VectorFit : Adaptive Singular & Bias Vector Fine-Tuning of Pre-trained Foundation Models

Suhas Hegde[a], **Shilpy Kaur**[a] **and Aruna Tiwari**[a]

[a]Indian Institute of Technology Indore

Abstract. Popular PEFT methods reduce trainable parameter count for fine-tuning by parameterizing new low-rank or sparse trainable weights in parallel to the frozen pre-trained weights W. However, these weights are trained from scratch, and there exists a performance gap between these methods and full fine-tuning, especially in low-budget settings. We introduce VectorFit, a new way of parameterization that efficiently utilizes the existing knowledge embedded in W by adaptively training their singular vectors and biases. We show that utilizing the structural and transformational properties of W in this way can lead to high-rank incremental weight matrices ΔW, comparable to that of full fine-tuning. VectorFit delivers superior results with **9**$\times$ fewer trainable parameters than the leading PEFT methods. Through comprehensive experiments across 19 datasets covering a wide range of language and vision tasks such as natural language understanding and generation, question answering, image classification, and image generation, we demonstrate that VectorFit surpasses baselines in terms of performance as a function of parameter-efficiency.

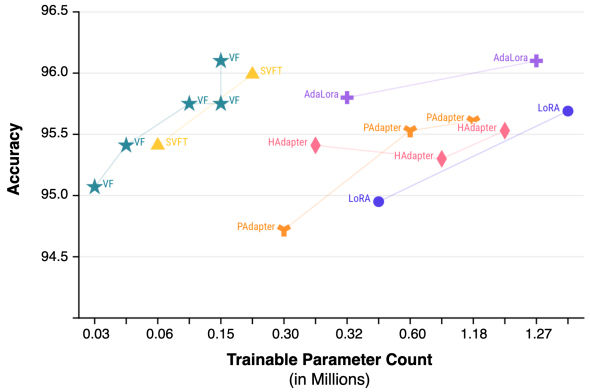

Figure 1. Accuracy vs Trainable parameter count for SST2 dataset. VectorFit (labeled as VF for brevity) outperforms baselines with 85% less trainable parameters. The graph highlights that VectorFit is a PEFT method in extremely low parameter regime of <0.1% trainable parameters.

1 Introduction

Pre-trained foundation models (PFMs) have set unprecedented standards in language, vision, and audio tasks [39, 33, 30], showcasing their strong performance across diverse domains. Refining these models through fine-tuning is a powerful approach to enhance their performance across diverse downstream tasks [20, 45]. This process helps models adhere to given instructions [43], adopt preferred behaviors, and discard undesirable ones [31]. However, adapting these models to downstream tasks through full fine-tuning (Full-FT) is a significant challenge, primarily due to the immense computational and memory overhead. For instance, models like DeBERTa-V3 [10] with 300 million parameters, ViT-22B [3] with 22 billion parameters, and Llama-3 [8] with a staggering 405 billion parameters exemplify the scale of modern PFMs. Adapting these models for multiple downstream tasks is resource-intensive and typically requires maintaining separate copies of the full model for each task, leading to a very high memory consumption.

Parameter-Efficient Fine-Tuning (PEFT) mitigates these challenges by introducing a small set of trainable parameters to produce specialized models. For example, PFMs like LLMs are fine-tuned for tasks such as text classification [40], question answering [32], and text generation [19] using task-specific datasets. PEFT techniques, such as LoRA [13] and adapter [28], significantly reduce the number of trainable parameters compared to Full-FT, although this can compromise the performance. More advanced methods like AdaLoRA [49] try to increase expressiveness by adaptively choosing trainable parameters, bridging the performance gap. However, majority of successful PEFT methods work by adding a new set of weight matrices with the assumption that incremental weight matrices are low-rank. This may limit their expressiveness and disregards the nuances of weight matrix transformation during Full-FT. Moreover, even state-of-the-art PEFT techniques (e.g., LoRA, Adapter, and AdaLoRA) can still result in a substantial number of trainable parameters, even in their highest parameter-efficient setup (e.g., LoRA with rank 1). Although there are a few recent methods that do not rely on low-rank updates [29, 22], all of them, to the best of our knowledge, work based on fine-tuning a newly initialized set of weight matrices. This leads to a high overall parameter count and memory consumption, twice as that of LoRA in SVFT [22].

This prompts the question: Can we achieve extreme parameter-efficiency through the efficient use of pre-training knowledge embedded in the weight matrices, without incurring prohibitive parameter and memory costs? As an answer, we introduce VectorFit, which directly leverages the structural and transformational characteristics of the pre-trained weights instead of introducing new weights for fine-tuning, differentiating our method from prior work. Given a pre-trained weight matrix W_0, VectorFit applies singular value decomposition (SVD), such that $W_0 = U\Sigma V^T$. The method then selectively adapts the singular vector (Σ) and the bias (b) associated with W_0, focusing on those Σ and b that exhibit suboptimal training compared to those of other weight matrices. We propose a mechanism called

Adaptive Vector Freezing to achieve this.

Since the singular vectors represent the stretching of the transformation applied by the weight matrices in their high-dimensional subspace, directly fine-tuning them allows for high expressiveness (Appendix D.5). This is evident from the fact that VectorFit performs high-rank updates comparable to Full-FT (Figure 10) while using significantly fewer trainable parameters ($\leq 0.1\%$). Additionally, training the bias vectors gives translational degree of freedom, further enhancing the expressiveness during fine-tuning. VectorFit outperforms the baselines in terms of performance relative to parameter efficiency (Figure 1). It also gives a practical memory consumption approximately equivalent to that of LoRA with rank 1 for smaller base models (Figure 5). The memory consumption of VectorFit lies in between LoRA and SVFT for larger base models (Figure 6).

2 Related Work

Researchers have explored three primary approaches to reduce the number of parameters required for fine-tuning while preserving or enhancing the performance of PFMs. These approaches can be broadly categorized into Adapter-based methods, LoRA-based methods, and other PEFT methods.

Adapter-Based methods. This research direction emphasizes incorporating small neural networks into PFMs and fine-tuning only these modules for specific tasks, keeping the base model frozen and shared across tasks. This approach introduces a limited number of task-specific parameters, significantly improving the scalability and practicality of large models. For instance, adapter tuning [12, 28, 9] integrates small neural networks, known as adapters, between the layers of the base model. Other methods, such as prefix tuning [21] and prompt tuning [17], add trainable prefix tokens to the input or hidden layers of the model. These techniques claim to have demonstrated performance comparable to Full-FT while updating less than 1% of the model parameters, significantly reducing memory requirements.

