Supplemental Material
𝟙LoRA: Summation Compression for Very Low-Rank Adaptation

A. Algorithm

As stated in the main paper, 𝟙LoRA is straightforward to implement. We illustrate the python implementation in Listing 1. In our work, we applied the 𝟙LoRA module to all linear layers in the model, excluding the classification layers, when present.

```python
import torch
import torch.nn as nn

class ILoRA(nn.Module):
    def __init__(self, linear):
        super().__init__()
        self.linear = linear
        self.b = nn.Parameter(torch.zeros(self.
    linear.out_features))

    def forward(self, x):
        ilora = self.b * x.sum(-1, keepdim=True)
        return self.linear(x) + ilora
```

Listing 1. Implementation of 𝟙LoRA in Python

B. Considered PEFT methods

We provide the technical details of state-of-the-art PEFT methods, analyzing in particular their computational and memory requirements.

LoRA [13] was one of the first methods proposed for parameter efficient fine-tuning. Instead of fine-tuning the entire parameter matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA fine-tunes an additive shift matrix $\Delta W \in \mathbb{R}^{d\times k}$ as follows:

$$o = W_0 x + \underline{\Delta W} x = (W_0 + \underline{\Delta W})x \tag{4}$$

where $x \in \mathbb{R}^k$ and $o \in \mathbb{R}^d$ are input and output features, respectively. We underline symbols indicating trainable parameters to distinguish them from pretrained, frozen parameters. LoRA decomposes the shift matrix ΔW by two rectangular matrices:

$$\Delta W = \underline{BA}, \quad \text{with } A \in \mathbb{R}^{r\times k}, B \in \mathbb{R}^{d\times r}. \tag{5}$$

Matrix A reduces the rank of input from k to r and matrix B decompresses it to the size of output d.

DoRA (weight-decomposed low-rank adaptation) [23] builds on LoRA and proposes to decompose the pretrained matrices into magnitude $m \in \mathbb{R}^d$ and direction $(W_0 + \underline{BA})/(||W_0 + \underline{BA}||_c)$. Here, the norm is defined as vector-wise norm of a matrix across each column.

VeRA (Vector-based Random-matrix Adaptation) [17] builds on top of LoRA using fixed random compression and decompression matrices (A and B), where the only trainable parameters are two vectors ($d \in \mathbb{R}^r$ and $b \in \mathbb{R}^d$), respectively after the compression and decompression.

MoRA [14], similar to VeRA, also uses fixed compression f_c and decompression f_d schemes, while only learning the inner low-rank squared matrix $M \in \mathbb{R}^{\hat{r}\times\hat{r}}$, where the rank is $\hat{r} = \lfloor\sqrt{(k+d)r}\rfloor$. The authors of [14] propose two versions, named types 1 and 6 in their Python code[2]. The version 'type 1 (Sharing)' is according to Eq. 6 in [14] and uses a sum within groups of features as compression and the decompression is a copy of learned features. The 'type 6 (RoPE based)' is a version according to Eq. 9 in [14] where the features are reshaped into a new axis as compression, processed, and then reshaped back. We name these methods MoRA_1 and MoRA_6, respectively.

BitFit [46] is one of the first PEFT methods, proposing to fine-tune only the biases $\beta \in \mathbb{R}^d$ of a pre-trained model.

DiffFit [41] extends BitFit to diffusion models by introducing a scaling vector $\gamma \in \mathbb{R}^d$. As a result, this approach adds an additional parameter that requires fine-tuning. Moreover, DiffFit trains the normalization layers and the class embedding, if present in the model.

In this paper, we explore the "very low-rank regime", where we use the smallest amount of parameters for each state-of-the-art method. We notice that BitFit [46] and VeRA [17] are the methods that use the least parameters in this scenario, with d and $1+d$ per layer, when the rank is $r = 1$. All other methods use more parameters. VeRA smartly shares the constant random matrices A and B across modules with the same shape to reduce computation, however, it still uses considerable memory for matrix storage. BitFit, on the other hand, only fine-tunes the bias terms that remain constant across inputs and therefore lack generalization capabilities. Furthermore, we notice that MoRA [14] has memory and computational inefficiencies due to the grouping and duplication operations, whereas other state-of-the-art LoRA methods heavily rely on the decomposition matrices A and B, which requires them to use additional memory.

