SVFT: Parameter-Efficient Fine-Tuning with Singular Vectors

Vijay Lingam[†][§][*] **Atula Tejaswi**[†][*] **Aditya Vavre**[†][*] **Aneesh Shetty**[†][*]
Gautham Krishna Gudur[†][*] **Joydeep Ghosh**[†] **Alex Dimakis**[†] **Eunsol Choi**[†]
Aleksandar Bojchevski[‡][*] **Sujay Sanghavi**[†][*]
[†]University of Texas at Austin [‡]University of Cologne
§CISPA Helmholtz Center for Information Security

Abstract

Popular parameter-efficient fine-tuning (PEFT) methods, such as LoRA and its variants, freeze pre-trained model weights $\mathbf{W}$ and inject learnable matrices $\mathbf{\Delta W}$. These $\mathbf{\Delta W}$ matrices are structured for efficient parameterization, often using techniques like low-rank approximations or scaling vectors. However, these methods typically show a performance gap compared to full fine-tuning. Although recent PEFT methods have narrowed this gap, they do so at the cost of additional learnable parameters. We propose SVFT[2], which enables a trade-off between the number of trainable parameters and model expressivity by allowing a flexible number of off-diagonal interactions between singular vectors in $\mathbf{\Delta W}$, distinguishing it from previous SVD-based methods. This approach provides fine-grained control over expressivity through the number of coefficients. Extensive experiments on language and vision benchmarks demonstrate that SVFT recovers up to **96%** of full fine-tuning performance while training only **0.006 to 0.25**% of parameters, outperforming existing methods that recover only up to **85%** performance using **0.03 to 0.8**% of the trainable parameter budget.

1 Introduction

Large-scale foundation models are often adapted for specific downstream tasks after pre-training. Parameter-efficient fine-tuning (PEFT) facilitates this adaptation efficiently by learning a minimal set of new parameters, thus creating an "expert" model. For instance, Large Language Models (LLMs) pre-trained on vast training corpora are fine-tuned for specialized tasks such as text summarization [13, 37], sentiment analysis [27, 21], and code completion [28] using instruction fine-tuning datasets. Although full fine-tuning (Full-FT) is a viable method to achieve this, it requires re-training and storing all model weights, making it impractical for deployment with large foundation models.

To address these challenges, PEFT techniques [14] (e.g., LoRA [15]) were introduced to significantly reduce the number of learnable parameters compared to Full-FT, though often at the cost of performance. DoRA [19] bridges this performance gap by adding more learnable parameters and being more expressive than LoRA. Almost all these methods apply a low-rank update additively to the frozen pre-trained weights, potentially limiting their expressivity. Furthermore, these adapters are agnostic to the structure and geometry of the weight matrices they modify. Finally, more expressive PEFT methods (e.g., LoRA, DoRA, BOFT [20]) still accumulate a considerable portion of learnable parameters even in their most efficient configuration (e.g., setting rank=1 in LoRA and DoRA). The

[*]indicates equal contribution/advising
[2]Code is available at `https://github.com/VijayLingam95/SVFT/`

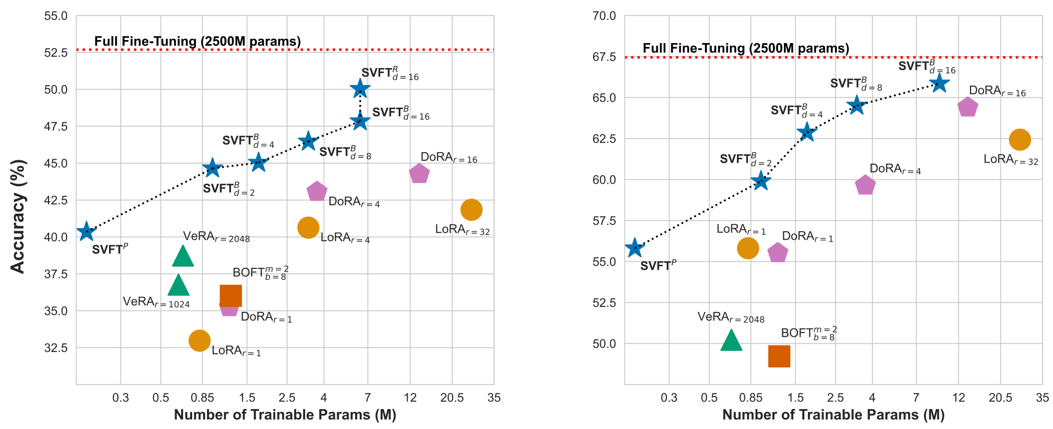


Figure 1: Performance vs total trainable parameters for GSM-8K (left) and Commonsense Reasoning (right) on Gemma-2B. SVFT$^{B/R}_{d=16}$ outperforms DoRA$_{r=8/16}$ with 75% less trainable parameters.

storage requirements for the learnable adapters can grow very quickly when adapting to a large number of downstream tasks [17].

Is it possible to narrow the performance gap between PEFT and Full-FT while being highly parameter-efficient? We propose SVFT: Singular Vectors guided Fine-Tuning — a *simple* approach that involves updating an existing weight matrix by adding to it a sparse weighted combination of *its own singular vectors*. The structure of the induced perturbation in SVFT depends on the specific matrix being perturbed. Our contributions can be summarized as follows:

- We introduce SVFT, a new PEFT method. Given a weight matrix $\boldsymbol{W}$, SVFT involves adapting it with a matrix $\Delta\boldsymbol{W} := \sum_{(i,j)\in\Omega} m_{ij}\boldsymbol{u}_i\boldsymbol{v}_j^T$ where the $\{\boldsymbol{u}_i\}$ and $\{\boldsymbol{v}_j\}$ are the left and right singular vectors of $\boldsymbol{W}$, Ω is an a-priori fixed sparsity pattern, and m_{ij} for $(i,j)\in\Omega$ are learnable parameters. By controlling $|\Omega|$ we can efficiently explore the accuracy vs parameters trade-off.
- SVFT achieves higher downstream accuracy, as a function of the number of trainable parameters, as compared to several popular PEFT methods (see Figure 1) and over several downstream tasks across both vision and language tasks. Our method recovers up to **96%** of full fine-tuning performance while training only **0.006 to 0.25**% of parameters, outperforming existing methods that only recover up to **85%** performance using **0.03 to 0.8%** the trainable parameter budget.

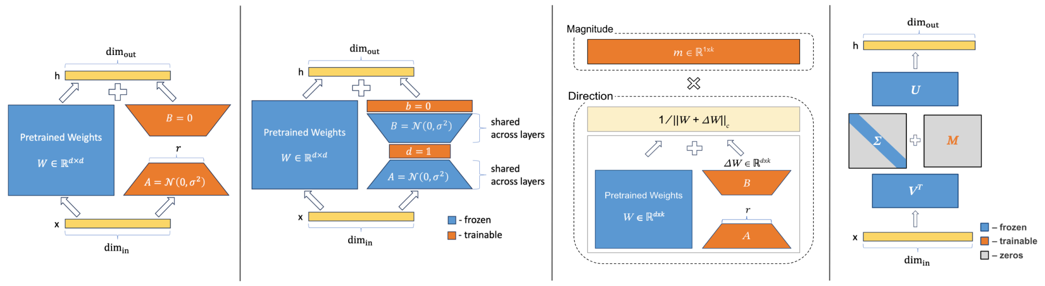
We introduce three variants for parameterizing weight updates, namely: *Plain*, *Random*, and *Banded* in SVFT (which differ in their choices of the fixed sparsity pattern Ω) and validate these design choices empirically. Additionally, we theoretically show that for any fixed parameters budget, SVFT can induce a higher rank perturbation compared to previous PEFT techniques.

