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. Theory Analysis.

COAP proposes to approximately keep the same low-rank subspace across iterations. Intuitively, this requires the low-rank subspace to remain stable across iterations, which can be bounded in terms of the learning rate and Lipchitz constant. Theoretical justification composed of the following steps:

- Bounded update will lead to bounded change of the subspace.
- Equation 2 finds a subspace not much worse than the true subspace.

We aim to demonstrate that this procedure guarantees an approximation to the underlying true subspace. Let $\boldsymbol{P}_t^\natural$ be the true subspace. Let $\boldsymbol{P}_t^{\text{MSE}}$ and $\boldsymbol{P}_t^{\text{SIM}}$ be the solutions of the MSE loss and CosSim loss respectively, then

$$\begin{aligned} \boldsymbol{P}_t^{\text{MSE}} &= \boldsymbol{P}_{t-1} \\ \boldsymbol{P}_t^{\text{SIM}} &= \text{argmax} \left\langle \boldsymbol{G}_t \boldsymbol{P}_t, \boldsymbol{M}_{t-1}^{proj} \right\rangle, \end{aligned} \tag{8}$$

where $\boldsymbol{M}_{t-1}^{\text{proj}} = \beta \boldsymbol{M}_{t-2}^{\text{proj}} + (1-\beta)\boldsymbol{P}_t^T \boldsymbol{G}_t = \sum_{i=0}^{t-1} \beta^i (1-\beta) \boldsymbol{P}_{t-i}^T \boldsymbol{G}_{t-i}$, argmax $\langle \cdot, \cdot \rangle$ maximizes an inner product between two entities. Therefore,

$$\begin{aligned} \boldsymbol{P}_t^{\text{SIM}} &= \sum_{i=0}^{t-1} \beta^i (1-\beta) \text{argmax} \left\langle \boldsymbol{P}_t^T \boldsymbol{G}_t, \boldsymbol{P}_{t-i}^T \boldsymbol{G}_{t-i} \right\rangle \\ &= \sum_{i=0}^{t-1} \beta^i (1-\beta) \boldsymbol{P}_{t-i} \end{aligned} \tag{9}$$

Theorem 1.1. Assume that the gradient is Lipchitz with respect to the weight with Lipchitz constant L. Assume $\kappa_1 \doteq \frac{\sigma_1}{\sigma_{r+1}} > 1$ and $\kappa_r \doteq \frac{\sigma_r}{\sigma_{r+1}} > 1$ to be the ratio between the 1-th and r-th to the $r+1$-th singular value of the gradient $\boldsymbol{G}$ respectively. Then the proposed procedure in Equation 2 recovers a subspace well approximating the true subspace.

$$\begin{aligned} &\left\| \boldsymbol{P}_t - \boldsymbol{P}_t^\natural \right\|_F \\ &\leq c(t \mod (\lambda \times T_u)) \frac{r\kappa_1 + \min(m,n) - r}{\kappa_r - 1} L\eta. \end{aligned} \tag{10}$$

With learning rate η, Lipchitz property of the gradient gives

$$\begin{aligned} \|\boldsymbol{G}_{t-1} - \boldsymbol{G}_t\|_2 &\leq L \|\boldsymbol{W}_{t-1} - \boldsymbol{W}_t\|_2 \\ &= L \left\| \eta \boldsymbol{G}_{t-1}^{proj} \right\|_2 \leq L\eta \|\boldsymbol{G}_{t-1}\|_2 . \end{aligned} \tag{11}$$

Consequently, the optimal subspace $\boldsymbol{P}_t^\natural$ is also close enough to the previous optimal subspace $\boldsymbol{P}_{t-1}^\natural$:

$$\left\| \boldsymbol{P}_{t-1}^\natural - \boldsymbol{P}_t^\natural \right\|_F \leq \frac{r\kappa_1 + \min(m,n) - r}{\kappa_r - 1} L\eta. \tag{12}$$