B.1. 𝟙LoRA vs LoRA with rank 1

For 𝟙LoRA, inspired by MoRA, we use a fixed summation compression, making the method closely related to VeRA and LoRA with rank 1. The summation vector $\mathbb{1}$ is also the

[2] https://github.com/kongds/MoRA/

reason behind the method name (𝟙LoRA). On one hand, VeRA uses fixed random compression and decompression matrices and only learns 2 scaling vectors (after encoding and decoding, respectively), on the other hand, LoRA with rank 1 learns the compression vector, which we replace by a vector of ones (or with a simple sum in the implementation). We empirically show that learning this additional compression vector leads to higher memory and computational budgets, while only bringing minor improvements in terms of quality (or no improvements in some cases). Our PCA analysis (Sec. 5.2) shows that a summation vector (𝟙LoRA) after ReLUs has higher correlation to the full update than a random vector, and similar or better correlation than a learned vector (LoRA with rank 1). For other layers the 3 vectors are similarly low-correlated to the full update. Therefore, using 𝟙LoRA achieves similar results to LoRA with rank 1, while being more efficient in terms of parameters count ($\sim 2\times$), memory ($\sim 1.2\times$) and compute time (up to $\sim 4\times$).

C. Ablations

C.1. Complementary methods

We investigate all possible combinations of BitFit, DiffFit and 𝟙LoRA, including individual components (the biases β, the scaling factors γ, and the norms, see Table 1), in order to understand which modules impact fine-tuning most. We report experiments for MDE, fine-tuning DepthAnything from KITTI to NYU.

Interestingly, Figure 10 shows that, for the MDE case, only fine-tuning the normalization layers (norms, ●) leads to suboptimal results, whereas combining it with 𝟙LoRA ● allows it to achieve similar results as 𝟙LoRA ●, but faster. Fine-tuning the biases alone (i.e. BitFit, ●) achieves below state-of-the-art performance, but combining it with 𝟙LoRA ● has the same effect as combining norms with 𝟙LoRA ●. Other combinations seem to lead to no additional benefit, and only slow down the fine-tuning process. In general, any combination also increases the amount of trainable parameters, and 𝟙LoRA already achieves the best results while using the least number of trainable parameters in this experiment.

C.2. Additional experiments

Classification We report classification results on the more challenging Stanford Cars [18] and FGVC-Aircraft [25] datasets (Fig. 11). The figures show that on both datasets 𝟙LoRA is still among the best methods, while 𝟙LoRA (norms), i.e. 𝟙LoRA combined with the fine-tuning of normalization layers, achieves the best results among the tested peft methods.

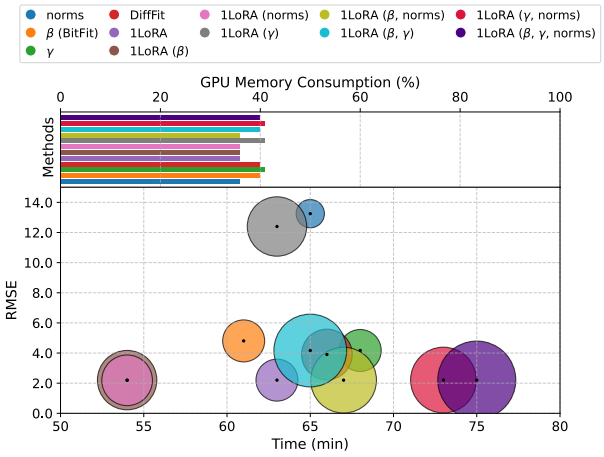

Figure 10. RMSE (↓) of DepthAnything model pre-trained on KITTI and fine-tuned to NYU, using all combinations of 𝟙LoRA, BitFit and DiffFit, including individual components (biases β, scaling factors γ and normalization layers $norms$). Bubble size is proportional to the number of parameters. Bottom left (and smallest bubble) is better.

Image generation We report image generation results on the more challenging DomainNet - Sketch [32] dataset (Fig. 12). The figure shows 𝟙LoRA is still among the best methods, on par with DiffFit which achieves similar performance slightly faster, but using more parameters and memory. Note that VeRA achieves high FID (~ 46) and it's not included in the plot to better highlight differences among other methods.