2 Related Work

Recent advancements in large language models (LLMs) have emphasized the development of PEFT techniques to enhance the adaptability and efficiency of large pre-trained language models.

LoRA. A notable contribution in this field is Low-Rank Adaptation (LoRA) [15], which freezes the weights of pre-trained models and integrates trainable low-rank matrices into each transformer layer. For a pre-trained weight matrix $\boldsymbol{W}_0 \in \mathbb{R}^{d\times n}$, LoRA constrains the weight update $\Delta\boldsymbol{W}$ to a low-rank decomposition: $\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \Delta\boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}_0\boldsymbol{x} + \underline{\boldsymbol{B}\boldsymbol{A}}\boldsymbol{x}$, where $\boldsymbol{B}\in\mathbb{R}^{d\times r}$, $\boldsymbol{A}\in\mathbb{R}^{r\times n}$ and rank $r \ll \min(d,n)$. We underline the (trainable) parameters that are updated via gradient descent.

LoRA variants. We highlight some recent approaches that further improve the vanilla LoRA architecture. Vector-based Random Matrix Adaptation (VeRA) [17] minimizes the number of trainable parameters by utilizing a pair of low-rank random matrices shared between layers and learning compact scaling vectors while maintaining performance comparable to LoRA. Formally, VeRA can be expressed as: $\boldsymbol{h} = \boldsymbol{W}_0\boldsymbol{x} + \Delta\boldsymbol{W}\boldsymbol{x} = \boldsymbol{W}_0\boldsymbol{x} + \underline{\boldsymbol{\Lambda}_b}\boldsymbol{B}\underline{\boldsymbol{\Lambda}_d}\boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ are initialized randomly, frozen, and shared across layers, while $\boldsymbol{\Lambda}_b$ and $\boldsymbol{\Lambda}_d$ are trainable diagonal matrices.



dim_{out}
h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
x
dim_{in}
dim_{out}
h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$b = 0$
$B = \mathcal{N}(0, \sigma^2)$
shared across layers
$d = 1$
$A = \mathcal{N}(0, \sigma^2)$
shared across layers
x
dim_{in}
🟦 - frozen
🟧 - trainable
Magnitude
$m \in \mathbb{R}^{1 \times k}$
Direction
$1/\|W + \Delta W\|_c$
$\Delta W \in \mathbb{R}^{d \times k}$
B
r
A
Pretrained Weights
$W \in \mathbb{R}^{d \times k}$
dim_{out}
h
U
Σ
M
V^T
x
dim_{in}
🟦 – frozen
🟧 – trainable
⬜ – zeros

Figure 2: Schematic comparison of LoRA, VeRA, DoRA, and SVFT (left to right).

An alternative approach, Weight-Decomposed Low-Rank Adaptation (DoRA) [19], decomposes pre-trained weight matrices into magnitude and direction components, and applies low-rank updates for directional updates, reducing trainable parameters and enhancing learning capacity and training stability. DoRA can be expressed as: $\boldsymbol{h} = \underline{\boldsymbol{m}} \frac{\boldsymbol{W}_0 + \Delta \boldsymbol{W}}{\|\boldsymbol{W}_0 + \Delta \boldsymbol{W}\|_c} \boldsymbol{x} = \underline{\boldsymbol{m}} \frac{\boldsymbol{W}_0 + \underline{\boldsymbol{BA}}}{\|\boldsymbol{W}_0 + \underline{\boldsymbol{BA}}\|_c} \boldsymbol{x}$, where $\| \cdot \|_c$ denotes the vector-wise norm of a matrix across each column. Similar to LoRA, $\boldsymbol{W}_0$ remains frozen, whereas the magnitude vector $\boldsymbol{m}$ (initialized to $\|\boldsymbol{W}_0\|_c$) and low-rank matrices $\boldsymbol{A}, \boldsymbol{B}$ contain trainable parameters.

AdaLoRA [38] adaptively distributes the parameter budget across weight matrices based on their importance scores and modulates the rank of incremental matrices to manage this allocation effectively. PiSSA (Principal Singular Values and Singular Vectors Adaptation) [22] is another variant of LoRA, where matrices $\boldsymbol{A}, \boldsymbol{B}$ are initialized with principal components of SVD and the remaining components are used to initialize $\boldsymbol{W}_0$. FLoRA [34] enhances LoRA by enabling each example in a mini-batch to utilize distinct low-rank weights, preserving expressive power and facilitating efficient batching, thereby extending the domain adaptation benefits of LoRA without batching limitations.

Other PEFT variants. Orthogonal Fine-tuning (OFT) [26] modifies pre-trained weight matrices through orthogonal reparameterization to preserve essential information. However, it still requires a considerable number of trainable parameters due to the high dimensionality of these matrices. Butterfly Orthogonal Fine-tuning (BOFT) [20] extends OFT's methodology by incorporating Butterfly factorization thereby positioning OFT as a special case of BOFT. Unlike the additive low-rank weight updates utilized in LoRA, BOFT applies multiplicative orthogonal weight updates, marking a significant divergence in the approach but claims to improve parameter efficiency and fine-tuning flexibility. BOFT can be formally expressed as: $\boldsymbol{h} = (\underline{\boldsymbol{R}(m, b)} \cdot \boldsymbol{W}_0)\boldsymbol{x}$, where the orthogonal matrix $\boldsymbol{R}(m, b) \in \mathbb{R}^{d \times d}$ is composed of a product of multiple orthogonal butterfly components. When $m = 1$, BOFT reduces to block-diagonal OFT with block size b. When $m = 1$ and $b = d$, BOFT reduces to the original OFT with an unconstrained full orthogonal matrix.

SVD-based variants. SVF [31], SVDiff [10], and SAM-Parser [25] also consider the structure of W matrices and decompose them into three successive matrices using Singular Value Decomposition (SVD). Both these methods fine-tune only the singular values while keeping the other parameters frozen. These methods are equivalent to SVFT^P.

3 Method

In this section, we introduce Singular Vectors guided Fine-Tuning (SVFT). The main innovation in SVFT lies in exploring off-diagonal interactions in the singular values spaces.

3.1 SVFT Formulation

We now formally describe our method, SVFT for parameter-efficient fine-tuning of a pre-trained model. Let $\boldsymbol{W}_0 \in \mathbb{R}^{d_1 \times d_2}$ denote a weight matrix in the pre-trained model. For instance, in a transformer block, this could be the key matrix, the query matrix, a matrix in the MLP, etc. We add a structured, learned $\Delta \boldsymbol{W}$ to this matrix as follows.

As a first step, we compute the Singular Value Decomposition (SVD) of the given matrix: $\boldsymbol{W}_0 = \boldsymbol{U\Sigma V}^T$. That is, $\boldsymbol{U}$ is the $d_1 \times d_1$ matrix of left singular vectors (i.e., its columns are orthonormal),

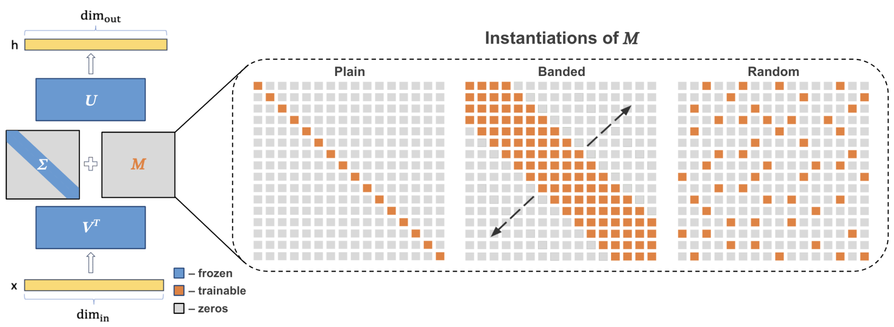


Figure 3: An Overview of SVFT. The original weights $\boldsymbol{W}$ are decomposed into $\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$. Here, $\boldsymbol{M}$ contains all the trainable parameters, which can be configured into patterns such as Plain, Random, and Banded, represented by patterns of trainable (orange) and zero (gray) elements.