LoRA-Based methods. A significant advancement in PEFT is Low-Rank Adaptation (LoRA) [13], which preserves the pre-trained model weights and incorporates trainable low-rank matrices within each transformer layer. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d_r \times d_c}$, LoRA constrains the weight update ΔW to a low-rank decomposition: $y = W_0x + \Delta Wx = W_0x + BAx$, where $B \in \mathbb{R}^{d_r \times r}$, $A \in \mathbb{R}^{r \times d_c}$ and rank $r << min(d_r, d_c)$. Only A and B are trainable parameters.

Several studies have introduced variations of the LoRA algorithm, focusing on reducing the number of trainable parameters [47, 15, 6], improving the flexibility of low-rank structures [14, 50, 37], enabling adaptive parameter allocation [46], and integrating LoRA with techniques like quantization [5, 44] and pruning [48].

A significant enhancement over LoRA is AdaLoRA [49], which addresses LoRA's limitation of evenly distributing trainable parameters across weight matrices, ignoring their varying importance. In AdaLoRA, the incremental updates are parameterized as singular value decomposed matrices $P\Lambda Q$, where $P \in \mathbb{R}^{d_r \times r}$ and $Q \in \mathbb{R}^{r \times d_c}$ are the left and right singular matrices, $\Lambda \in \mathbb{R}^{r \times 1}$ is the singular vector. The orthogonality of P and Q is maintained using the regularizer $R(P, Q) = \|P^\top P - I\|_F^2 + \|QQ^\top - I\|_F^2$. The rank of low-rank updates is dynamically adjusted using an importance metric derived from Λ. By pruning less significant singular values while allowing for recovery, AdaLoRA claims to have improved performance with a similar parameter budget as LoRA. Pissa [24] is another method that works similar to LoRA while the initialization of the low-rank trainable weights is done using the principal singular values and vectors.

Other PEFT methods. Orthogonal Fine-Tuning (OFT) [29] introduces an orthogonal projection approach using orthogonal regularization. It focuses on optimizing parameters while preserving the orthogonality of weight updates, ensuring minimal interference with pre-trained knowledge. However, it still demands a significant number of trainable parameters because of the high dimensionality of the matrices. Butterfly Orthogonal Fine-Tuning (BOFT) [23] builds upon OFT by introducing Butterfly factorization and claims to improve parameter efficiency, and fine-tuning flexibility. Singular Vectors guided Fine-Tuning (SVFT) [22] leverages the singular value decomposition of pre-trained weight matrices to parameterize weight updates as $y = W_0x + \Delta Wx = U(\Sigma + M)V^Tx$, where M is a sparse trainable matrix with pre-determined and fixed sparsity pattern. As M is not restricted to be low-rank, SVFT claims to achieve high-rank gradient updates. Nonetheless, SVFT uses four matrices, U, Σ, V, and M, for every pre-trained weight matrix. Also, their dimensions are comparable to those of the pre-trained weight matrix. This leads to a high parameter and memory cost.

3 VectorFit

In this section, we describe VectorFit and its components in detail. VectorFit comprises two key components: (1) Vector Fine-Tuning, based on SVD. (2) *Adaptive Vector Freezing*, a mechanism to avoid co-adaptation and to improve the performance.

3.1 Vector Fine-Tuning

VectorFit initially performs SVD on the pre-trained weight matrix $W_0 \in \mathbb{R}^{d_r \times d_c}$, such that, $W_0 = U\Sigma V^T$. W_0 can be the weight matrix of any of the modules in self-attention (q, k, v, o) or multilayer perceptron (f_1, f_2) of a transformer block. Then we potentially fine-tune only the singular vector Σ and the pre-trained bias vector b_0 corresponding to W_0, subject to *Adaptive Vector Freezing*, as shown in Figure 2. Formally, this can be denoted as follows:

$$y = \left(U\Sigma V^T\right)x + b_0 \tag{1}$$

where $x \in \mathbb{R}^{d_{in} \times d_r}$ is the input hidden state. $U \in \mathbb{R}^{d_r \times d_r}$ is the left singular matrix and $V \in \mathbb{R}^{d_c \times d_r}$ is the right singular matrix of W_0, consisting of orthonormal column vectors. V^T is the transpose of V. $\Sigma \in \mathbb{R}^{d_r \times 1}$ and $b_0 \in \mathbb{R}^{d_c}$ are the potentially trainable singular vector and bias vector of W_0, respectively. $y \in \mathbb{R}^{d_{out} \times d_c}$ is the output hidden state. Note that Σ in standard SVD is a diagonal matrix and we store it as a vector for memory efficiency. The weight updates of VectorFit are parameterized as:

$$W = W_0 + \Delta W = U(\Sigma + \Delta\Sigma)V^T \tag{2}$$

$$b = b_0 + \Delta b \tag{3}$$

where ΔW, $\Delta\Sigma$, and Δb are the incremental matrix/vectors of W_0, Σ, and b_0, respectively.

Singular values of a weight matrix quantify the scaling factors for the transformation along its orthogonal directions, an important aspect of the weight matrix. Directly fine-tuning them as described above results in an overall high-rank incremental matrix whose rank is comparable to that of full fine-tuning of the weight matrix. This is analyzed in detail in Section 6.2.

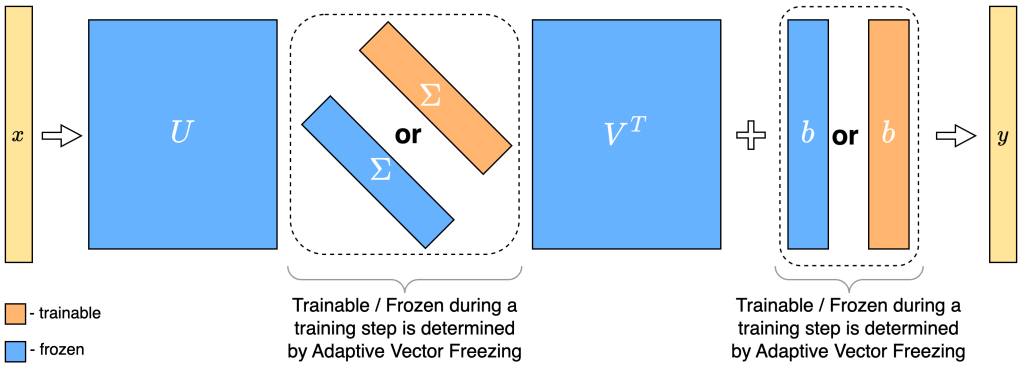

Figure 2. Architecture diagram of VectorFit. The pretrained weight matrix is initially decomposed into U, Σ, and V. Subsequently, only Σ and bias b are trained with Adaptive Vector Freezing mechanism.