C.3. Higher ranks

We report results for higher ranks for LoRA, VeRA and DoRA on the more challenging Stanford Cars and FGVC-Aircraft datasets (Fig. 13). Unsurprisingly, it emerges that increasing the rank slightly increases the performance, while also slightly increasing the memory budget. All methods can outperform 𝟙LoRA, when increasing the rank and disregarding the compute and memory budgets, however only LoRA with rank 32 can match 𝟙LoRA's performance when given the same compute budget (and higher memory budget). Notably, none of the reported ranks can outperform 𝟙LoRA (norms), especially within its compute budget. We additionally tested LoRA with rank 64 and 128 to understand what rank would be required by LoRA to match or surpass 𝟙LoRA (norms)'s performance within the same compute budget. On Stanford Cars we have that LoRA with rank 64 achieves $\sim 63\%$ accuracy and with rank 128 achieves $\sim 72\%$ accuracy, i.e. one slightly worse and one better than 𝟙LoRA (norms, $\sim 65\%$), but only when granted more compute. Therefore, ranks higher than 128 need to be

considered to match 𝟙LoRA (norms) within the same compute budget. On FGVC Aircraft, LoRA with rank 64 and 128 achieve $\sim$ 58% and $\sim$ 62%, respectively. Both surpass 𝟙LoRA (norms, $\sim$ 51%), but only LoRA with rank 128 can surpass 𝟙LoRA (norms) within the same compute budget, achieving $\sim$ 54 accuracy. Overall, these results suggest that higher ranks are needed for LoRA to match or surpass 𝟙LoRA and 𝟙LoRA (norms)'s performances within the same compute budget.

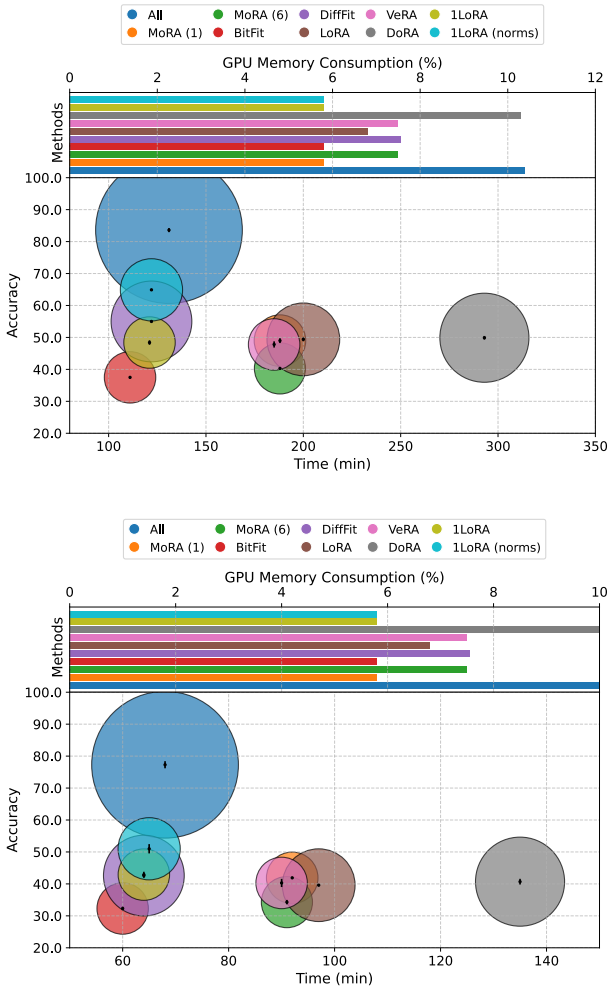

Figure 11. Accuracy (↑) of ViT-Base model pre-trained on ImageNet-21k and fine-tuned to Stanford Cars (top), to FGVC Aircraft (bottom). Bubble size is proportional to the number of parameters, except for "All", which is capped. Top left (and smallest bubble) is better.