$\boldsymbol{V}^T$ is the $d_2 \times d_2$ matrix of right singular vectors (i.e., its rows are orthonormal), and $\boldsymbol{\Sigma}$ is a $d_1 \times d_2$ diagonal matrix. Then, we parameterize our weight update as $\Delta \boldsymbol{W} = \boldsymbol{U}\underline{\boldsymbol{M}}\boldsymbol{V}^T$, where $\boldsymbol{U}, \boldsymbol{V}$ are fixed and frozen, while $\underline{\boldsymbol{M}}$ is a $d_1 \times d_2$ **sparse trainable matrix with pre-determined and fixed sparsity pattern**[3]. That is, we first pre-determine a small fixed set of elements in $\boldsymbol{M}$ that will be allowed to be non-zero and train only those elements. The forward pass for SVFT can be written as,

$$\boldsymbol{h} = \boldsymbol{W}_0 x + \Delta \boldsymbol{W} x = \boldsymbol{U}(\boldsymbol{\Sigma} + \underline{\boldsymbol{M}})\boldsymbol{V}^T \boldsymbol{x} \tag{1}$$

We explore three choices for Ω, the a-priori fixed sparsity pattern of $\underline{\boldsymbol{M}}$.
Plain $\left(\text{SVFT}^P\right)$. In this variant, we constrain $\underline{\boldsymbol{M}}$ to be a diagonal matrix, which can be interpreted as adapting singular values and reweighting the frozen singular vectors. Since only the diagonal elements are learned, this is the most parameter-efficient SVFT variant.
Banded $\left(\text{SVFT}^B_d\right)$. In this approach, we populate $\underline{\boldsymbol{M}}$ using a banded matrix, progressively making off-diagonals learnable. Specifically, for constants z_1 and z_2, $\underline{\boldsymbol{M}_{ij}} = 0$ if $j < i - z_1$ or $j > i + z_2$, where $z_1, z_2 \geq 0$. In our experiments, we set $z_1 = z_2 = d$ to induce off-diagonal elements that capture additional interactions beyond those represented by singular values. This banded perturbation induces local interactions, allowing specific singular values to interact with their immediate neighbors, ensuring smoother transitions. This method, although deviating from the canonical form of SVD, provides a mechanism to capture localized interactions.
Random $\left(\text{SVFT}^R_d\right)$. A straightforward heuristic for populating $\underline{\boldsymbol{M}}$ involves randomly selecting k elements to be learnable.

We illustrate these SVFT design choices in Figure 3. Our empirical results demonstrate that these simple design choices significantly enhance performance compared to state-of-the-art PEFT methods. Note that SVFT^P has a fixed number of learnable parameters, while the remaining variants are configurable. We hypothesize that further innovation is likely achievable through optimizing the sparsity pattern of $\underline{\boldsymbol{M}}$, including efficient learned-sparsity methods. In this paper, we explore these three choices to validate the overall idea: determining a perturbation using the singular vectors of the matrix that is being perturbed. Our empirical results suggest that SVFT^B_d has an edge over other sparsity patterns (see Appendix C.3).

3.2 Properties of SVFT

We highlight some properties of SVFT in the following lemma and provide insights into how its specific algebraic structure compares and contrasts with baseline PEFT methods.

Lemma: Let $\boldsymbol{W}_0$ be a matrix of size $d_1 \times d_2$ with SVD given by $\boldsymbol{U\Sigma V}^T$. Consider an updated final matrix $\boldsymbol{W}_0 + \boldsymbol{UMV}^T$, where $\boldsymbol{M}$ is a matrix of the same size as $\boldsymbol{\Sigma}$, which may or may not be diagonal. Then, the following holds:

(a) Structure: If $\boldsymbol{M}$ is also diagonal (i.e. the plain SVFT), then the final matrix $\boldsymbol{W}_0 + \boldsymbol{UMV}^T$ has $\boldsymbol{U}$ as its left singular vectors and $\text{sign}(\boldsymbol{\Sigma} + \boldsymbol{M})\boldsymbol{V}^T$ as its right singular vectors. That is, its singular

[3]Learnable parameters are underlined.

vectors are unchanged, except for possible sign flips. Conversely, if $\boldsymbol{M}$ is *not* diagonal (i.e., variants of SVFT other than plain), then $\boldsymbol{U}$ and $\boldsymbol{V}$ may no longer be the singular directions of the final matrix $\boldsymbol{W}_0 + \boldsymbol{U}\boldsymbol{M}\boldsymbol{V}^T$.

(b) Expressivity: Given *any* target matrix $\boldsymbol{P}$ of size $d_1 \times d_2$, there exists an $\boldsymbol{M}$ such that $\boldsymbol{P} = \boldsymbol{W}_0 + \boldsymbol{U}\boldsymbol{M}\boldsymbol{V}^T$. That is, if $\boldsymbol{M}$ is fully trainable, any target matrix can be realized using this method.

(c) Rank: If $\boldsymbol{M}$ has k non-zero elements, then the rank of the update $\boldsymbol{U}\boldsymbol{M}\boldsymbol{V}^T$ is at most $\min\{k, \min\{d_1, d_2\}\}$. For the same number of trainable parameters, SVFT can produce a much higher rank perturbation than LoRA (eventually becoming full rank), but in a constrained structured subspace.

We provide our proofs in Appendix A. Building on this lemma, we now compare the form of the SVFT update with LoRA and VeRA. SVFT's $\Delta\boldsymbol{W}$ can be written as a sum of rank-one matrices: $\Delta\boldsymbol{W} = \sum_{(i,j)\in\Omega} \underline{m_{ij}}\boldsymbol{u}_i\boldsymbol{v}_j^T$, where $\boldsymbol{u}_i$ is the i^{th} left singular vector, $\boldsymbol{v}_j$ is the j^{th} right singular vector, and Ω is the set of non-zero elements in $\boldsymbol{M}$. Thus, our method involves adding a weighted combination of specific rank-one perturbations of the form $\boldsymbol{u}_i\boldsymbol{v}_j^T$.

LoRA and VeRA updates can also be expressed as sums of rank-one matrices: $\Delta\boldsymbol{W}_{\text{LoRA}} = \sum_{i=1}^{r} \underline{\boldsymbol{a}_i}\, \underline{\boldsymbol{b}_i}^T$ and $\Delta\boldsymbol{W}_{\text{VeRA}} = \sum_{i=1}^{r} \underline{\alpha_i}(\hat{\boldsymbol{a}}_i \odot \underline{\boldsymbol{\beta}})\hat{\boldsymbol{b}}_i^T$, where $\underline{\boldsymbol{a}_i}$ and $\underline{\boldsymbol{b}_j}$ are the trainable columns of $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices in LoRA. In VeRA, $\hat{\boldsymbol{a}}_i$ and $\hat{\boldsymbol{b}}_i$ are random and fixed vectors, while $\underline{\boldsymbol{\alpha}}$ and $\underline{\boldsymbol{\beta}}$ represent the diagonal elements of $\boldsymbol{\Lambda}_d$ and $\boldsymbol{\Lambda}_b$ respectively.