To avoid performing the expensive calculation of SVD for every single pre-trained weight matrix during each training step, we perform SVD in the beginning of the fine-tuning and replace the original weight matrices of the model with their decomposed version. This takes a few seconds of extra time in the beginning of the fine-tuning, which is negligible. Although this approach increases the total parameter count—for instance, VectorFit with DeBERTaV3-base has 18% more parameters than LoRA (r = 1) with DeBERTaV3-base—its practical training memory consumption remains similar to LoRA (r = 1). More details on this is given in Appendix A.

3.2 *Adaptive Vector Freezing*

As we train only a small number of parameters (a few tens of singular and bias vectors), it is crucial to ensure balanced training across all trainable vectors. This prevents some vectors from being over-trained while others remain under-trained, a phenomenon known as co-adaptation.

To address this, we propose *Adaptive Vector Freezing (AVF)*, a mechanism that periodically freezes (disables the gradients of) the top-k trainable vectors that have undergone extensive training. This allows the remaining under-trained vectors to receive adequate updates. The extent of training of each vector is quantified in the form of training strength. Consider the set of all trainable vectors, $V = \{\Sigma_{l,m}, b_{l,m} : l = layer, m = module(q, k, v, o, f_1, f_2)\}$. We define the training strength $S_v(t)$ of a vector $v \in V$ at training step t as L1 norm between v_0 and v_t, formulated as follows:

$$S_v(t) = \frac{1}{dim(v)} ||v_0 - v_t||_1 \tag{4}$$

where v_0 is the value of v before fine-tuning and v_t is the value of v during the training step t. $dim(v)$ is the dimension of v. To find the top-k vectors, we perform exponential moving average of $S_v(t)$ as given in Eq. 5.

$$S'_v(t) = \beta S'_v(t - t_f) + (1 - \beta) S_v(t) \tag{5}$$

where $\beta = 0.99$, is a constant. We define the training step at which AVF is applied as the AVF step. Given the frequency of AVF steps t_f, the first AVF step t_i, the number of vectors k to freeze per AVF step, and the total number of AVF steps n_f, the top-k vectors with the highest $S'_v(t)$ values are frozen at each AVF step. Note that the trainability of vectors do not change in between AVF steps. However, a vector frozen during one AVF step may become trainable in a subsequent AVF step, ensuring that all vectors are adequately trained over time. More details on these hyperparameters are given in Appendix C.

In Section 6.1, we theoretically and experimentally show that this mechanism leads to similar effect as that of dropout. It is important to note that using the standard dropout algorithm for singular vectors results in a significant performance drop, even with a very low dropout probability. This signifies that certain singular values, and their corresponding left and right singular directions are extremely important that they cannot be dropped. Therefore, the AVF mechanism is crucial for maintaining effective training.

4 Experiments

Implementation details. All algorithms are implemented using PyTorch [27]. Our implementation builds upon the publicly available Huggingface Transformers codebase [42]. Appendix C contains the full details of our experimental setup and hyperparameter configurations.

4.1 *Tasks and Datasets*

We evaluate our method on the following tasks and datasets:

1. Natural Language Understanding (NLU): Experiments are conducted on the GLUE benchmark [40], which includes single-sentence classification, similarity/paraphrase, and natural language inference tasks.
2. Question Answering (QA): Performance is tested on SQuAD v1.1 and SQuAD v2.0 [32], treating QA as a sequence labeling problem to predict the start and end token probabilities for answer spans.
3. Natural Language Generation (NLG): Evaluation is performed on the XSum [26] and CNN/DailyMail [25] datasets for text summarization task. GSM8K [2] and Math [11] datasets are used to evaluate mathematical problem-solving capabilities.
4. Image Classification: Our method is assessed on image classification tasks using CIFAR10 [16], GTSRB [36], MNIST [4], and RESISC45 [1] datasets.
5. Image Generation: The results on subject-driven image generation is evaluated using the Dreambooth dataset [34].

Table 1. DeBERTaV3-base fine-tuned using various PEFT methods is evaluated on the GLUE benchmark. For performance metrics, we report matched accuracy for MNLI, Matthew's correlation for COLA, Pearson correlation for STS-B, and accuracy for the other tasks, where higher values indicate better performance across all metrics. # Params is the number of trainable parameters.

Method	# Params	MNLI	SST2	COLA	QQP	QNLI	RTE	MRPC	STSB
Full FT	184M	89.90/90.12	95.63	69.19	92.40/89.80	94.03	83.75	89.46	91.60
HAdapter	1.22M	90.13/90.17	95.53	68.64	91.91/89.27	94.11	84.48	89.95	91.48
PAdapter	1.18M	90.33/90.39	95.61	68.77	92.04/89.40	94.29	85.20	89.46	91.54
LoRA(r=8)	1.33M	90.65/90.69	94.95	69.82	91.99/89.38	93.87	85.20	89.95	91.60
AdaLora	1.27M	**90.76/90.79**	**96.10**	**71.45**	**92.23/89.74**	**94.55**	**88.09**	**90.69**	**91.84**
HAdapter	0.61M	90.12/90.23	95.30	67.87	91.65/88.95	93.76	85.56	89.22	91.30
PAdapter	0.60M	90.15/90.28	95.53	69.48	91.62/88.86	93.98	84.12	89.22	91.52
HAdapter	0.31M	90.10/90.02	95.41	67.65	91.54/88.81	93.52	83.39	89.25	91.31
PAdapter	0.30M	89.89/90.06	94.72	69.06	91.40/88.62	93.87	84.48	89.71	91.38
LoRA(r=2)	0.33M	90.30/90.38	94.95	68.71	91.61/88.91	94.03	85.56	89.71	91.68
AdaLora	0.32M	**90.66/90.70**	95.80	70.04	**91.78/89.16**	**94.49**	87.36	90.44	91.63
Pissa	0.33M	89.99/90.13	94.86	68.79	91.57/88.73	93.92	85.09	89.84	91.69
SVFT	0.28M	89.90/89.97	95.99	**72.61**	91.50/88.98	93.90	**88.09**	88.99	91.73
VectorFit	0.15M	90.12/89.89	**96.10**	70.94	91.51/88.70	94.05	84.12	**92.16**	**91.76**

4.2 Base Models

We use six distinct base model types that are representative of a wide range of PFMs for the evaluation of our algorithm:

1. DeBERTaV3-base [10], a transformer encoder-only language model, applied to NLU and QA tasks.
2. BART-large [18], a transformer encoder-decoder model, used for NLG tasks.
3. Gemma-7B [38], a transformer decoder-only model, used for NLG tasks.
4. Llama-3-8B [8], a transformer decoder-only model, used for NLG tasks.
5. ViT-base [7], a vision transformer model, applied to image classification tasks, pre-trained on Imagenet-1K.
6. Stable Diffusion v1.4 [33], a latent diffusion model based on UNet architecture, used for text-to-image generation.