When $t \mod (\lambda \times T_u) = 0$, the proposed algorithm updates $\boldsymbol{P}_t = \boldsymbol{P}_t^\natural$. Otherwise, we show that $\boldsymbol{P}_t$ highly approximates $\boldsymbol{P}_t^\natural$,

$$\begin{aligned} &\left\| \boldsymbol{P}_t - \boldsymbol{P}_t^\natural \right\|_F \leq \left\| \boldsymbol{P}_{t-1}^\natural - \boldsymbol{P}_t^\natural \right\|_F + \left\| \boldsymbol{P}_t - \boldsymbol{P}_{t-1}^\natural \right\|_F \\ &\leq \left\| \boldsymbol{P}_{t-1}^\natural - \boldsymbol{P}_t^\natural \right\|_F + \left\| \boldsymbol{P}_t^{\text{MSE}} - \boldsymbol{P}_{t-1}^\natural \right\|_F + \left\| \boldsymbol{P}_t^{\text{SIM}} - \boldsymbol{P}_{t-1}^\natural \right\|_F \\ &\leq \left\| \boldsymbol{P}_{t-1}^\natural - \boldsymbol{P}_t^\natural \right\|_F + (2-\beta) \left\| \boldsymbol{P}_{t-1} - \boldsymbol{P}_{t-1}^\natural \right\|_F \\ &\quad + (1-\beta) \sum_{i=2} \beta^i \left\| \boldsymbol{P}_{t-i} - \boldsymbol{P}_{t-1}^\natural \right\|_F \\ &\leq \left\| \boldsymbol{P}_{t-1}^\natural - \boldsymbol{P}_t^\natural \right\|_F + 2 \left\| \boldsymbol{P}_{t-1} - \boldsymbol{P}_{t-1}^\natural \right\|_F \\ &\quad + (1-\beta) \sum_{i=2} \beta^i \left\| \boldsymbol{P}_{t-i} - \boldsymbol{P}_{t-1} \right\|_F \\ &\leq 2^{(t \mod (\lambda \times T_u))} \frac{r\kappa_1 + \min(m,n) - r}{\kappa_r - 1} L\eta \end{aligned} \tag{13}$$

1.3. Hyper-parameter Settings.

Table 1 presents the detailed hyper-parameters used in all experiments of COAP. Except for the Update Interval (T_u) and Re-project Factor (λ) specific to COAP, all other parameters remain consistent with the corresponding baselines. The optimizer, rank settings, learning rate, GPU configuration, and mixed precision are listed for each experiment. Hyper-parameters for different models, including LDM, SiT-XL/2, ControlNet SDXL, and LLaMA series, are provided to ensure reproducibility.

1.4. Adafactor with COAP.

Adafactor is an adaptive gradient optimization algorithm designed to reduce memory usage while maintaining training efficiency. Unlike Adam, which stores full-matrix second-moment estimates, Adafactor factorizes the second-moment accumulation to save memory, making it particularly suitable for training large-scale models. However, first-moment remains crucial for stabilizing training and accelerating convergence. Without it, optimization can become unstable, especially in deep networks where gradients vary significantly. The first moment helps smooth updates and improves the overall training dynamics. Algorithm 1 describes the Adafactor-based training procedure, by projecting the gradient and its first-moment estimate into a lower-dimensional subspace, we effectively compress the momentum while retaining its benefits. This approach allows us to maintain training stability with reduced memory overhead, making it highly efficient for large-scale models.

1.5. 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

Table 1. Table 1: Hyper-parameter settings for all experiments.

Optimizer	Exprement	Rank r	Rank Ratio α	Update Interval T_u	Re-project Factor λ	Learning Rate η	GPU	Mixed-precision
Adafactor (COAP)	Table 1. Pre-training LDM	-	2	16	10	2×10^{-5}	8×V100	FP32
	Table 2. Pre-training SiT-XL/2	512	-	200	5	1×10^{-4}	8×H100	FP16
	Table 3. ControlNet SDXL	-	2	8	10	1×10^{-5}	8×H100	BF16
		-	4	8	10	1×10^{-5}	8×H100	BF16
		-	8	8	10	1×10^{-5}	8×H100	BF16
AdamW (COAP)	Table 1. Pre-training LDM	-	2	16	10	2×10^{-5}	8×V100	FP32
	Table 2. Pre-training SiT-XL/2	512	-	30	10	1×10^{-4}	8×H100	FP16
	Table 5. Pre-training LLaMA-1B	512	-	40	5	1×10^{-2}	8×H100	BF16
	Table 5. Pre-training LLaMA-7B	1024	-	100	1	5×10^{-3}	8×H100	BF16
	Table 6. Fine-tuning LLaVA-v1.5-7B	-	4	32	1	2×10^{-5}	1×A100	BF16