Note that LoRA requires $d_1 + d_2$ trainable parameters per rank-one matrix, while SVFT and VeRA require only one. Although LoRA can potentially capture directions different from those achievable by the fixed $\{\boldsymbol{u}_i, \boldsymbol{v}_j^T\}$ pairs, each of these directions incurs a significantly higher parameter cost.

VeRA captures new directions at a parameter cost similar to SVFT; however, there is a key distinction: in VeRA, each vector $\hat{\boldsymbol{a}}_i$ or $\hat{\boldsymbol{b}}_i$ appears in only one of the rank-one matrices. In contrast, in SVFT, the same vector $\boldsymbol{u}_i$ can appear in multiple terms in the summation, depending on the sparsity pattern of $\boldsymbol{M}$. This results in an important difference: unlike SVFT, VeRA is *not universally expressive* – it cannot represent any target matrix $\boldsymbol{P}$. Moreover, $\hat{\boldsymbol{a}}_i, \hat{\boldsymbol{b}}_i$ are random, while $\boldsymbol{u}_i, \boldsymbol{v}_j$ depend on $\boldsymbol{W}_0$.

Note. SVFT requires storing both left and right singular vectors due to its computation of SVD on pre-trained weights. While this increases memory usage compared to LoRA (by roughly $1.2\times$), we achieve memory savings in additional buffers, gradient checkpoints, and activations, resulting in lesser overall memory consumption than BOFT and DoRA. A more detailed discussion can be found in section 6.

4 Experiments

4.1 Base Models

We adapt widely-used language models, encoder-only model (DeBERTaV3$_{\text{base}}$ [11]) and two decoder-only models (Gemma-2B/7B [32], LLaMA-3-8B [1]). We also experiment with vision transformer models (ViT-B/16 and ViT-L/16) [9]) pre-trained on ImageNet-21k [8], following prior work [17]. The complete details of our experimental setup and hyperparameter configurations are provided in Appendix C.
Baselines. We compare with **Full Fine-Tuning (FT)** updating all learnable parameters in all layers, along with **LoRA** [15], **DoRA** [19], **BOFT** [20] and **VeRA** [17].[4] **Note:** SVFT^P is equivalent in design to SVF [31] and SVDiff [10], so we do not compare against these methods.

4.2 Datasets

Language. For natural language generation (NLG) tasks, we evaluate on GSM-8K [7] and MATH [12] by fine-tuning on MetaMathQA-40K [35], following [20]. We also evaluate on 8 commonsense reasoning benchmarks (BoolQ [5], PIQA [3], SIQA [30], HellaSwag [36], Winogrande [29], ARC-easy/challenge [6], and OpenBookQA [23]). We follow the setting outlined in

[4]BOFT is approximately three times slower than LoRA. The shared matrices in VeRA can become a limiting factor for models with non-uniform internal dimensions, such as LLaMA-3.

prior work [19, 16], where the training sets of all benchmarks are amalgamated for fine-tuning. We fine-tune on 15K examples from this training set. For natural language understanding (NLU), we evaluate on the General Language Understanding Evaluation (GLUE) benchmark consisting of classification and regression tasks, in line with [17, 15].

Vision. Our experiments on vision tasks consist of 4 benchmarks: CIFAR-100 [18], Food101 [4], RESISC45 [33], and Flowers102 [24]. We follow the setup from [17], and fine-tune on a subset comprising 10 samples from each class.

5 Results

5.1 Performance on Language Tasks

Natural Language Generation. We present results on mathematical question answering against baseline PEFT techniques across three base models – varying from 2B to 8B parameters in Table 1. To ensure a comprehensive comparison, we test baseline techniques (LoRA, DoRA) with different configurations, and varying hyper-parameters like rank to cover a range of learnable parameters from low to high. Note that even when the rank is as low as 1, both methods yield more trainable parameters than SVFT^P. SVFT^P ($\sim$0.2M) shows as much as 18% relative improvement over techniques that use 6$\times$ more trainable parameters ($\text{BOFT}^{b=8}_{m=2}$, $\text{LoRA}_{r=1}$). Against techniques of comparable size (VeRA), SVFT^P achieves **15.5%** relative improvement on average. Even in the default regime, SVFT^R_d matches techniques with at least 3$\times$ more trainable parameters. Notably, on GSM-8K, SVFT^R_d again achieves **96%** of full fine-tuning performance, while $\text{DoRA}_{r=16}$ recovers 86% with 2$\times$ more parameters than SVFT^R_d.

Table 1: Performance (Accuracy) on Mathematical Reasoning (GSM-8K and MATH). #Params denote the number of trainable parameters. **bold** and <u>underline</u> represent the best and second best performing PEFT method, respectively. SVFT offers superior/competitive performance at much lower #Params. For SVFT^R_d, we set $d = 16$ for Gemma and $d = 12$ for LLaMA-3 models.

Method	Gemma-2B			Gemma-7B			LLaMA-3-8B		
	#Params	**GSM-8K**	**MATH**	**#Params**	**GSM-8K**	**MATH**	**#Params**	**GSM-8K**	**MATH**
Full-FT	2.5B	52.69	17.94	8.5B	74.67	25.70	8.0B	64.13	16.24
$\text{LoRA}_{r=32}$	26.2M	43.06	15.50	68.8M	<u>76.57</u>	29.34	56.6M	**75.89**	**24.74**
$\text{DoRA}_{r=16}$	13.5M	<u>44.27</u>	**16.18**	35.5M	74.52	<u>29.84</u>	29.1M	75.66	**24.72**
$\text{BOFT}^{b=8}_{m=2}$	1.22M	36.01	12.13	2.90M	71.79	28.98	4.35M	67.09	21.64
$\text{DoRA}_{r=1}$	1.19M	35.25	13.04	3.26M	74.37	26.28	2.55M	68.30	21.96
$\text{LoRA}_{r=1}$	0.82M	32.97	13.04	0.82M	72.4	26.28	1.77M	68.84	20.94
$\text{VeRA}_{r=1024}$	0.63M	36.77	14.12	0.43M	71.11	27.04	0.98M	63.76	20.28
SVFT^P	0.19M	40.34	14.38	0.43M	73.50	27.30	0.48M	<u>69.22</u>	20.44
SVFT^R_d	6.35M	**50.03**	<u>15.56</u>	19.8M	**76.81**	**29.98**	13.1M	**75.90**	<u>24.22</u>

Commonsense Reasoning. In Table 2, we compare performance on commonsense reasoning benchmarks with Gemma-7B, and observe similar trends. In the lower and moderately parameterized regime ($\sim$0.43M), SVFT^P shows competitive performance in comparison to $\text{LoRA}_{r=1}$ and $\text{DoRA}_{r=1}$, which have 1.9$\times$ and 7.7$\times$ more parameters, respectively. Against VeRA, which trains 3.5$\times$ more parameters, SVFT^P shows a relative improvement of $\sim$**1.16**%. Similarly, $\text{SVFT}^B_{d=8}$ also matches or exceeds methods that use up to 7$\times$ more trainable parameters. For instance, $\text{SVFT}^B_{d=8}$ attains an average performance of 83.35% with only 9.8M parameters, closely matching $\text{LoRA}_{r=16}$ (83.69%, 68.8M parameters). We observe similar trends with Gemma-2B (refer Table 8).

Parameter efficiency. In Figure 1, we plot the performance of SVFT on mathematical reasoning and commonsense reasoning against other PEFT techniques across a range of configurations. Across

Table 2: Evaluation results on eight commonsense reasoning benchmarks with Gemma-7B. We follow [19] for hyperparameter configurations, and report accuracy for all tasks. HS and WG denote HellaSwag [36] and WinoGrande [29], respectively. SVFT^P offers competitive performance at a fraction of #Params. $\mathrm{SVFT}^B_{d=8}$ can match $\mathrm{LoRA}_{r=32}$ with ∼7x fewer parameters.