Baselines. We compare our approach against Full-FT, which updates all parameters across all layers. Additionally, we evaluate it against state-of-the-art methods from each of the three categories mentioned in Section 2. These include LoRA [13], AdaLoRA [49], PAdaptor [28], HAdaptor [12], Pissa [24], and SVFT [22].

5 Results

In the tables, the highest accuracy within each trainable parameter count regime is highlighted in **bold**, and the overall parameter efficiency (% accuracy / % trainable parameters) is reported with underline.

5.1 Natural Language Understanding

Table 1 presents the results on the GLUE benchmark, where VectorFit outperforms Full-FT by an average of 0.6% while requiring over 1200× fewer trainable parameters. Its performance is comparable to baselines like LoRA (r = 8), which uses 9× more trainable parameters. VectorFit outperforms SVFT by upto 3.2% with 2× less parameters. Notably, VectorFit achieves the highest parameter efficiency across all datasets in the GLUE benchmark, establishing it as the most optimal PEFT method for natural language understanding tasks.

5.2 Question Answering

We evaluate the performance of our method on the SQuAD v1.1 and the more challenging SQuAD v2.0 datasets, using exact match (EM) and F1 scores as metrics. The results, summarized in Table 2, demonstrate that VectorFit outperforms the baselines on SQuAD v1.1 giving 0.9% better F1 score on an average. It achieves superior results compared to Full-FT with 1250× fewer parameters. On SQuAD v2.0, VectorFit delivers performance comparable to Full-FT and the best-performing baselines, highlighting its efficiency and effectiveness.

Table 2. Performance results for DeBERTaV3-base fine-tuned on SQuAD v1.1 and SQuAD v2.0 are presented. # Params indicates the percentage of trainable parameters. The metrics reported are Exact Match and F1 scores (EM/F1).

Model	Squad v1.1 (EM/F1)	Squad v2.0 (EM/F1)
Full FT	86.0 / 92.7	85.4 / 88.4
# Params	0.08%	0.08%
HAdapter	84.4 / 91.5	83.4 / 86.6
PAdaptor	84.4 / 91.7	84.2 / 87.2
LoRA	86.4 / 92.8	84.6 / 87.5
AdaLora	86.8 / 93.0	**84.7 / 87.6**
Pissa	85.9 / 92.3	84.2 / 87.3
SVFT	86.3 / 92.5	84.3 / 87.3
VectorFit	**87.0 / 93.2**	84.4 / **87.6**

5.3 Natural Language Generation

We evaluate VectorFit on the XSum and CNN/DailyMail datasets using the ROUGE (1/2/L) metrics in Table 3. Despite a 33.3% higher relative parameter efficiency than the baselines, VectorFit consistently outperforms them on both datasets. Notably, VectorFit recovers 95% of Full-FT Rouge-L score with only 0.12% trainable parameters, compared to the baselines that recover 86% accuracy with

Table 3. Performance results for BART-large fine-tuned on the XSum and CNN/DailyMail datasets are shown. The # Params column represents the percentage of trainable parameters. The reported metrics are ROUGE-1, ROUGE-2, and ROUGE-L (R-1/2/L).

Method	# Params	Xsum	CNN/Dailymail
Full FT	100%	45.49 / 22.33 / 37.26	44.16 / 21.28 / 40.90
PAdapter	0.16%	40.21 / 18.92 / 32.34	41.96 / 19.47 / 38.10
LoRA	0.16%	42.81 / 19.68 / 34.73	43.68 / 20.63 / 40.71
AdaLoRA	0.16%	43.29 / 19.95 / 35.04	43.94 / 20.83 / 40.96
Pissa	0.16%	42.67 / 19.51 / 34.16	42.97 / 20.04 / 40.11
SVFT	0.15%	**43.30** / 19.82 / 35.13	43.87 / 20.72 / 40.80
VectorFit	0.12%	43.28 / **20.71** / **35.42**	**44.01** / **21.60** / **40.98**

Table 4. Comparison of various methods for Gemma-7B and Llama-3-8B models, fine-tuned for mathematical reasoning. The table includes number of parameters and accuracy on GSM-8k and MATH benchmarks.

Method	Gemma-7B			Llama-3-8B		
	# Params	GSM-8k	MATH	# Params	GSM-8k	MATH
Full FT	8.5B	74.67	25.70	8.0B	64.13	16.24
LoRA (r=1)	0.82M	72.4	26.28	1.77M	68.84	20.94
AdaLoRA (r=1)	0.81M	72.5	26.41	1.77M	68.73	**21.09**
Pissa	0.82M	72.3	26.31	1.77M	67.92	21.01
SVFT	0.43M	73.50	27.30	0.48M	69.22	20.44
VectorFit	0.43M	**73.94**	**27.41**	0.48M	**70.83**	21.02

Table 5. The performance results for ViT-base fine-tuned on the CIFAR10, GTSRB, MNIST, and RESISC45 datasets are presented. The # Params column indicates the proportion of trainable parameters, and the corresponding image classification accuracies are reported. Section 6.3 contains more details about the variations of VectorFit.

Method	# Params	CIFAR10	GTSRB	MNIST	RESISC45
Full-FT	100%	98.5	99.2	99.8	95.7
LoRA	0.3%	98.4	99.2	**99.7**	**95.8**
AdaLoRA	0.3%	98.6	99.3	99.6	**95.8**
SVFT	0.3%	98.7	99.5	99.6	95.0
VectorFit (Σ)	0.06%	98.6	98.0	98.3	92.2
VectorFit (*no avf*)	0.1%	99.0	99.6	99.0	94.4
VectorFit	0.1%	**99.1**	**99.8**	99.4	95.1

0.16% trainable parameters on the Xsum dataset. VectorFit's performance surpasses Full-FT on CNN/Dailymail dataset, demonstrating superior efficiency and performance on complex tasks. Additionally, we evaluate VectorFit for mathematical reasoning with Gemma-7B and Llama-3-8B, as shown in Table 4, highlighting that our method scales well to larger base model sizes.

5.4 Image Classification

Table 5 showcases the results on image classification tasks. Our method surpasses Full-FT performance with only 0.1% trainable parameters. VectorFit achieves comparable results to the baselines while maintaining 80% higher relative parameter efficiency. Notably, VectorFit (Σ) demonstrates the highest parameter efficiency, with an average accuracy reduction of just 1.5% compared to Full-FT.