Algorithm 1: Adafactor 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 $[\lambda, T_\mathrm{u}]$, Decay rate γ.
Initialize: $\boldsymbol{M}_0^{\mathrm{proj}} \in \mathbb{R}^{m \times r} \leftarrow 0$, $\boldsymbol{V}_0^{\mathrm{proj}} \in \mathbb{R}^{m \times r} \leftarrow 0$, $t \leftarrow 0$
$\boldsymbol{R}_0 \in \mathbb{R}^{m \times 1} \leftarrow 0$, $\boldsymbol{C}_0 \in \mathbb{R}^{1 \times r} \leftarrow 0$.
Randomly Initialize: $\boldsymbol{P}_0 \in \mathbb{R}^{n \times r}$
Compute: $\boldsymbol{P}_0 \leftarrow (\boldsymbol{P}_0, \boldsymbol{G}_0)$ ▷ Occasional Low-cost SVD
for t in $[1, 2, \cdots]$ **do**
  Compute: gradient $\boldsymbol{G}_t$ of $\boldsymbol{W}_t$ in the loss function.
  if $t \mod T_\mathrm{u} = 0$ **then**
    if $t \mod (\lambda \times T_\mathrm{u}) = 0$ **then**
      ▷ Occasional Low-cost SVD
      Compute: $\boldsymbol{P}_t \leftarrow (\boldsymbol{P}_{t-1}, \boldsymbol{G}_t)$
    else
      Update: $\boldsymbol{P}_t \leftarrow (\boldsymbol{P}_{t-1}, \boldsymbol{G}_t, \boldsymbol{M}_{t-1})$ ▷ Eqn. 2
  else
    $\boldsymbol{P}_t \leftarrow \boldsymbol{P}_{t-1}$
  ▷Project gradient and moments into low-rank space.
  $\beta_2 \leftarrow 1 - t^{\gamma}$
  $\boldsymbol{G}_t^{\mathrm{proj}} \leftarrow \boldsymbol{G}_t \boldsymbol{P}_t$
  $\boldsymbol{M}_t^{\mathrm{proj}} \leftarrow \beta_1 \boldsymbol{M}_{t-1}^{\mathrm{proj}} + (1 - \beta_1) \boldsymbol{G}_t^{\mathrm{proj}}$
  $\boldsymbol{R}_t = \beta_2 \boldsymbol{R}_{t-1} + (1 - \beta_2) \cdot \mathrm{Sum}(\boldsymbol{G}_t^{\mathrm{proj}^2}, -1)$
  $\boldsymbol{C}_t = \beta_2 \boldsymbol{C}_{t-1} + (1 - \beta_2) \cdot \mathrm{Sum}(\boldsymbol{G}_t^{\mathrm{proj}^2}, -2)$
  $\hat{\boldsymbol{V}}_t = \sqrt{\frac{\mathrm{Mean}(\boldsymbol{R}_t, -1)}{\boldsymbol{R}_t \boldsymbol{C}_t}}$
  ▷Calculate the bias correction term in low-rank space.
  $\Delta \boldsymbol{W}_t^{\mathrm{proj}} \leftarrow \beta_1 \boldsymbol{M}_t^{\mathrm{proj}} + (1 - \beta_1) \eta \hat{\boldsymbol{V}}_t \odot \boldsymbol{G}_t^{\mathrm{proj}}$
  ▷Restore $\Delta \boldsymbol{W}_t^{\mathrm{proj}}$ to original space and update $\boldsymbol{W}$.
  $\boldsymbol{W}_t \leftarrow \boldsymbol{W}_{t-1} - \Delta \boldsymbol{W}_t^{\mathrm{proj}} \boldsymbol{P}_t^{\top}$
Return: updated $\boldsymbol{W}$

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 [5].

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 [7] as the factorization method. In this scenario, $\boldsymbol{\mathcal{W}}$ can be represented with a core tensor $\boldsymbol{\mathcal{C}}$ and two factor matrices ($\boldsymbol{U}_1$ and $\boldsymbol{U}_2$) along each mode as follows:

$$\boldsymbol{\mathcal{W}} = \boldsymbol{\mathcal{C}} \times_1 \boldsymbol{U}_1 \times_2 \boldsymbol{U}_2, \tag{14}$$