C.4. Summation vs random compression

Here, we report ablation on the MDE (Table S1) taskcomparing 𝟙LoRA with fixed summation and random compression. Results show that summation is overall slightly better

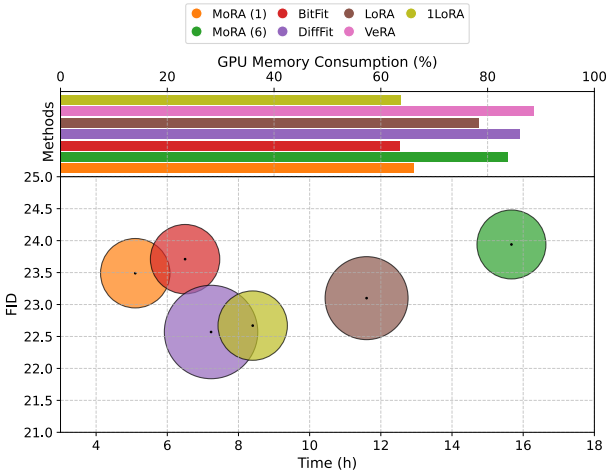

Figure 12. FID of pretrained DiT fine-tuned to DomainNet - Sketch. Note that DoRA is missing as it cannot be fine-tuned with the given memory budget. The figure is zoomed to show main baselines: VeRA achieves $\sim$ 46 and lands outside the zoomed area. Bubble size is proportional to the number of parameters. Bottom left (and smallest bubble) is better.

in terms of performance or required time, suggesting that using a fixed summation compression is at least as good as using random compression.

Table S1. (MDE) 𝟙LoRA: summation vs random compression.

dataset	compression	time [min] (↓)	RMSE (↓)
NYU → KITTI	summation	**63**	**2.203±0.0078**
	random	65	2.239 ± 0.0089
KITTI → NYU	summation	**65**	**0.238±0.0005**
	random	67	0.24 ± 0.0002
KITTI → DIODE Out	summation	**13**	**4.574±0.1217**
	random	22	4.573 ± 0.0582
NYU → DIODE Out	summation	42	**4.719 ± 0.06**
	random	42	4.725 ± 0.06

C.5. Analysis of edge cases

To explore edge cases, we analyzed the error distribution and calibration of full fine-tuning, LoRA $r = 1$ and 𝟙LoRA for our classification experiments (Fig. 14). Contrary to expectation, full fine-tuning seems to be more heavy-tailed in terms of error distribution than the very low-rank PEFT methods. As for the calibration, again the very low-rank PEFT methods seem to slightly improve above full fine-tuning, though the improvement – if any – is very much negligible.