Method	**#Params**	**BoolQ**	**PIQA**	**SIQA**	**HS**	**WG**	**ARC-e**	**ARC-c**	**OBQA**	**Average**
Full-FT	8.5B	72.32	87.32	76.86	91.07	81.76	92.46	82.76	89.00	84.19
$\mathrm{LoRA}_{r=32}$	68.8M	71.55	**87.95**	**77.27**	91.80	**79.71**	92.67	82.16	**86.40**	**83.69**
$\mathrm{DoRA}_{r=16}$	35.5M	71.46	87.59	76.35	**92.11**	78.29	92.00	80.63	85.60	83.00
$\mathrm{DoRA}_{r=1}$	3.31M	68.22	86.72	75.23	91.14	78.13	91.87	**83.19**	86.20	82.59
$\mathrm{VeRA}_{r=2048}$	1.49M	64.25	86.28	74.04	86.96	69.00	92.76	82.33	82.00	79.70
$\mathrm{LoRA}_{r=1}$	0.82M	65.44	86.28	75.02	89.91	75.92	91.79	81.91	85.40	81.46
SVFT^P	0.51M	67.92	86.45	75.47	86.92	74.03	91.80	81.23	83.00	80.85
$\mathrm{SVFT}^B_{d=8}$	9.80M	**71.90**	86.98	76.28	91.55	78.76	**92.80**	83.11	85.40	83.35

trainable parameter budgets ranging from lowest to highest, SVFT obtains the best overall performance, matching methods that require significantly more trainable parameters. These results establish SVFT as a Pareto-dominant approach for parameter-efficient fine-tuning.

We present additional results on **Natural Language Understanding** in Appendix C.1.

5.2 Performance on Vision Tasks

Table 3 contrasts SVFT against other PEFT techniques on image classification benchmarks using ViT-B and ViT-L models. SVFT consistently outperforms other methods across datasets, particularly $\mathrm{SVFT}^B_{d=4}$, which achieves comparable or superior results to full fine-tuning.

Table 3: Performance on image classification benchmarks. For all methods, we only adapt {Q, V} modules of the transformer.

Method	**ViT Base**					**ViT Large**				
	#Params	**CIFAR100**	**Food101**	**Flowers102**	**RESISC45**	**#Params**	**CIFAR100**	**Food101**	**Flowers102**	**RESISC45**
Head	-	78.58	75.14	98.71	64.42	-	79.14	75.66	98.89	64.99
Full-FT	85.8M	85.02	75.41	99.16	**75.3**	303.3M	87.37	78.67	98.88	**80.17**
$\mathrm{LoRA}_{r=8}$	294.9K	**85.65**	76.13	99.14	74.01	786.4K	87.36	**78.95**	99.24	79.55
$\mathrm{VeRA}_{r=256}$	24.6K	84.00	74.02	99.10	71.86	61.4K	**87.55**	77.87	**99.27**	75.92
SVFT^P	18.5K	83.78	74.43	98.99	70.55	49.2K	86.67	77.47	99.09	73.52
$\mathrm{SVFT}^R_{d=4}$	165.4K	84.85	76.45	99.17	74.53	441.5K	87.05	**78.95**	99.23	78.90
$\mathrm{SVFT}^B_{d=4}$	165.4K	84.65	**76.51**	**99.21**	75.12	441.5K	86.95	78.85	99.24	78.93

5.3 Contribution of Each Weight Type

In Figure 4, we investigate the contribution of each weight type. Starting with the base configuration, we apply SVFT^B_d to the $\boldsymbol{Q}$ and $\boldsymbol{V}$ weights in each transformer block and report the performance. We then incrementally add the remaining weight modules ($\boldsymbol{K}, \boldsymbol{U}, \boldsymbol{D}, \boldsymbol{O}, \boldsymbol{G}$) and observe the changes in performance. For each configuration, we also vary the trainable parameters by incrementing the total learnable off-diagonals.

Note that applying SVFT^B_d to $\boldsymbol{U}, \boldsymbol{D}, \boldsymbol{O}$, and $\boldsymbol{G}$ does not increase trainable parameters as much as applying LoRA or DoRA to these modules (Table 6). For example, for a large matrix of shape $d_1 \times d_2$, $\mathrm{LoRA}_{r=1}$ learns $d_1 + d_2$ parameters, while SVFT^P learns $\min(d_1, d_2)$ parameters. We observe that adapting only $\boldsymbol{U}$ and $\boldsymbol{D}$ with SVFT yields up to a 10% relative improvement over adapting $\boldsymbol{Q}$ and $\boldsymbol{V}$ for the same parameter budget ($\sim 0.8M$). Our findings indicate that adapting more weight types enhances performance.

Table 4: Impact of pre-trained weight quality. Results on GSM-8K after fine-tuning on Pythia-2.8B checkpoints at different stages of pre-training (PT). Compared to LoRA, SVFT benefits more from better pre-trained weights.

Method	#Params	PT Steps		ΔPerf
		39K	**143K**	
Full-FT	2.5B	21.00	30.09	9.09
LoRA	5.24M	11.22	18.95	7.73
SVFT	5.56M	15.08	23.19	8.11

Table 5: GPU Memory analysis for Gemma-7B. SVFT outperforms both LoRA and DoRA in terms of performance while requiring lesser GPU memory than DoRA. GPU memory is measured in gigabytes (GB).

Method	Target Modules	#Params	GPU Mem	GSM-8K	MATH
LoRA$_{r=4}$	Q,K,V,U,D	8.6M	63.57	73.76	28.3
DoRA$_{r=4}$	Q,K,V,U,D	9.72M	78.70	75.43	28.46
LoRA$_{r=32}$	Q,K,V,U,D	68.8M	64.24	76.57	29.34
DoRA$_{r=16}$	Q,K,V,U,D	35.5M	78.99	74.52	29.84
SVFT^{P}	Q,K,V,U,D,O,G	429K	76.26	73.5	27.3
SVFT$^{R}_{d=8}$	Q,K,V,U,D,O,G	9.8M	76.65	73.62	28.36
SVFT$^{R}_{d=16}$	Q,K,V,U,D,O,G	19.8M	77.04	**76.81**	**29.98**

5.4 Impact of Pre-trained Weight Quality

A key feature of SVFT is that the weight update depends on the pre-trained weights $\boldsymbol{W}$. We therefore ask the following question: *Does the quality of pre-trained weights have a disproportionate impact on* SVFT*?* To answer this, we consider two checkpoints from the Pythia suite [2] at different stages of training, i.e., 39K steps and 143K steps, respectively. We fine-tune each of these checkpoints independently with Full-FT, LoRA, and SVFT. We then compare the increase in performance (ΔPerf). As shown in Table 4, compared to LoRA, SVFT benefits more from better pre-trained weights. We also note that SVFT outperforms LoRA in both settings, suggesting that the benefits of inducing a $\Delta\boldsymbol{W}$ that explicitly depends on $\boldsymbol{W}$ are beneficial even when $\boldsymbol{W}$ is sub-optimal.

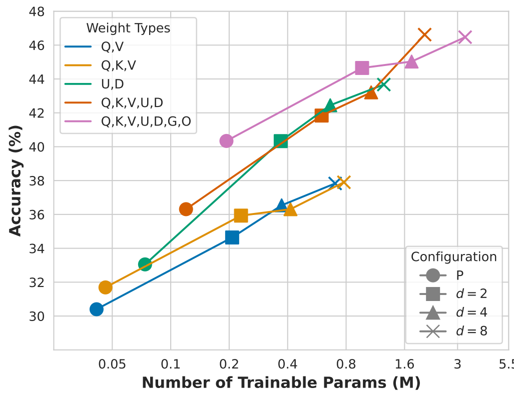


Figure 4: Performance variation of SVFT$^{B}_{d}$ on GSM-8K with Gemma-2B based on adapted weight matrices. Adapting more target weight types results in higher performance gains. Notably, under a fixed parameter budget, adapting $\boldsymbol{U}$ and $\boldsymbol{D}$ yields greater gains than adapting $\boldsymbol{Q}$ and $\boldsymbol{V}$.

