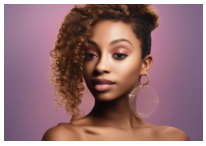
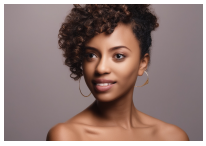
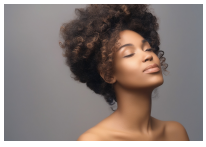
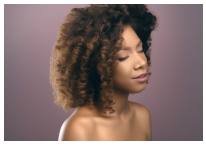
Loggerhead sea turtle (33)	Sulphur-crested cockatoo (89)	Golden retriever (207)
		
Husky (250)	Panda (388)	Balloon (417)
		
Baseball (429)	Space shuttle (812)	Volcano (980)
		

Table 6. Comparison of images generated at different training steps (20K, 40K, 80K) with ControlNet-XL trained under various optimizers. DDIM with a guidance scale of 5.0 is applied for image generation, with the number of inference steps set to 50.

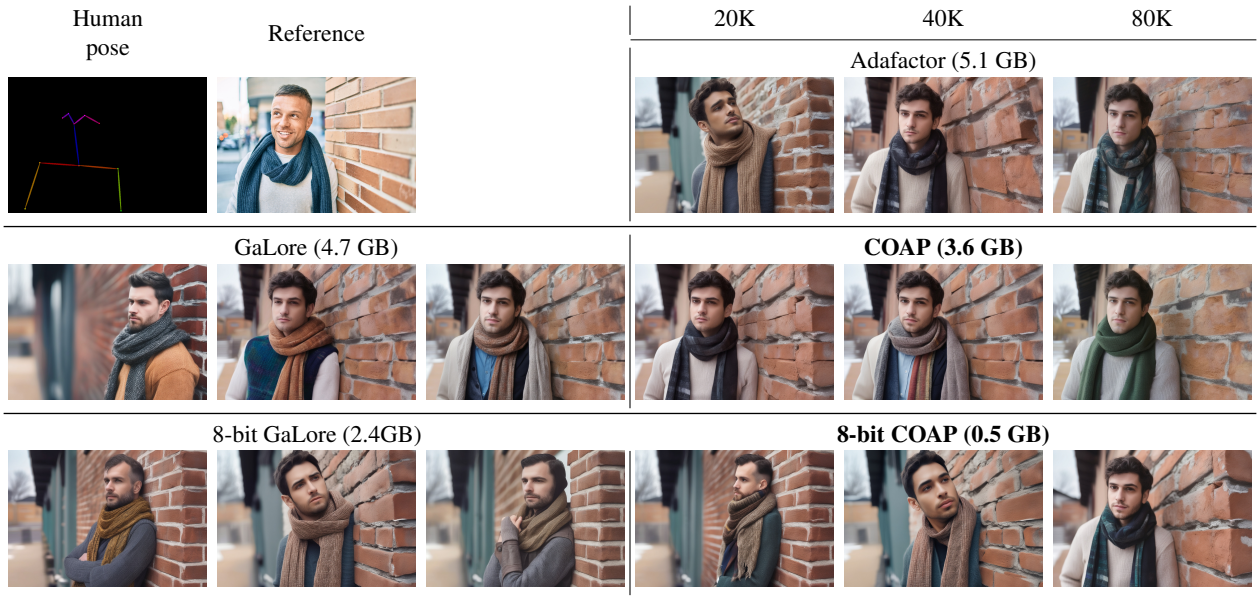
Prompt: a young woman with a yellow flower crown on her head

Human pose	Reference		20K	40K	80K
			Adafactor (5.1 GB)		
					
GaLore (4.7 GB)			**COAP (3.6 GB)**		
					
8-bit GaLore (2.4GB)			**8-bit COAP (0.5 GB)**		
					

Prompt: beautiful african american woman with curly hair on blue background

Human pose	Reference		20K	40K	80K
			Adafactor (5.1 GB)		
					
GaLore (4.7 GB)			**COAP (3.6 GB)**		
					
8-bit GaLore (2.4GB)			**8-bit COAP (0.5 GB)**		
					

Prompt: a man in a scarf and sweater leaning against a brick wall



Prompt: a young girl with her hands painted with colorful paint