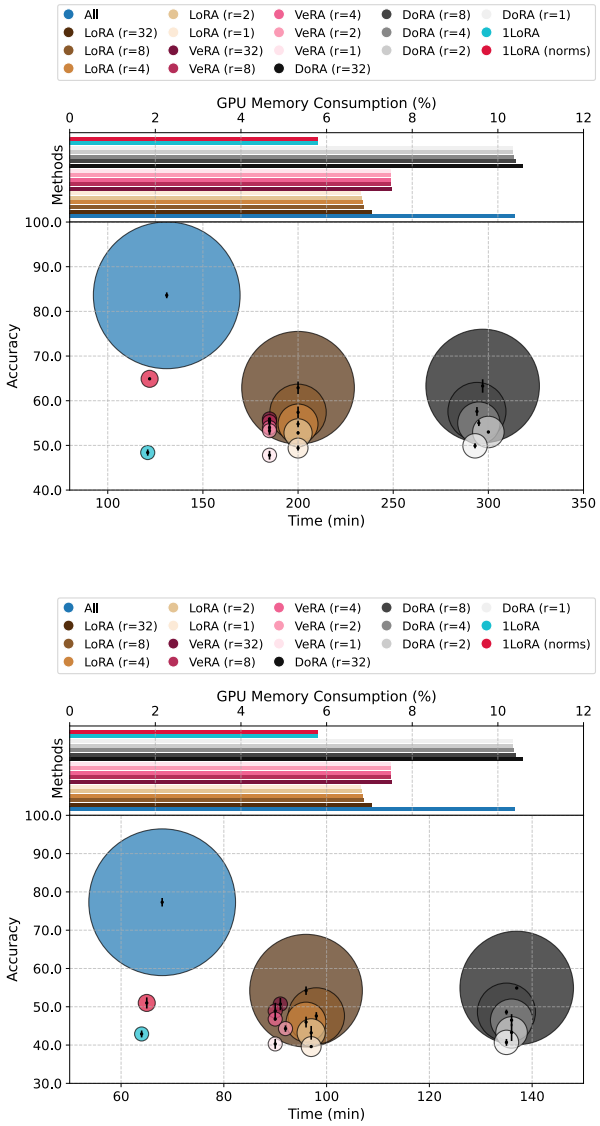

Figure 13. Accuracy (↑) of ViT-Base model pre-trained on ImageNet-21k and fine-tuned with higher ranks for LoRA, VeRA and DoRA, to Stanford Cars (top), to FGVC Aircraft (bottom). Bubble size is proportional to the number of parameters, except for "All", which is capped. Top left (and smallest bubble) is better.

D. Mathematical reasoning visualization

We report visualization for LLaMA2 13b finetuned on 80% MetaMath, comparing 𝟙LoRA and LoRA QKV. We use random samples from our generated validation split (20%) of MetaMath. We use black, olive and cyan boxes to highlight the input prompt (question and answer, where only the question is given as input and the answer serves as ground truth), the 𝟙LoRA and LoRA QKV answers, respectively. Note that we omit "Below is an instruction that describes a task. Write a response that appropriately completes the request." from Q & A boxes due to space limitations.

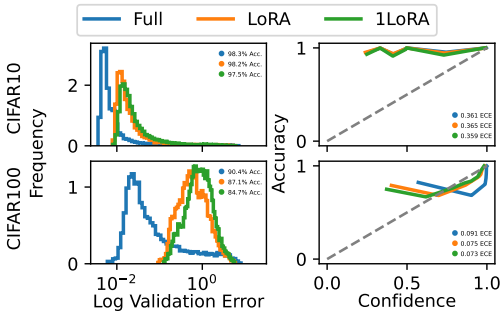

Figure 14. Error distribution and calibration for classification examples.

Question & Answer

Instruction:
On Friday, a snowboard originally priced at $100 was discounted 50%. On Monday, that sale price was reduced by 30%. In dollars, what is the price of the snowboard after the Monday reduction?
Response:
The Friday sale price was $100 × 0.50 = $50. The Monday reduction is 30% of $50, so it is $50 × 0.30 = $15. Therefore, the price of the snowboard after the Monday reduction is $50−$15 = $\boxed{\$35}$.The answer is: 35.

𝟙LoRA

The Friday discount of 50% reduces the price to 100−0.5(100) = 50. The Monday reduction of 30% reduces the price further to 50−0.3(50) = $\boxed{35}$. The answer is: 35

LoRA QKV

The snowboard was originally priced at $100. The 50% discount brings the price down to $100 · 0.5 = $50. The 30% reduction brings the price down further to $50 · 0.7 = $35. Therefore, the price of the snowboard after the Monday reduction is $\boxed{\$35}$.The answer is: 35

Question & Answer

Instruction:
George went to a movie theater to see a movie. He paid $16 for the ticket and decided to buy some nachos. The nachos were half the price of the ticket. How much did George pay in total for the visit to the movie theater?
Response:
George paid $16 for the ticket. The nachos were half the price of the ticket, so they cost $16/2 = $8. In total, George paid $16 + $8 = $24 for the visit to the movie theater.

𝟙LoRA

The nachos were half the price of the ticket, so they cost $16/2 = $8. Therefore, George paid a total of $16 + $8 = $24 for the visit to the movie theater.
24
The answer is: 24

LoRA QKV

The nachos were half the price of the ticket, so they cost $16/2 = $8 In total, George paid $16 + $8 = $24 for the visit to the movie theater.
24
The answer is: 24

Question & Answer

Instruction:
If $ab + bc + cd + da = 30$ and $b + d = 5$, what is the value of $a + c$?
Response:
We can rewrite the given equation as $a(b+d)+c(b+d) = 30$. Using the second equation, we substitute $b + d = 5$ to get $5a + 5c = 30$. Dividing both sides by 5, we have $a + c = \boxed{6}$. The answer is: 6