6 Discussion

In this work, we investigate sparsity patterns guided by heuristics and show that they can significantly enhance performance. A principled approach to constructing these sparsity patterns can lead to additional improvements. In terms of memory consumption, SVFT requires computing the SVD and storing both left and right singular vectors, resulting in $\sim 1.2\times$ more memory usage than LoRA and its variants. However, by significantly reducing the total number of trainable parameters, we achieve memory savings in additional buffers, gradient checkpoints, and activations. Consequently, SVFT consumes comparable or even less memory than DoRA (see Table 5). Future work will focus on quantization and other techniques to further improve memory efficiency.

7 Conclusion

This work introduces SVFT, a novel and efficient PEFT approach that leverages the structure of pre-trained weights to determine weight update perturbations. We propose three simple yet effective sparse parameterization patterns, offering flexibility in controlling the model's expressivity and the number of learnable parameters. Extensive experiments on language and vision tasks demonstrate SVFT's effectiveness as a PEFT method across diverse parameter budgets. Furthermore, we theoretically show that SVFT can induce higher-rank perturbation updates compared to existing methods, for a fixed parameter budget. In future work, we aim to develop principled methods to generate sparsity patterns, potentially leading to further performance improvements.

References

[1] Meta AI. Introducing meta llama 3: The most capable openly available llm to date. April 2024.

[2] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023.

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

[4] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

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

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

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

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[10] Ligong Han, Yinxiao Li, Han Zhang, Peyman Milanfar, Dimitris Metaxas, and Feng Yang. Svdiff: Compact parameter space for diffusion fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7323–7334, 2023.

[11] Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023.

[12] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

[13] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, page 1693–1701. MIT Press, 2015.

[14] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2019.

[15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[16] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models, 2023.

[17] Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. ELoRA: Efficient low-rank adaptation with random matrices. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[19] 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, 2024.

[20] Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. Parameter-efficient orthogonal finetuning via butterfly factorization. In *The Twelfth International Conference on Learning Representations*, 2024.

[21] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[22] Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *arXiv preprint arXiv:2404.02948*, 2024.

[23] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering, 2018.

[24] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[25] Zelin Peng, Zhengqin Xu, Zhilin Zeng, Xiaokang Yang, and Wei Shen. Sam-parser: Fine-tuning sam efficiently by parameter space reconstruction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:4515–4523, 03 2024.

[26] Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, volume 36, pages 79320–79362, 2023.

[27] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[28] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[29] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale, 2019.

[30] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions, 2019.

[31] Yanpeng Sun, Qiang Chen, Xiangyu He, Jian Wang, Haocheng Feng, Junyu Han, Errui Ding, Jian Cheng, Zechao Li, and Jingdong Wang. Singular value fine-tuning: Few-shot segmentation requires few-parameters fine-tuning. In *Advances in Neural Information Processing Systems*, volume 35, pages 37484–37496, 2022.

[32] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[33] Ihsan Ullah, Dustin Carrion, Sergio Escalera, Isabelle M Guyon, Mike Huisman, Felix Mohr, Jan N van Rijn, Haozhe Sun, Joaquin Vanschoren, and Phan Anh Vu. Meta-album: Multi-domain meta-dataset for few-shot image classification. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[34] Yeming Wen and Swarat Chaudhuri. Batched low-rank adaptation of foundation models. In *The Twelfth International Conference on Learning Representations*, 2024.

[35] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models, 2023.

[36] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[37] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 11328–11339. PMLR, 13–18 Jul 2020.

[38] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

Appendix

The appendix is organized as follows.

- In Appendix A, we give proofs for the lemmas outlined in 3.2.
- In Appendix B, we compare how the trainable parameters count for different PEFT techniques (LoRA, DoRA, VeRA) versus our method SVFT.
- In Appendix C, we describe results for additional experiments and provide implementation details for all the experiments.

A Proofs

We provide brief proofs for the *Structure*, *Expressivity* and the *Rank* lemmas for SVFT:

(a) *Structure:* If $\boldsymbol{M}$ is diagonal, then the final matrix $\boldsymbol{W}_0 + U\boldsymbol{M}V^T$ can be written as $U(\Sigma + \boldsymbol{M})V^T$ since $\boldsymbol{W}_0 = U\Sigma V^T$, where $(\Sigma + \boldsymbol{M})$ is also a diagonal matrix. Thus, $U(\Sigma + \boldsymbol{M})V^T$ is a valid and unique SVD of $\boldsymbol{W}_0 + U\boldsymbol{M}V^T$ up to sign flips in the singular vectors.

(b) *Expressivity:* Finding $\boldsymbol{M}$ for any target matrix P of size $d_1 \times d_2$ such that $P = \boldsymbol{W}_0 + U\boldsymbol{M}V^T$ is the same as finding $\boldsymbol{M}$ for a new target matrix $P' = P - \boldsymbol{W}_0$ such that $P' = U\boldsymbol{M}V^T$. For a full SVD, the dimension of $\boldsymbol{M}$ is $d_1 \times d_2$ and since the dimension of P' is also $d_1 \times d_2$, $P' = U\boldsymbol{M}V^T$ is a bijection and $\boldsymbol{M} = U^T(P - \boldsymbol{W}_0)V$ (since U and V are orthogonal).

(c) *Rank:* If $\boldsymbol{M}$ has k non-zero elements, then the rank of the update $U\boldsymbol{M}V^T$ will be upper bounded by k (since by Gaussian elimination, k or less elements will remain, the best case being all k elements in the diagonal). We also know that the rank is upper bounded by $\min\{d_1, d_2\}$, giving an achievable upper bound on the rank as $\min\{k, \min\{d_1, d_2\}\}$.

B Parameter Count Analysis

Table 6: Parameter count analysis. L_{tuned}, D_{model}, r, k denote total layers being adapted, hidden dimension, rank, and additional off-diagonals respectively.

Method	**Trainable Parameter Count**
LoRA	$2 \times L_{\text{tuned}} \times D_{\text{model}} \times r$
DoRA	$L_{\text{tuned}} \times D_{\text{model}} \times (2r + 1)$
VeRA	$L_{\text{tuned}} \times (D_{\text{model}} + r)$
SVFT^P	$L_{\text{tuned}} \times D_{\text{model}}$
SVFT$^B_{d=k}$	$L_{\text{tuned}} \times (D_{\text{model}} \times k + (D_{\text{model}} - k)(k + 1))$

C Additional Experiments and Implementation Details

All of our experiments are conducted on a Linux machine (Debian GNU) with the following specifications: 2xA100 80 GB, Intel Xeon CPU @ 2.20GHz with 12 cores, and 192 GB RAM. For all our experiments (including baseline experiments), we utilize hardware-level optimizations like mixed weight precision (e.g., bfloat16) whenever possible.

C.1 Natural Language Understanding

Results on the GLUE benchmark are summarized in Table 7. SVFT matches LoRA$_{r=8}$ and DoRA$_{r=4}$ which use **12-22**$\times$ more trainable parameters. Similarly, when compared to OFT and BOFT, SVFT^P maintains a comparable average performance despite being 12$\times$ smaller. These results highlight SVFT's ability to strike a balance between parameter efficiency and performance, making it an attractive PEFT choice for simple classification tasks.