5.5 Image Generation

Table 6 summarizes the results of personalized image generation using Dreambooth-style fine-tuning [34]. The evaluation is conducted using three metrics: DINO, CLIP-I, and CLIP-T, as outlined in [34]. Our method recovers an average DINO, CLIP-I, and CLIP-T score of 98.2% of Full-FT accuracy as opposed to 97.3% achieved by LoRA with 0.04% of trainable parameters. Figure 12 provides a visual comparison between Full-FT and VectorFit.

Table 6. Quantitative evaluation of Stable Diffusion v1.4 fine-tuned with PEFT methods using Dreambooth approach for Subject-driven Image generation. We evaluate subject fidelity using DINO and CLIP-I, and prompt fidelity using CLIP-T. Higher values indicate better performance across all metrics. # Params indicates the percentage of trainable parameters.

Method	# Params	DINO	CLIP-I	CLIP-T
Full-FT	100%	0.651	0.817	0.293
LoRA	0.04%	0.636	0.789	0.286
VectorFit	0.04%	0.642	0.796	0.289

6 Discussion

6.1 Effect of Adaptive Vector Freezing

Proposition 1. *AVF has an effect comparable to that of dropout.*

Proof. Let n_u represent the total number of training/gradient update steps. The gradient of a vector v with respect to loss L is $\nabla_v L$.

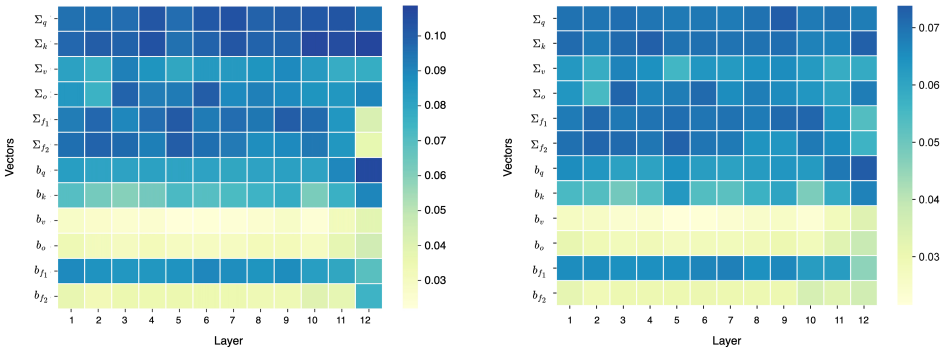

Figure 3. The training strength S_v of each trainable vector after fine-tuning of DeBERTaV3-base on the COLA dataset is shown for VectorFit without AVF (left) and with AVF (right). The x-axis represents the layer index, while the y-axis corresponds to different types of trainable vectors. The heatmaps show the regularization effect (overall lower S_v values) and the balanced training achieved with AVF.

The expected gradient for the vector v per step without AVF is expressed as:

$$\mathbb{E}\left[\nabla_v L\right] = \frac{1}{n_u}\sum_{i=1}^{n_u}\left(\nabla_v L\right)_{[i]} \tag{6}$$

With n_f AVF steps, the training process can be divided into n_f+1 gradient update intervals. Let p_j denote the probability that a vector v is frozen during the interval j. Let i_s and i_e be the temporary variables that denote the first and last gradient update step within each interval respectively. The expected gradient update per step for v under AVF can be expressed as:

$$\mathbb{E}_f\left[\nabla_v L\right] = \frac{1}{n_u}\sum_{j=1}^{n_f+1}\sum_{i=i_s}^{i_e}(1-p_j)\left(\nabla_v L\right)_{[i]} \tag{7}$$

$$= \frac{1}{n_u}\left[\sum_{i=1}^{n_u}\left(\nabla_v L\right)_{[i]} - \sum_{j=1}^{n_f+1}\sum_{i=i_s}^{i_e}(p_j)\left(\nabla_v L\right)_{[i]}\right] \tag{8}$$

$$\mathbb{E}_f\left[\nabla_v L\right] = \mathbb{E}\left[\nabla_v L\right] - \frac{1}{n_u}\sum_{j=1}^{n_f+1}\sum_{i=i_s}^{i_e}(p_j)\left(\nabla_v L\right)_{[i]} \tag{9}$$

The second term of Eq. 9 captures the regularization effect of AVF. A similar analysis can be applied to dropout, demonstrating that AVF effectively minimizes co-adaptation. This effect is empirically validated in Figure 3.

6.2 Rank Analysis

Proposition 2: *If the rank of a matrix is high, it contains more information compared to its low-rank counterpart.*

Proof. Let A be a matrix of dimensions $m \times n$ with rank p. Suppose A is decomposed using Singular Value Decomposition (SVD) as:

$$A = U\Sigma V^T,$$

where U and V are orthonormal matrices, and Σ is a diagonal matrix containing singular values.

Now, consider a lower-rank approximation of A, denoted as A_k, with rank k such that $k < p$. The truncated SVD of A_k is given by:

$$A_k = U_k\Sigma_k V_k^T,$$

where U_k, V_k, and Σ_k correspond to the top k singular components of A.

The difference in Frobenius norm between A and A_k is given by:

$$\|A - A_k\|_F = \sqrt{\sum_{i=k+1}^{p}\sigma_i^2},$$

where σ_i are the singular values of A.

This difference represents the information loss due to low-rank approximation. Since the Frobenius norm quantifies the variance within the matrix, reducing the rank leads to a loss in variance information. Therefore, a higher-rank matrix retains more information than its lower-rank approximation.

Proposition 3: *Singular vector updates lead to high-rank incremental matrices.*

Proof. Consider the incremental weight matrix Δ^*:

$$\Delta^* = W_{\text{init}} - W_{\text{final}}$$

Expressing in terms of singular value decomposition:

$$\Delta^* = U\Sigma_{\text{init}}V^T - U\Sigma_{\text{final}}V^T$$

$$\Delta^* = U\Delta\Sigma V^T$$

Since U and V are orthogonal matrices, the rank of Δ^* is upper bounded by the rank of $\Delta\Sigma$. This ensures that Δ^* achieves a full-rank incremental update when all singular values are updated, meaning no singular values are truncated.

Furthermore, by the Eckart–Young–Mirsky theorem [41], the best rank-k approximation of a matrix A is obtained by retaining the top k singular values. Conversely, if no singular values are truncated (i.e., all singular values are updated or remain nonzero), the approximation Δ^* preserves the full rank:

$$r = \min(d_r, d_c).$$

where d_r and d_c are the dimensions of weight matrix. This confirms that singular vector updates lead to high-rank incremental matrices. We experimentally verify this by plotting the singular values of Δ^* from randomly selected layers as shown in Figure 10 of Appendix D.4.