𝟙LoRA

We can rewrite the equation $ab + bc + cd + da = 30$ as $(a + c)(b + d) = 30$. Substituting $b + d = 5$, we have $(a + c)(5) = 30$. Dividing both sides by 5, we get $a + c = \boxed{6}$. The answer is: 6

LoRA QKV

We can rewrite the given equation as $a(b+d)+c(b+d) = 30$. Using the given information that $b + d = 5$, we have $2ab + 2cd = 30$. We can rewrite this equation as $ab+cd = 15$. Since $ab+bc+cd+da = 30$, we can subtract $ab + cd$ from both sides to get $bc + da = 15$. We can also rewrite this equation as $bc + da = 15$. Since $b + d = 5$, we can substitute this into the equation to get $bc + da = 5$. Now, we can solve for $a + c$ by subtracting a and c from both sides: $a + c = 5$. Therefore, the value of $a + c$ is $\boxed{5}$.
The answer is: 5

Question & Answer

Instruction:
In 5 years, Raven will be 4 times as old as Phoebe. If Phoebe is currently x years old, Raven is 55 years old. What is the value of unknown variable x?
Response:
To solve this problem, we need to determine the value of x, which represents Phoebe's current age. Let's break down the information given: In 5 years, Raven will be 4 times as old as Phoebe. Raven is currently 55 years old. We can set up the equation as follows: Raven's age in 5 years = 4 * Phoebe's age in 5 years (Raven's current age + 5) = 4 * (Phoebe's current age + 5) 55 + 5 = 4 * (x + 5) 60 = 4 * (x + 5) Let's simplify and solve for x: 60 = 4x + 20 To isolate x, we subtract 20 from both sides of the equation: 60 - 20 = 4x + 20 - 20 40 = 4x To solve for x, we divide both sides of the equation by 4: 40 / 4 = 4x / 4 10 = x The value of x is 10.
10
The answer is: 10.

1LoRA

To solve this problem, we need to determine the value of x, which represents Phoebe's current age. We know that in 5 years, Raven will be 4 times as old as Phoebe. Therefore, Raven's age in 5 years will be 4 times Phoebe's age in 5 years. Let's break down the information given: Raven's current age: 55 years Phoebe's current age: x years In 5 years, Raven's age: 55 + 5 = 60 years In 5 years, Phoebe's age: x + 5 years We can set up the equation as follows: Raven's age in 5 years = 4 * Phoebe's age in 5 years 60 = 4 * (x + 5) Let's simplify and solve for x: 60 = 4x + 20 To isolate x, we subtract 20 from both sides of the equation: 60 - 20 = 4x + 20 - 20 40 = 4x To solve for x, we divide both sides of the equation by 4: 40 / 4 = 4x / 4 10 = x The value of x is 10.
10
The answer is: 10

LoRA QKV

To solve this problem, we need to determine the value of x, which represents Phoebe's current age. We know that in 5 years, Raven will be 4 times as old as Phoebe. Therefore, Raven will be 4x years old in 5 years. We also know that Raven is currently 55 years old. We can set up the equation as follows: Raven's age in 5 years = 4 * Phoebe's current age 55 = 4 * x Let's simplify and solve for x: 55 = 4x To isolate x, we divide both sides of the equation by 4: 55 / 4 = 4x / 4 14 = x The value of x is 14.
14
The answer is: 14

E. Detailed results

We report tables (from S2 to S12) illustrating the detailed results shown in the main paper as bubble and bar plots. For each column reporting a result, we highlight the best value in **bold**, the second best in underlined and the third in *italic*.

method	time [min] (↓)	RMSE (↓)	memory [%] (↓)	params (↓)
MoRA$_1$	*69*	*0.248 ± 0.00021*	**36**	220.06K
MoRA$_6$	73	0.277 ± 0.00066	**36**	**217.44K**
BitFit	**64**	0.349 ± 0.00073	40	*221.18K*
DiffFit	78	0.314 ± 0.00078	40	321.54K
LoRA	71	0.25 ± 0.00087	*41*	393.22K
VeRA	*69*	0.244 ± 0.00059	40	221.28K
DoRA	104	0.249 ± 0.00088	54	614.40K
𝟙LoRA	65	**0.238 ± 0.0005**	**36**	*221.18K*
All	83	0.211 ± 0.00064	56	335.32M