where "$\times_n$" denotes the n-mode product. Specifically, $\boldsymbol{U}_1 \in \mathbb{R}^{O \times r_O}$ represents the left singular vectors of the mode-1 unfolding of $\boldsymbol{\mathcal{W}}$, denoted as $\boldsymbol{W}_{(1)}$, *i.e.*, $\boldsymbol{W}_{(1)} = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^{\top}$. Similarly, $\boldsymbol{U}_2 \in \mathbb{R}^{I \times r_I}$ represents the left singular vectors of the mode-2 unfolding of $\boldsymbol{\mathcal{W}}$, denoted as $\boldsymbol{W}_{(2)}$, *i.e.*, $\boldsymbol{W}_{(2)} = \boldsymbol{U}_2 \boldsymbol{\Sigma}_2 \boldsymbol{V}_2^{\top}$. The core tensor $\boldsymbol{\mathcal{C}}$ is obtained by projecting $\boldsymbol{\mathcal{W}}$ onto the subspaces spanned by $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$, i.e.,

$$\boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{W}} \times_1 \boldsymbol{U}_1^{\top} \times_2 \boldsymbol{U}_2^{\top}, \tag{15}$$

where $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{r_O \times r_I \times K_1 \times K_2}$. 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 $\boldsymbol{U}_1$, $\boldsymbol{U}_2$, and $\boldsymbol{\mathcal{C}}$ are typically obtained using Higher-Order Singular Value Decomposition (HOSVD) [1]. To refine these initial values and achieve the final decomposition, the Alternating Least Squares (ALS) [4] method is employed. This iterative optimization technique alternates between updating the core tensor $\boldsymbol{\mathcal{C}}$ and the factor matrices $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ to minimize the reconstruction error.

Algorithm 2 outlines the Adam-based training procedure, integrating the proposed $\boldsymbol{P}_t$ update method for convolutional layers.

Typically, for a convolutional layer with a weight tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{O \times I \times K_1 \times K_2}$, 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. 14 and Eq. 15, the low-rank projection space of $\boldsymbol{\mathcal{G}}_t$ becomes $[\boldsymbol{P}_{O_t} \in \mathbb{R}^{O\times r_O}, \boldsymbol{P}_{I_t} \in \mathbb{R}^{I\times r_I}]$. The gradient $\boldsymbol{\mathcal{G}}_t$ in the low-rank space is $\boldsymbol{\mathcal{G}}_t^{\text{proj}} = \boldsymbol{\mathcal{G}}_t \times_1 \boldsymbol{P}_{O_t}^\top \times_2 \boldsymbol{P}_{I_t}^\top$, and the restored tensor from the low-rank space is $\hat{\boldsymbol{\mathcal{G}}}_t = \boldsymbol{\mathcal{G}}_t^{\text{proj}} \times_1 \boldsymbol{P}_{O_t} \times_2 \boldsymbol{P}_{I_t}$. Here, $\boldsymbol{P}_{O_t}$ and $\boldsymbol{P}_{I_t}$ can be updated according to Eqn. 7 and Occasional Low-cost SVD on the mode-1 and mode-2 unfolding of tensor $\boldsymbol{\mathcal{G}}$, respectively.

Algorithm 2: Adam with COAP (CONV)

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

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

Randomly Initialize: $\boldsymbol{P}_O \in \mathbb{R}^{O\times r_O}, \boldsymbol{P}_I \in \mathbb{R}^{I\times r_I}$

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

Compute: $\boldsymbol{P}_{O_0} \leftarrow (\boldsymbol{P}_O, \boldsymbol{G}_{O_0}), \boldsymbol{P}_{I_0} \leftarrow (\boldsymbol{P}_I, \boldsymbol{G}_{I_0})$ ▷ Occasional Low-cost SVD

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: $\boldsymbol{P}_{O_t} \leftarrow (\boldsymbol{P}_{O_{t-1}}, \boldsymbol{G}_{O_t})$, $\boldsymbol{P}_{I_t} \leftarrow (\boldsymbol{P}_{I_{t-1}}, \boldsymbol{G}_{I_t})$ ▷ Occasional Low-cost SVD

    else

      Update: $\boldsymbol{P}_{O_t} \leftarrow (\boldsymbol{P}_{O_{t-1}}, \boldsymbol{G}_{O_t}, \boldsymbol{M}_{O_{t-1}})$ ▷ Eqn. 7

      Update: $\boldsymbol{P}_{I_t} \leftarrow (\boldsymbol{P}_{I_{t-1}}, \boldsymbol{G}_{I_t}, \boldsymbol{M}_{I_{t-1}})$ ▷ Eqn. 7

  else

    $\boldsymbol{P}_{O_t} \leftarrow \boldsymbol{P}_{O_{t-1}}, \boldsymbol{P}_{I_t} \leftarrow \boldsymbol{P}_{I_{t-1}}$

  ▷ Project gradient and moments into low-rank space.