Table 7. Ablation study to analyze the efficacy of AVF. The table presents results for L1 regularization, Random Vector Freezing, and AVF.

Method	SST-2	CoLA	RTE	MRPC	STSB	SQuAD v1.1	SQuAD v2.0
L1 regularization	90.02	64.74	69.38	74.77	84.53	71.3 / 74.5	60.9 / 63.1
Random vector freezing	94.16	70.13	82.82	91.26	90.17	85.1 / 91.7	82.8 / 85.9
Adaptive vector freezing (AVF)	96.10	70.94	84.12	92.16	91.76	87.0 / 93.2	84.4 / 87.6

6.3 *Ablations on Choice of Vectors and AVF*

This section presents the experiments that reveal the contribution of different vectors and the AVF mechanism to the performance of VectorFit. To this end, we explore 5 variants of VectorFit.

VectorFit (Σ_a): The singular vectors corresponding to $\{q, k, v, o\}$ are trained.

VectorFit (Σ): The singular vectors corresponding to $\{q, k, v, o, f_1, f_2\}$ are trained.

VectorFit ($\Sigma_a + b$): The singular vectors corresponding to $\{q, k, v, o\}$ and all the bias vectors are trained.

VectorFit (no avf): The singular vectors corresponding to $\{q, k, v, o, f_1, f_2\}$ and all the bias vectors are trained.

VectorFit: The singular vectors corresponding to $\{q, k, v, o, f_1, f_2\}$ and all the bias vectors are trained along with AVF.

Figure 4 presents the results of fine-tuning the DeBERTaV3-base model across five variants mentioned above on QA tasks. On the SQuADv1.1 dataset, the performance difference between VectorFit (*no AVF*) and VectorFit is 0.2%. On the more challenging SQuADv2.0 dataset, this difference increases to 0.5%, highlighting the critical role of AVF. Additionally, the average 1% performance gap between VectorFit (Σ_a) and VectorFit (Σ) underscores the importance of singular vectors associated with the fully connected modules (f_1 and f_2) in the transformer block. Lastly, the performance difference between VectorFit (Σ_a) and VectorFit ($\Sigma_a + b$) emphasizes the significance of training the bias vectors.

Figure 8 provides a similar analysis on the GLUE benchmark, yielding results consistent with the observations discussed earlier. Further examination of the training strength of various vectors for different VectorFit variants fine-tuned on the COLA dataset is included in Appendix D.1.

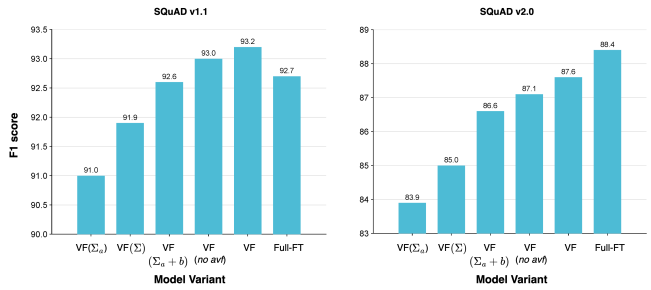

Figure 4. Ablation study about AVF and different trainable vectors configuration. We report the F1 scores for SQuAD v1.1 and SQuAD v2.0 datasets of QA task.

6.4 *Efficacy of AVF Against Other Possible Approaches*

In this section, we present the experiments conducted by replacing AVF with other possible approaches. We explore L1 regularization and Random vector freezing, where we randomly freeze trainable vectors. From Table 7 it can be seen that AVF consistently outperforms the other two possible approaches. A key distinction between AVF and L1 regularization lies in the granularity at which they operate. Standard L1 regularization regularizes individual singular values, whereas AVF targets the overall strength of training of singular vectors and bias vectors. We hypothesize that regularizing individual singular values may constrain the model's expressiveness, as it indirectly limits the flexibility of the associated left and right singular vectors. Random vector freezing performs better than L1 regularization but it leads to slower and unstable training. On the other hand, AVF leads to higher performance with stable training.

6.5 *Limitations*

Although VectorFit demonstrates exceptional performance with high parameter efficiency, the AVF mechanism's effectiveness depends on careful hyperparameter selection, a challenge shared by comparable methods like AdaLoRA. To overcome this, we provide some heuristics for hyperparameter selection in Appendix C. Another limitation is that the number of trainable parameters is currently bounded, as no new parameterized weights are introduced. In future work, we aim to address this by exploring the parameterization of left and right singular matrices, potentially increasing the upper limit of trainable parameters and further enhancing the method's flexibility and performance.

7 Conclusion

We introduce VectorFit, a novel PEFT approach that extracts meaningful singular vectors from weight matrices using SVD and adaptively trains the singular and bias vectors. This method enables high-rank and intrinsic knowledge-aware adaptation of pre-trained models, significantly enhancing both model performance and parameter efficiency. Through comprehensive experiments across diverse language and vision tasks, we demonstrate that VectorFit surpasses existing methods in terms of performance as a function of parameter efficiency. Also, utilizing VectorFit to fine-tune PFMs for downstream tasks is straightforward and cost effective. Additionally, we provide extensive theoretical and empirical insights into its operation to enable further research in this area. In future, we plan to conduct mathematical analysis of weight matrix transformations during fine-tuning, aiming to develop novel parameterization strategies beyond singular vectors and biases.

References

[1] G. Cheng, J. Han, and X. Lu. Remote sensing image scene classification: Benchmark and state of the art. 2017. URL http://dx.doi.org/10.1109/JPROC.2017.2675998.

[2] K. Cobbe and et al. Training verifiers to solve math word problems. In *Arxiv.org*, 2021. URL arXiv:2110.14168.

[3] M. Dehghani, J. Djolonga, and et al. Scaling vision transformers to 22 billion parameters, 2023. URL https://arxiv.org/abs/2302.05442.

[4] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 2012.

[5] T. Dettmers and et al. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2024.

[6] N. Ding, X. Lv, and et al. Sparse low-rank adaptation of pre-trained language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=jxgz7FEqWq.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, and et al. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

[8] A. Grattafiori, A. Dubey, A. Jauhri, and A. Pandey. The llama 3 herd of models. Technical report, Meta, https://arxiv.org/abs/2407.21783, 2024.

[9] J. He and et al. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=0RDcd5Axok.

[10] P. He, J. Gao, and W. Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2023. URL https://arxiv.org/abs/2111.09543.