Table 7: DeBERTaV3$_{\text{base}}$ with different adaptation methods on the GLUE benchmark. We report matched accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all tasks. * indicates numbers published in prior work.

Method	**#Params**	**MNLI**	**SST-2**	**MRPC**	**CoLA**	**QNLI**	**QQP**	**RTE**	**STS-B**	**Avg.**
Full-FT*	184M	89.90	95.63	89.46	69.19	94.03	**92.40**	83.75	91.60	88.25
LoRA*$_{r=8}$	1.33M	**90.65**	94.95	<u>89.95</u>	69.82	93.87	91.99	85.20	91.60	88.50
DoRA$_{r=4}$	0.75M	89.92	95.41	89.10	69.37	94.14	91.53	87.00	<u>91.80</u>	88.53
BOFT*$^{b=8}_{m=2}$	0.75M	<u>90.25</u>	**96.44**	**92.40**	**72.95**	<u>94.23</u>	<u>92.10</u>	**88.81**	**91.92**	**89.89**
LoRA$_{r=1}$	0.17M	90.12	95.64	86.43	69.13	94.18	91.43	87.36	91.52	88.23
VeRA$_{r=1024}$	0.09M	89.93	95.53	87.94	69.06	93.24	90.4	87.00	88.71	87.73
SVFT^P	0.06M	89.69	95.41	88.77	70.95	**94.27**	90.16	87.24	<u>91.80</u>	88.54
SVFT$^R_{d=2}$	0.28M	89.97	<u>95.99</u>	88.99	<u>72.61</u>	93.90	91.50	<u>88.09</u>	91.73	<u>89.10</u>

C.2 Commonsense Reasoning Gemma-2B

We evaluate and compare SVFT variants against baseline PEFT methods on commonsense reasoning tasks with the Gemma-2B model and tabulate results in Table 8.

Table 8: Results with Gemma-2B on eight commonsense reasoning benchmarks. We follow [19] for hyperparameter configurations, and report accuracy for all tasks.

Method	**#Params**	**BOOLQ**	**PIQA**	**SIQA**	**HellaSwag**	**Winogrande**	**ARC-E**	**ARC-C**	**OBQA**	**Average**
Full-FT	2.5B	63.57	74.1	65.86	70.00	61.95	75.36	59.72	69	67.45
LoRA$_{r=32}$	26.2M	63.11	73.44	63.20	47.79	52.95	74.78	57.16	67.00	62.43
LoRA$_{r=16}$	13.5M	62.87	73.93	65.34	53.16	55.51	76.43	59.55	68.4	64.40
BOFT$^{b=8}_{m=2}$	1.22M	59.23	63.65	47.90	29.93	50.35	59.04	42.66	41.00	49.22
VeRA$_{r=2048}$	0.66M	62.11	64.31	49.18	32.00	50.74	58.08	42.83	42.6	50.23
LoRA$_{r=1}$	0.82M	62.2	69.31	56.24	32.47	51.53	69.52	48.8	56.4	55.81
DoRA$_{r=1}$	1.19M	62.17	68.77	55.93	32.95	51.22	68.81	48.72	55.6	55.52
SVFT^P	0.19M	62.26	70.18	56.7	32.47	47.04	69.31	50.08	58.4	55.81
SVFT$^B_{d=16}$	6.35M	63.42	73.72	63.86	71.21	59.58	73.69	54.77	66.6	65.86

C.3 Impact of M's Structure on Performance

We explore another heuristic-based parameterization, Top-k, which focuses on identifying and leveraging the most significant interactions in the data. The design choices we explore involve computing the alignment between the left and right singular vectors as $\boldsymbol{u}_i^T \boldsymbol{v}_j$. We then select the top-k elements and make them learnable. However, note that this only works when left and right singular vectors have the same size. A possible interpretation of this is we make only the top-k strong interactions between singular vector directions learnable.

We analyze the impact of different parameterizations of $\boldsymbol{M}$ (Plain, Banded, Random, Top-k) on downstream performance. To ensure a fair comparison, we match the number of trainable coefficients across all variants. As shown in Table 9, both Random and Top-k variants outperform Banded on the GSM-8K dataset. However, this improvement comes at the cost of performance on MATH. This observation suggests that the choice of parameterization has a significant impact on model performance, and the effectiveness of a particular structure may vary depending on the downstream task.

Table 9: Results on fine-tuning Gemma-2B with SVFT using different $\boldsymbol{M}$ parameterizations.

Structure	#Params	GSM-8K	MATH
Plain	0.2M	40.34	14.38
Banded	3.3M	46.47	**16.04**
	6.4M	47.84	15.68
Random	3.3M	47.76	<u>15.98</u>
	6.4M	**50.03**	15.56
Top-k	3.3M	48.00	15.80
	6.4M	<u>49.65</u>	15.32

C.4 Are All Singular Vectors Important?

To determine the importance of considering all singular vectors and singular values during fine-tuning, we reduce the rank of $\boldsymbol{U}$ and $\boldsymbol{V}$, and truncate $\boldsymbol{\Sigma}$ and $\boldsymbol{M}$ to an effective rank of r. If the original weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, then after truncation, $\boldsymbol{U} \in \mathbb{R}^{m \times r}, \boldsymbol{V} \in \mathbb{R}^{n \times r}$. This truncation significantly reduces the number of trainable parameters, so we compensate by increasing the number of off-diagonal coefficients (d) in $\boldsymbol{M}$.

Our results, with four different configurations of r and d, are presented in Table 10. The findings show that a very low rank ($r = 128$) leads to poor performance, even when parameters are matched. A reasonably high rank of $r = 1536$, which is 75% of the full rank, still fails to match the performance of the full-rank variant that has $0.25\times$ the trainable parameters. This indicates that all singular vectors significantly contribute to the end task performance when fine-tuning with SVFT, and that important information is lost even when truncating sparingly.

Table 10: Performance with varying rank (r) and the off-diagonal elements (d) of $\boldsymbol{M}$. When $r = 2048$, the update is full-rank.

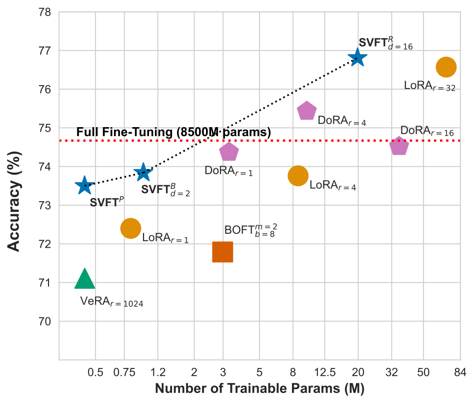
Rank (r)	Diags (d)	#Params	GSM-8K	MATH
128	64	1.55M	0.98	0.21
1536	-	0.15M	16.37	3.64
1536	2	0.74M	25.01	6.04
2048	-	0.19M	**40.34**	**14.38**

C.5 Performance vs Total Trainable Parameters

In addition to the experiments performed in Figure 1 for Gemma-2B on challenging natural language generation (NLG) tasks like GSM-8K and Commonsense Reasoning, we also plot the performance vs total trainable parameters for larger state-of-the-art models like Gemma-7B and LLaMA-3-8B on GSM-8K. Figure 5 further demonstrates SVFT's Pereto-dominance. On larger models, we observe that full-finetuning overfits, leading to sub-optimal performance in comparison to PEFT methods.