Table S2. Table for Figure 2a

method	time [min] (↓)	RMSE (↓)	memory [%] (↓)	params (↓)
MoRA$_1$	67	2.509 ± 0.00455	**36**	220.06K
MoRA$_6$	71	2.97 ± 0.00328	**36**	**217.44K**
BitFit	**61**	4.806 ± 0.00392	40	*221.18K*
DiffFit	*66*	3.904 ± 0.00539	40	321.54K
LoRA	*66*	2.253 ± 0.00315	*41*	393.22K
VeRA	68	*2.304 ± 0.00524*	40	221.28K
DoRA	100	2.346 ± 0.00275	54	614.40K
𝟙LoRA	63	**2.203 ± 0.0078**	**36**	*221.18K*
All	85	1.916 ± 0.00303	56	335.32M

Table S3. Table for Figure 2b

method	time [min] (↓)	RMSE (↓)	memory [%] (↓)	params (↓)
MoRA_1	52	4.704 ± 0.01546	**36**	220.06K
MoRA_6	14	*4.637 ± 0.00449*	**36**	**217.44K**
BitFit	47	5.158 ± 0.00868	40	*221.18K*
DiffFit	50	4.955 ± 0.0064	40	321.54K
LoRA	51	4.612 ± 0.00368	*41*	393.22K
VeRA	*24*	4.797 ± 0.0181	40	221.28K
DoRA	74	4.644 ± 0.00405	54	614.40K
𝟙LoRA	**13**	**4.574 ± 0.1217**	**36**	*221.18K*
All	55	4.59 ± 0.0784	56	335.32M

Table S4. Table for Figure 2c

method	time [min] (↓)	RMSE (↓)	memory [%] (↓)	params (↓)
MoRA_1	51	5.067 ± 0.02273	**36**	220.06K
MoRA_6	54	6.242 ± 0.0141	**36**	**217.44K**
BitFit	47	7.698 ± 0.00293	40	*221.18K*
DiffFit	*50*	7.017 ± 0.00998	40	321.54K
LoRA	67	4.94 ± 0.00776	*41*	393.22K
VeRA	51	4.839 ± 0.01025	40	221.28K
DoRA	75	*4.918 ± 0.00949*	54	614.40K
𝟙LoRA	**42**	**4.719 ± 0.066**	**36**	*221.18K*
All	61	4.53 ± 0.0278	56	335.32M

Table S5. Table for Figure 2d

method	time [min] (↓)	Loss (↓)	memory [%] (↓)	params (↓)
MoRA_1	187	0.18 ± 0.000724	80	1.36M
MoRA_6	198	0.19 ± 6.6e-05	91	1.35M
BitFit	**130**	0.22 ± 0.0047	**75**	*1.36M*
DiffFit	132	0.21 ± 0.00411	*77*	1.66M
LoRA	154	**0.15 ± 0.00202**	93	2.50M
VeRA	159	*0.176 ± 0.00192*	92	1.36M
𝟙LoRA	152	0.17 ± 0.003768	76	*1.36M*
LoRA (QKV)	*134*	0.2 ± 0.002273	79	**786.43K**

Table S6. Table for Figure 3

method	time [min] (↓)	Loss (↓)	memory [%] (↓)	params (↓)
BitFit	66	0.19 ± 0.001738	**91**	2.13M
DiffFit	*71*	*0.18 ± 0.003334*	*93*	*2.58M*
𝟙LoRA	77	**0.14 ± 0.00152**	92	2.13M
LoRA (QKV)	**61**	0.17 ± 0.005107	92	**1.23M**

Table S7. Table for Figure 4

method	time [h] (↓)	FID (↓)	memory [%] (↓)	params (↓)
MoRA_1	**9.6**	13.9004	*67*	604.41K
MoRA_6	23.5	14.0791	87	**596.52K**
BitFit	*13.0*	13.8074	66	603.68K
DiffFit	14.3	*13.6644*	87	1.09M
LoRA	19.5	13.7621	79	864.03K
VeRA	12.0	13.1129	87	*603.82K*
𝟙LoRA	17.0	**12.8218**	**65**	603.68K