[11] D. Hendrycks, C. Burns, and et al. A new proof of eckart-young-mirsky theorem. In *Arxiv.org*, 2021. URL arXiv:2103.03874.

[12] N. Houlsby, , and et al. Parameter-efficient transfer learning for NLP. *CoRR*, 2019. URL http://arxiv.org/abs/1902.00751.

[13] E. J. Hu and et al. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[14] S. A. Koohpayegani, N. K. L, and et al. NOLA: Compressing loRA using linear combination of random basis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=TjfXcDgvzk.

[15] D. J. Kopiczko, T. Blankevoort, and Y. M. Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

[16] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. URL https://api.semanticscholar.org/CorpusID:18268744.

[17] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://aclanthology.org/2021.emnlp-main.243/.

[18] M. Lewis, Y. Liu, and et al. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. URL https://aclanthology.org/2020.acl-main.703/.

[19] J. Li and et al. Pretrained language models for text generation: A survey, 2021. URL https://arxiv.org/abs/2105.10311.

[20] R. Li, L. B. allal, and et al. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=KoFOg41haE.

[21] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.acl-long.353/.

[22] V. Lingam, A. T. Neerkaje, and et al. SVFT: Parameter-efficient fine-tuning with singular vectors. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024. URL https://openreview.net/forum?id=DOUskwCqg5.

[23] W. Liu, Z. Qiu, Y. Feng, and et al. Parameter-efficient orthogonal fine-tuning via butterfly factorization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=7NzgkEdGyr.

[24] F. Meng, Z. Wang, and M. Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. In *Thirty-eigth Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6ZBHIEtdP4.

[25] R. Nallapati, B. Zhou, and et al. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2016. URL https://aclanthology.org/K16-1028/.

[26] S. Narayan and et al. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018. URL https://aclanthology.org/D18-1206/.

[27] A. Paszke and et al. Pytorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019.

[28] J. Pfeiffer and et al. Adapterfusion: Non-destructive task composition for transfer learning, 2021. URL https://arxiv.org/abs/2005.00247.

[29] Z. Qiu, W. Liu, H. Feng, and et al. Controlling text-to-image diffusion by orthogonal finetuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=K30wTdIIYc.

[30] A. Radford, J. W. Kim, and et al. Robust speech recognition via large-scale weak supervision, 2022. URL https://arxiv.org/abs/2212.04356.

[31] R. Rafailov, A. Sharma, and et al. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

[32] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text, 2016. URL https://arxiv.org/abs/1606.05250.

[33] R. Rombach and et al. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. URL https://api.semanticscholar.org/CorpusID:245335280.

[34] N. Ruiz, Y. Li, and et al. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation, 2023. URL https://arxiv.org/abs/2208.12242.

[35] A. Shi and Z. DeVito. Understanding gpu memory 1: Visualizing all allocations over time. URL https://pytorch.org/blog/understanding-gpu-memory-1/.

[36] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Selected Papers from IJCNN 2011*, 2012. URL https://www.sciencedirect.com/science/article/pii/S0893608012000457.

[37] C. Sun, J. Wei, and et al. Svfit: Parameter-efficient fine-tuning of large pre-trained models using singular values, 2024. URL https://arxiv.org/abs/2409.05926.

[38] G. Team. Gemma: Open models based on gemini research and technology. In *Arxiv.org*, 2024.

[39] H. Touvron, T. Lavril, and et al. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

[40] A. Wang and et al. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019. URL https://arxiv.org/abs/1804.07461.

[41] H. Wang. A new proof of eckart-young-mirsky theorem. In *Preprints.org*, 2025. URL https://doi.org/10.20944/preprints202502.1203.v1.

[42] T. Wolf, L. Debut, V. Sanh, and et al. Huggingface's transformers: State-of-the-art natural language processing, 2020. URL https://arxiv.org/abs/1910.03771.

[43] C. Xu, Q. Sun, K. Zheng, and et al. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=CfXh93NDgH.

[44] Y. Xu, L. Xie, X. Gu, and et al. QA-loRA: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=WvFoJccpo8.

[45] L. Yu, W. Jiang, H. Shi, and et al. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=N8N0hgNDRt.

[46] F. Zhang, L. Li, and et al. Increlora: Incremental parameter allocation method for parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2308.12043.

[47] L. Zhang and et al. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning, 2024. URL https://openreview.net/forum?id=RbKThNNFxr.

[48] M. Zhang, H. Chen, and et al. LoRAPrune: Pruning meets low-rank parameter-efficient fine-tuning, 2024. URL https://openreview.net/forum?id=9KVT1e1qf7.

[49] Q. Zhang, M. Chen, A. Bukharin, and et al. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2303.10512.

[50] B. Zi, X. Qi, and et al. Delta-loRA: Fine-tuning high-rank parameters with the delta of low-rank matrices, 2024. URL https://openreview.net/forum?id=FAO4VS9QRV.

Appendix

A Memory Usage

As discussed in Section 3.1, while retaining the left and right singular matrices increases the overall parameter count, the practical impact is minimal. Figure 5 compares the GPU memory usage of VectorFit and LoRA (r = 1) on the MNLI dataset using DeBERTaV3-base with full precision training. The figure demonstrates that both methods use comparable amounts of memory, where VectorFit requires approximately 200MB of additional memory using 0.08% of trainable parameters. Figure 6 shows the GPU memory consumption comparison between LoRA, VectorFit, and SVFT with different base models. We observe that VectorFit consumes higher memory compared to LoRA on larger models while using significantly less memory than that of SVFT. The experiments were conducted on an Nvidia A100 GPU with 40GB of RAM.

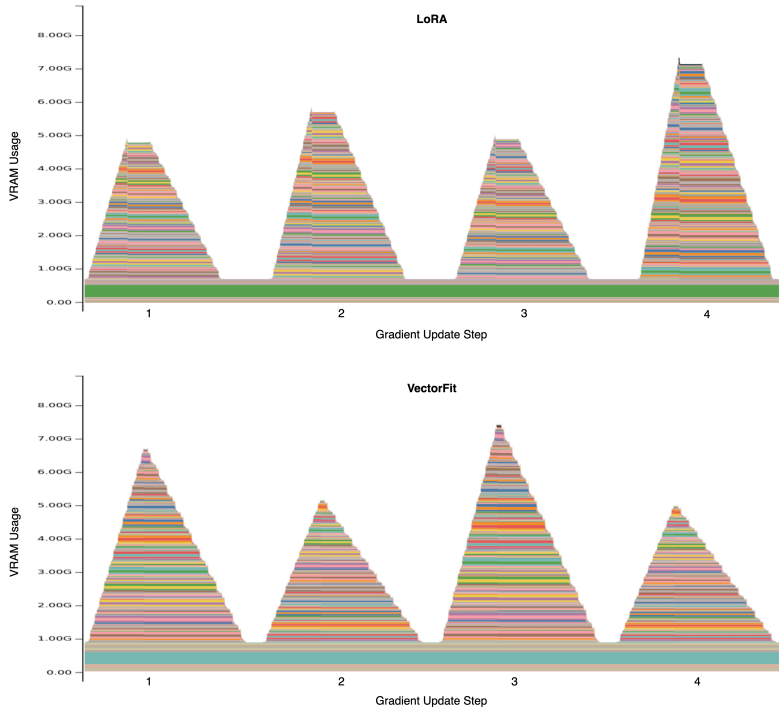

Figure 5. PyTorch memory trace [35] comparison of 4 training steps for LoRA (r = 1) on the top and VectorFit on the bottom.