  $\boldsymbol{\mathcal{G}}_t^{\text{proj}} \leftarrow \boldsymbol{\mathcal{G}}_t \times_1 \boldsymbol{P}_{O_t}^\top \times_2 \boldsymbol{P}_{I_t}^\top$

  $\boldsymbol{\mathcal{M}}_t^{\text{proj}} \leftarrow \beta_1\boldsymbol{\mathcal{M}}_{t-1}^{\text{proj}} + (1-\beta_1)\boldsymbol{\mathcal{G}}_t^{\text{proj}}$

  $\boldsymbol{\mathcal{V}}_t^{\text{proj}} \leftarrow \beta_2\boldsymbol{\mathcal{V}}_{t-1}^{\text{proj}} + (1-\beta_2)(\boldsymbol{\mathcal{G}}_t^{\text{proj}})^2$

  ▷ Calculate the bias correction term in low-rank space.

  $\Delta\boldsymbol{\mathcal{W}}_t^{\text{proj}} \leftarrow \frac{\boldsymbol{\mathcal{M}}_t^{\text{proj}}/(1-\beta_1^t)}{\sqrt{\boldsymbol{\mathcal{V}}_t^{\text{proj}}/(1-\beta_2^t)}+\epsilon}$

  ▷ Restore $\Delta\boldsymbol{\mathcal{W}}_t^{\text{proj}}$ to original space and update $\boldsymbol{\mathcal{W}}$.

  $\boldsymbol{\mathcal{W}}_t \leftarrow \boldsymbol{\mathcal{W}}_{t-1} - \eta\Delta\boldsymbol{\mathcal{W}}_t^{\text{proj}} \times_1 \boldsymbol{P}_{O_t} \times_2 \boldsymbol{P}_{I_t}$

Return: updated $\boldsymbol{\mathcal{W}}$

1.6. 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.

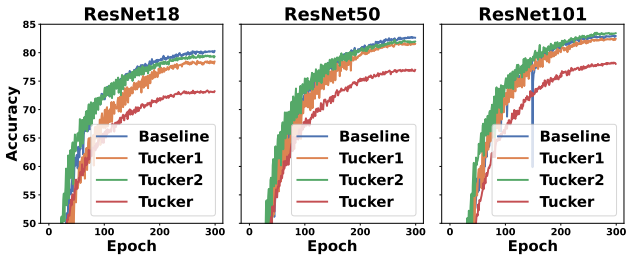


Figure 1. The comparison of Top-1 accuracy for ResNet under different low-rank projection formats is conducted with a rank ratio of 4. The models are trained for 100 epochs on the CIFAR-100 dataset.

Table 2. 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**

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

The COAP optimizer is versatile and can be applied to various model training scenarios. We have provided training results for DDPM and ControlNet, and this method is also suitable for other application scenarios [8–11].

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 [2]. For CIFAR-10 [3], the model is trained with a

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

batch size of 128 for 800K steps. For CelebA-HQ [6], 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 2 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 3, 4, 5, 6, and 7, respectively.

Table 3. Comparison of images generated by DDPM trained on CIFAR-10 with different Adam-based optimizers.