Table S8. Table for Figure 5

method	time [min] (↓)	Accuracy (↑)	memory [%] (↓)	params (↓)
MoRA_1	30	96.55 ± 0.0465	**6**	234.06K
MoRA_6	31	97.5 ± 0.0544	**6**	**232.76K**
BitFit	**16**	97.0 ± 0.055	**6**	233.67K
DiffFit	*18*	*97.9 ± 0.0585*	*8*	355.21K
LoRA	33	98.1 ± 0.058	7	316.61K
VeRA	29	97.7 ± 0.1631	*8*	*233.64K*
DoRA	51	**98.2 ± 0.0866**	11	399.56K
𝟙LoRA	17	97.7 ± 0.186	**6**	233.67K
𝟙LoRA (norms)	*18*	*98.0 ± 0.1397*	**6**	272.07K
All	22	98.3 ± 0.0755	10	86M

Table S9. Table for Figure 6

method	time [min] (↓)	Accuracy (↑)	memory [%] (↓)	params (↓)
MoRA_1	31	82.7 ± 0.3253	**6**	234.06K
MoRA_6	30	81.2 ± 0.1875	**6**	**232.76K**
BitFit	**16**	83.0 ± 0.204	**6**	233.67K
DiffFit	*18*	*87.9 ± 0.0775*	*8*	355.21K
LoRA	34	**88.4 ± 0.2173**	7	316.61K
VeRA	29	85.3 ± 0.1723	*8*	*233.74K*
DoRA	50	87.3 ± 0.2245	11	399.56K
𝟙LoRA	17	85.0 ± 0.567	**6**	233.67K
𝟙LoRA (norms)	*18*	88.2 ± 0.314	**6**	272.07K
All	22	90.9 ± 0.3251	10	86M

Table S10. Table for Figure 7

method	time [min] (↓)	RMSE (↓)	memory [%] (↓)	params (↓)
norms	65	13.25 ± 0.004751	**36**	**100.35K**
β (BitFit)	61	4.806 ± 0.003922	40	221.18K
γ	68	4.17 ± 0.00482	*41*	221.18K
DiffFit	66	3.904 ± 0.00539	40	*321.54K*
𝟙LoRA	*63*	*2.203 ± 0.0078*	**36**	221.18K
𝟙LoRA (β)	**54**	*2.203 ± 0.005498*	**36**	442.37K
𝟙LoRA (norms)	**54**	**2.19 ± 0.00655**	**36**	*321.54K*
𝟙LoRA (γ)	*63*	12.399 ± 0.006093	*41*	442.37K
𝟙LoRA (β, norms)	67	2.2 ± 0.0051438	**36**	542.72K
𝟙LoRA (β, γ)	65	4.166 ± 0.008069	40	663.55K
𝟙LoRA (γ, norms)	73	2.206 ± 0.001668	*41*	542.72K
𝟙LoRA (β, γ, norms)	75	2.206 ± 0.002445	40	763.90K

Table S11. Table for Figure 10

method	time [min] (↓)	Accuracy (↑)	memory [%] (↓)	params (↓)
norms	**16**	97.7 ± 0.0395	**6**	**189.12K**
β (BitFit)	**16**	97.0 ± 0.055	**6**	233.67K
γ	17	97.3 ± 0.0457	8	316.81K
DiffFit	*18*	97.9 ± 0.0585	8	355.21K
𝟙LoRA	17	97.7 ± 0.186	**6**	233.67K
𝟙LoRA (β)	*18*	*97.8 ± 0.1245*	**6**	316.51K
𝟙LoRA (norms)	*18*	**98.0 ± 0.139**	**6**	*272.07K*
𝟙LoRA (γ)	19	*97.8 ± 0.1043*	8	317K
𝟙LoRA (β, norms)	19	97.9 ± 0.078	**6**	355.01K
𝟙LoRA (β, γ)	21	*97.8 ± 0.1297*	**6**	399.95K
𝟙LoRA (γ, norms)	20	**98.0 ± 0.0837**	8	355.4K
𝟙LoRA (β, γ, norms)	22	**98.0 ± 0.109**	8	438.35K

Table S12. Table for Figure 8