C.6 Memory Consumption Analysis

We compare the accuracy and memory required to train Gemma-2B and Llama3-8B with different PEFT techniques and summarize the results in Table 11 and Table 12. We used HuggingFace's internal memory profiler to measure peak GPU memory usage during training. In summary, SVFT uses $\sim 1.2\times$ more GPU memory than LoRA but is comparable to or less than DoRA while offering better performance on GSM-8K and MATH.



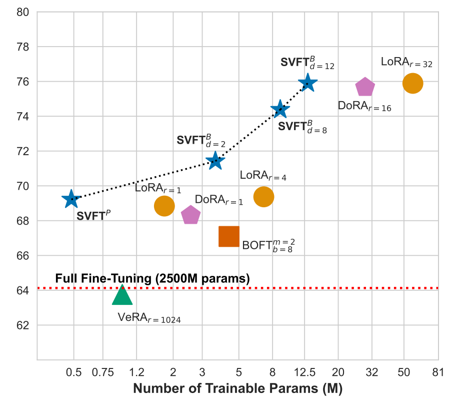


Figure 5: Performance versus total trainable parameters for GSM-8K on Gemma-7B (left) and LLaMA-3-8B (right).

Table 11: Memory analysis for Gemma-2B.

Method	**Target Modules**	**#Params**	**GPU Mem (GB)**	**GSM-8K**	**MATH**
LoRA$_{r=4}$	Q,K,V,U,D	3.28M	18.88	40.63	14.5
DoRA$_{r=4}$	Q,K,V,U,D	3.66M	24.58	41.84	15.04
LoRA$_{r=32}$	Q,K,V,U,D	26.2M	19.06	43.06	15.5
DoRA$_{r=16}$	Q,K,V,U,D	13.5M	24.64	44.27	16.18
SVFT$^R_{d=12}$	Q,K,V,U,D	2.98M	20.50	47.61	**16.72**
SVFT^P	Q,K,V,U,D,O,G	194K	21.90	40.34	14.38
SVFT$^R_{d=8}$	Q,K,V,U,D,O,G	3.28M	22.02	47.76	15.98
SVFT$^R_{d=16}$	Q,K,V,U,D,O,G	6.35M	22.15	**50.03**	15.56

C.7 Settings for Language Tasks

Natural Language Understanding. We fine-tune the DeBERTaV3$_{\text{base}}$ [11] model and apply SVFT to all linear layers in every transformer block of the model. We only moderately tune the batch size, learning rate, and number of training epochs. We use the same model sequence lengths used by [20] to keep our comparisons fair. The hyperparameters used in our experiments can be found in Table 13.

Natural Language Generation. See the hyperparameters used in our experiments in Table 14. For LoRA, DoRA, we adapt Q, K, V, U, D matrices. We apply BOFT on Q, V matrices since applying on multiple modules is computationally expensive. For VeRA, which enforces a constraint of uniform internal dimensions for shared matrices, we apply on G, U projection matrices as it yields the highest number of learnable parameters. We apply SVFT on Q, K, V, U, D, O, G for the Gemma family of models, and U, D, O, G for LLaMA-3-8B. Note that applying SVFT on these modules does not increase trainable parameters at the same rate as applying LoRA or DoRA on them would. We adopt the code base from `https://github.com/meta-math/MetaMath.git` for training scripts and evaluation setups and use the fine-tuning data available at `https://huggingface.co/datasets/meta-math/MetaMathQA-40K`.

Commonsense Reasoning. See the hyperparameters used in our experiments in Table 15. We adopt the same set of matrices as that of natural language generation tasks. We use the code base from `https://github.com/AGI-Edgerunners/LLM-Adapters`, which also contains the training and evaluation data.

Table 12: Memory analysis for LLaMA-3-8B.

Method	Target Modules	#Params	GPU Mem (GB)	GSM-8K	MATH
LoRA$_{r=1}$	Q,K,V,U,D	1.77M	57.82	68.84	20.94
DoRA$_{r=1}$	Q,K,V,U,D	2.56M	70.17	68.30	21.96
LoRA$_{r=32}$	Q,K,V,U,D	56.6M	58.41	75.89	24.74
DoRA$_{r=16}$	Q,K,V,U,D	29.1M	70.44	75.66	24.72
SVFT$^B_{d=24}$	Q,K,V,U,D	15.98M	71.52	75.32	**25.08**
SVFT$^R_{d=12}$	U,D,O,G	13.1M	70.37	**75.90**	24.22
SVFT^P	Q,K,V,U,D,O,G	483K	73.95	69.22	20.44
SVFT$^R_{d=12}$	Q,K,V,U,D,O,G	15.9M	71.52	73.99	**25.08**

Table 13: Hyperparameter setup used for DeBERTaV3$_{\text{base}}$ on the GLUE benchmark.

Method	Dataset	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
	Optimizer	AdamW							
	Warmup Ratio	0.1							
	LR Schedule	Linear							
	Learning Rate (Head)	6E-03							
	Max Seq. Len.	256	128	320	64	512	320	320	128
	# Epochs	10	10	30	20	10	6	15	15
SVFT^P	Batch Size	32	32	16	16	32	16	4	32
	Learning Rate	5E-02	5E-02	5E-02	8E-02	8E-02	5E-02	5E-02	5E-02
SVFT$^R_{d=2}$	Batch Size	32	32	16	16	32	32	16	32
	Learning Rate	1E-02	1E-02	1E-02	1E-02	3E-02	1E-02	3E-02	1E-02

C.8 Settings for Vision Tasks

For each dataset in the vision tasks, we train on 10 samples per class, using 2 examples per class for validation, and test on the full test set. Similar to previous literature, we always train the classifier head for these methods since the number of classes is large. The parameter counts do not include the number of parameters in the classification head. The hyperparameters are mentioned in Table 16. We tune the learning rates for SVFT and BOFT select learning rates for other methods from [17], run training for 10 epochs, and report test accuracy for the best validation model. For all methods, since the classification head has to be fully trained, we report the parameter count other than the classification head.

Table 14: Hyperparameter setup used for fine-tuning on MetaMathQA-40K.

Hyperparameter	Gemma-2B		Gemma-7B		LLaMA-3-8B	
	SVFT^P	$\text{SVFT}^R_{d=16}$	SVFT^P	$\text{SVFT}^R_{d=16}$	SVFT^P	$\text{SVFT}^R_{d=12}$
Optimizer	AdamW					
Warmup Ratio	0.1					
LR Schedule	Cosine					
Learning Rate	5E-02	1E-03	5E-02	1E-03	5E-02	1E-03
Max Seq. Len.	512					
# Epochs	2					
Batch Size	64					

Table 15: Hyperparameter setup used for fine-tuning on commonsense-15K.

Hyperparameter	Gemma-2B		Gemma-7B	
	SVFT^P	$\text{SVFT}^B_{d=8}$	SVFT^P	$\text{SVFT}^B_{d=8}$
Optimizer	AdamW			
Warmup Steps	100			
LR Schedule	Linear			
Max Seq. Len.	512			
# Epochs	3			
Batch Size	64			
Learning Rate	5E-02	5E-03	5E-02	1E-03

Table 16: Hyperparameter setup used for fine-tuning on all vision tasks.

Hyperparameter	ViT-B	ViT-L
Optimizer	AdamW	
Warmup Ratio	0.1	
Weight Decay	0.01	
LR Schedule	Linear	
# Epochs	10	
Batch Size	64	
SVFT^P Learning Rate (Head)	4E-03	
SVFT^P Learning Rate	5E-02	
$\text{SVFT}^B_{d=2}$ Learning Rate (Head)	4E-03	
$\text{SVFT}^B_{d=2}$ Learning Rate	5E-02	
$\text{SVFT}^B_{d=8}$ Learning Rate (Head)	4E-03	
$\text{SVFT}^B_{d=8}$ Learning Rate	5E-03	