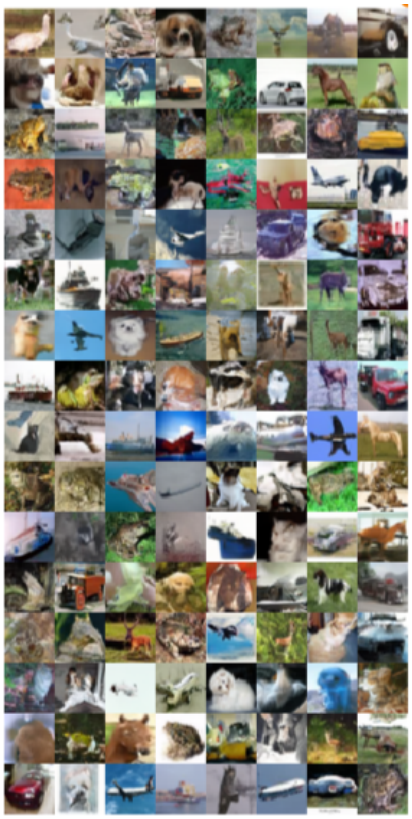
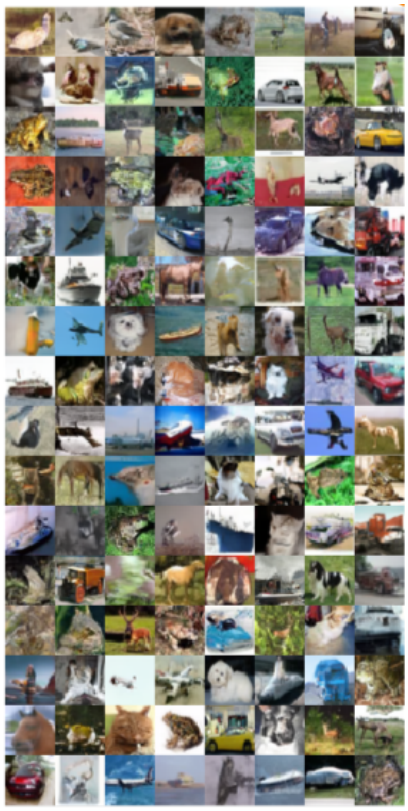
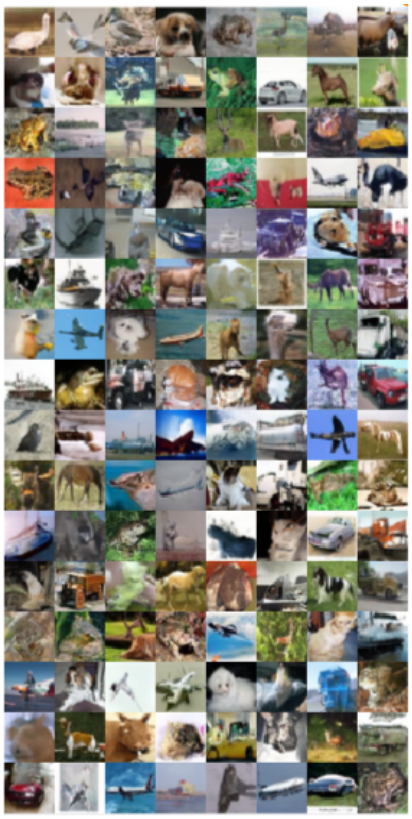
Adam (272.7 MB)	GaLore (302.4 MB)	**COAP (214.7 MB)**
		

Table 4. Comparison of images generated by DDPM trained on CelebA-HQ with different Adafactor-based optimizers.

Table 5. Random class-conditional samples generated by LDM trained on the ImageNet dataset using the COAP optimizer.

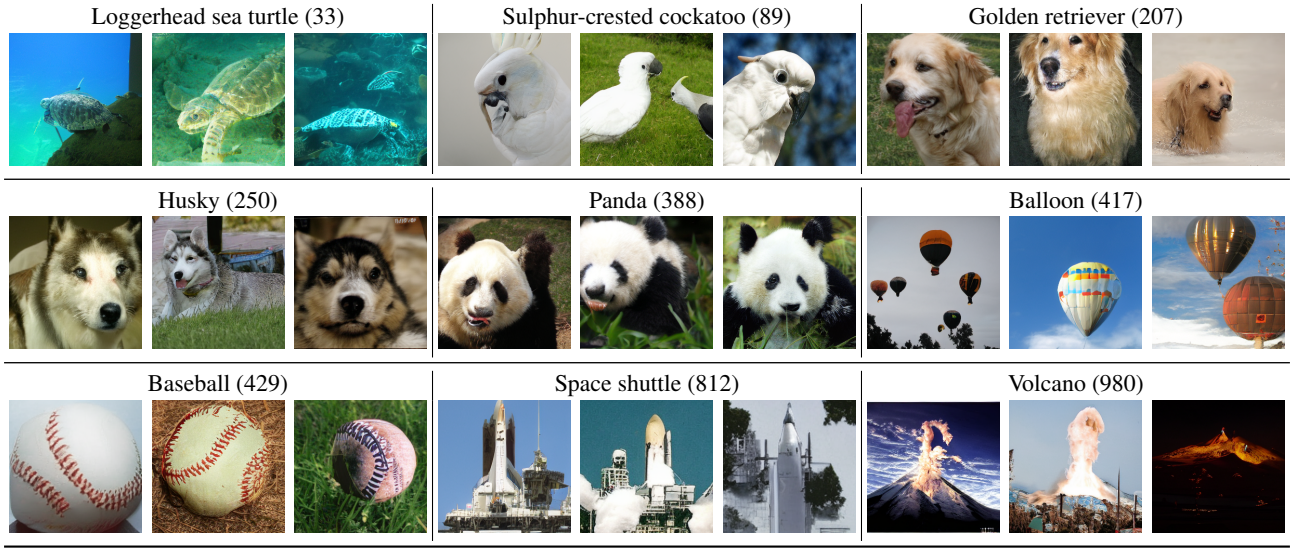


Table 6. Random class-conditional samples generated by SiT-XL/2 trained on the ImageNet dataset using the COAP optimizer.

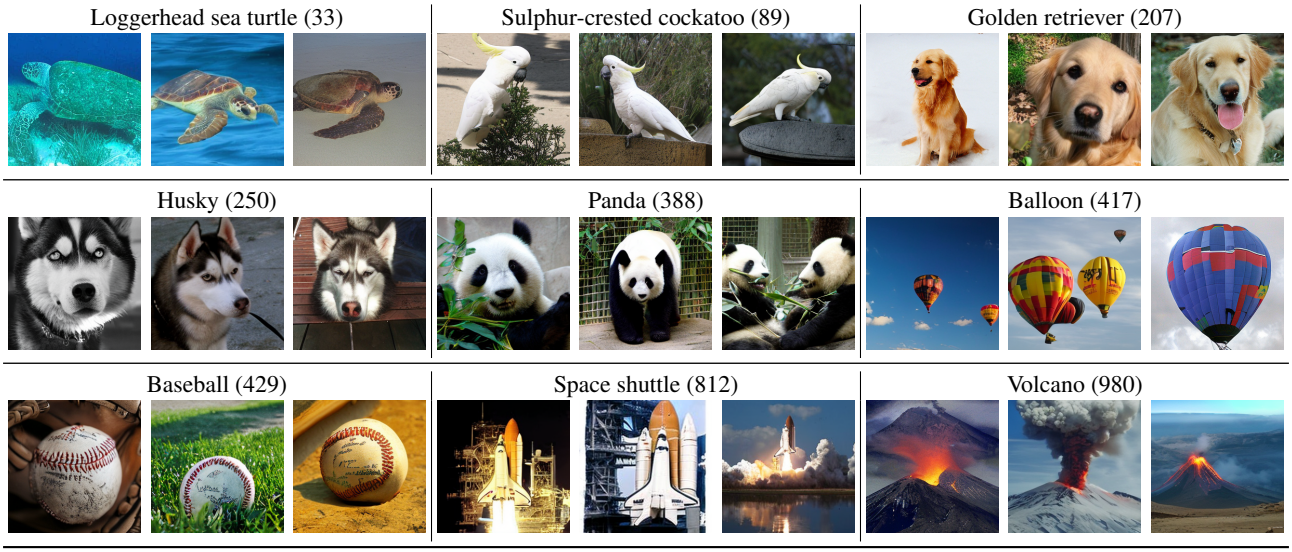
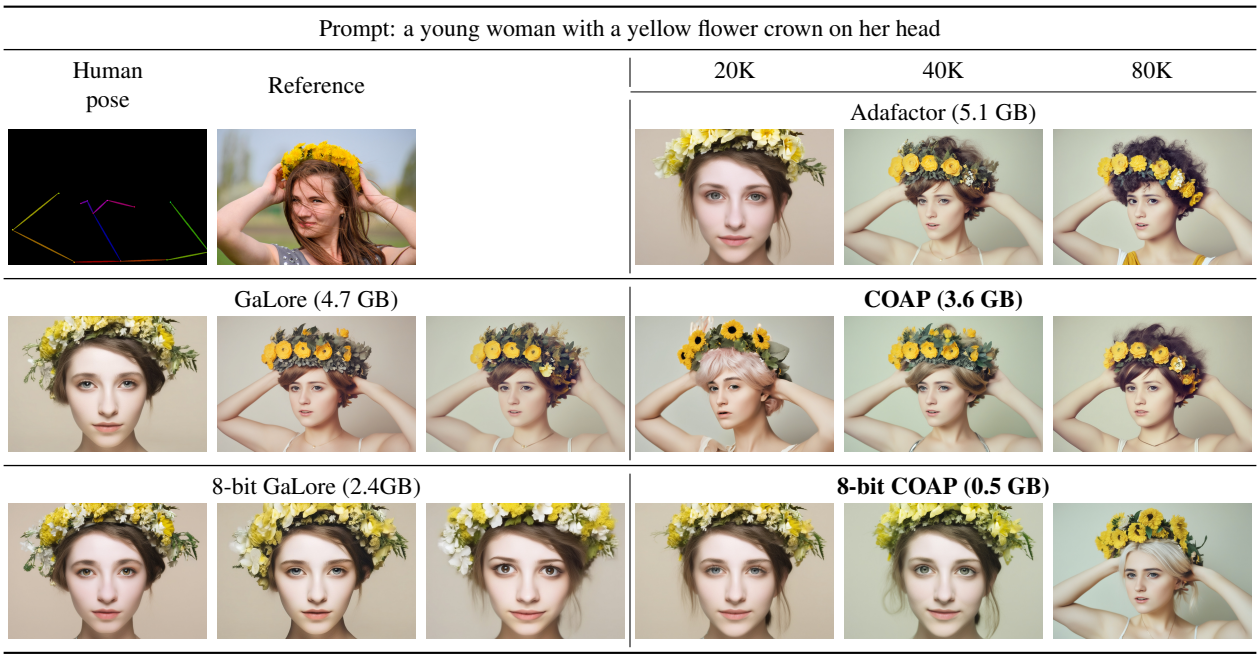
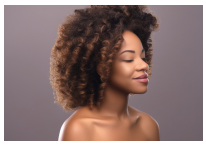
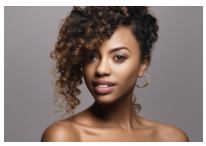
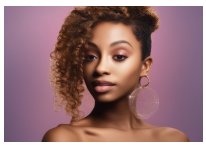
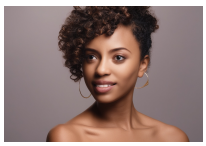
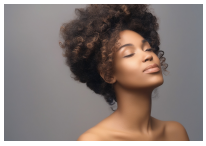


Table 7. Comparison of images generated at different training steps (20K, 40K, 80K) with ControlNet-XL trained under various optimizers. DDIM with a guidance scale of 5.0 is applied for image generation, with the number of inference steps set to 50.

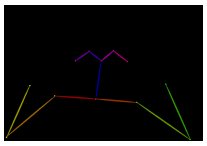


Prompt: beautiful african american woman with curly hair on blue background

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

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)**	
					

<table>
<tr><td colspan="5">Prompt: a young girl with her hands painted with colorful paint</td></tr>
<tr><td>Human pose</td><td>Reference</td><td colspan="3"></td></tr>
<tr><td>  </td><td>  </td><td colspan="3"></td></tr>
<tr><td colspan="2"></td><td>20K</td><td>40K</td><td>80K</td></tr>
<tr><td colspan="2"></td><td colspan="3">Adafactor (5.1 GB)</td></tr>
<tr><td colspan="2"></td><td>  </td><td>  </td><td>  </td></tr>
<tr><td colspan="2">GaLore (4.7 GB)</td><td colspan="3">COAP (3.6 GB)</td></tr>
<tr><td>  </td><td>  </td><td>  </td><td>  </td><td>  </td><td>  </td></tr>
<tr><td colspan="2">8-bit GaLore (2.4GB)</td><td colspan="3">8-bit COAP (0.5 GB)</td></tr>
<tr><td>  </td><td>  </td><td>  </td><td>  </td><td>  </td><td>  </td></tr>
</table>

References

[1] 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. 3

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 4

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

[4] 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. 3

[5] 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. 3

[6] 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. 5

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

[8] Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. Moe-i^2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition, 2024. 4

[9] Yimeng Zhang, Xin Chen, Jinghan Jia, Yihua Zhang, Chongyu Fan, Jiancheng Liu, Mingyi Hong, Ke Ding, and Sijia Liu. Defensive unlearning with adversarial training for robust concept erasure in diffusion models. *Advances in Neural Information Processing Systems*, 37:36748–36776, 2024.

[10] Yimeng Zhang, Jinghan Jia, Xin Chen, Aochuan Chen, Yihua Zhang, Jiancheng Liu, Ke Ding, and Sijia Liu. To generate or not? safety-driven unlearned diffusion models are still easy to generate unsafe images... for now. In *European Conference on Computer Vision*, pages 385–403. Springer, 2024.

[11] Yimeng Zhang, Tiancheng Zhi, Jing Liu, Shen Sang, Liming Jiang, Qing Yan, Sijia Liu, and Linjie Luo. Id-patch: Robust id association for group photo personalization. *arXiv preprint arXiv:2411.13632*, 2024. 4