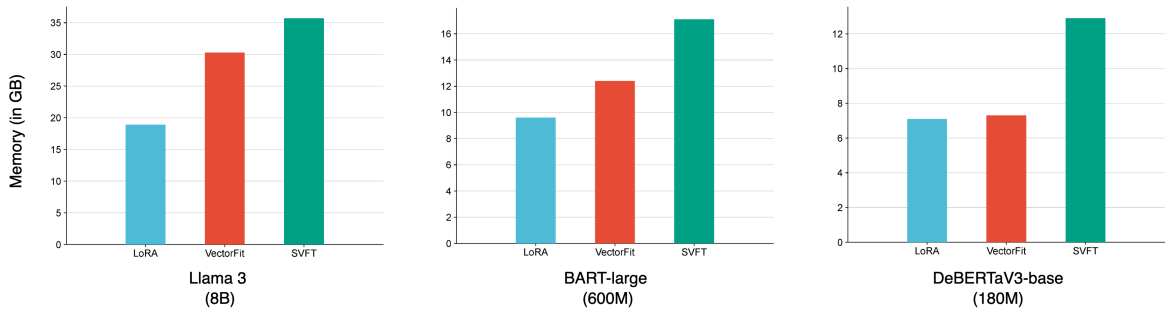

Figure 6. Memory usage.

B Training Speed

We fine-tune DeBERTaV3-base on the MNLI dataset with full precision and on the SQuAD v2.0 dataset with mixed precision training to assess the training speed of VectorFit, measured as the time required to train one epoch. Table 8 shows that VectorFit reduces training time by 17.5% on MNLI and 16.6% on SQuAD v2.0 compared to baseline. This improvement is due to VectorFit's simpler computational graph compared to other methods, resulting in faster processing. This experiment was conducted on an Nvidia Titan XP GPU with 12GB of RAM.

C Implementation Details

This section outlines the implementation details of our method and the baselines used in various experiments. Most of our experiments were conducted on the NVIDIA A100(40G) GPU. We employ the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, no warmup, and no weight

Table 8. Comparison of practical training time.

Dataset	# Params	Method	Time / Epoch
MNLI	0.08%	LoRA	82 min
	0.08%	AdaLoRA	91 min
	0.08%	VectorFit	75 min
	0.07%	VectorFit ($\Sigma_a + b$)	71 min
	0.01%	VectorFit (Σ_a)	64 min
SQuAD v2.0	0.08%	LoRA	98 min
	0.08%	AdaLoRA	108 min
	0.08%	VectorFit	90 min
	0.07%	VectorFit ($\Sigma_a + b$)	84 min
	0.01%	VectorFit (Σ_a)	75 min

decay for all our experiments. For the AVF-related hyperparameters, we adopt the following values as a general guideline:

- t_i: Approximately 11 epochs' worth of training steps to ensure proper warm-up for all trainable vectors.
- t_f: Approximately 1 epoch's worth of training steps to allow for significant updates to the trainable vectors.
- $k \leq 5$: As this value is generally observed to yield the best performance with stable training.

C.1 Natural Language Understanding

Table 9 gives the hyperparameters used for each task in GLUE benchmark. We experimented using the following learning rates ($1e-2, 1e-3, 1e-4, 3e-4, 5e-4$) and observed that $1e-3$ works best for all tasks in GLUE.

Table 9. Hyperparameter setup of VectorFit for GLUE benchmark.

Dataset	learning rate	epochs	batch size	t_i	t_f	n_f	k
MNLI	$1e-03$	20	32	135000	10000	5	5
SST2	$1e-03$	30	32	23200	2100	10	5
COLA	$1e-03$	35	32	3000	200	5	5
QQP	$1e-03$	25	32	125100	11000	10	5
QNLI	$1e-03$	25	32	36000	3200	10	5
RTE	$1e-03$	50	32	800	70	27	5
MRPC	$1e-03$	50	32	1260	110	27	5
STSB	$1e-03$	50	32	1900	180	27	5

Table 10 presents the hyperparameters related to budget allocation of the baselines. d is the hidden dimension for the adapters, r is the rank of LoRA incremental weight matrices, and $b^{(T)}$ is the target budget of AdaLoRA. We use SVFT with random setting and $d = 2$.

Table 10. Budget setup of baselines for GLUE benchmark.

# Params	Houlsby Adapter (d)	Pfeiffer Adapter (d)	LoRA (r)	AdaLoRA ($b^{(T)}$)
1.2M	32	64	8	576
0.6M	16	32	4	288
0.3M	8	16	2	144

C.2 Question Answering

Table 11 gives the hyperparameters used for each dataset of QA task. We experimented using the following learning rates ($1e-2, 1e-3, 1e-4, 3e-4, 5e-4$) and observed that $1e-3$ works best for both datasets of QA task.

Table 11. Hyperparameter setup of VectorFit for question answering tasks.

Dataset	learning rate	epochs	batch size	t_i	t_f	n_f	k
Squad v1.1	$1e-03$	20	16	60700	5500	6	5
Squad v2.0	$1e-03$	20	16	90300	8200	6	5

Table 12 presents the hyperparameters related to budget allocation of the baselines.

Table 12. Budget setup of baselines for QA tasks.

# Params	Houlsby Adapter (d)	Pfeiffer Adapter (d)	LoRA (r)	AdaLoRA ($b^{(T)}$)	SVFT (d)
0.08%	4	8	1	72	1

C.3 Natural Language Generation

Table 13 gives the hyperparameters used for each dataset of NLG task. We experimented using the following learning rates ($1e-2, 1e-3, 1e-4, 3e-4, 5e-4$) and observed that $1e-3$ works best for both datasets of NLG task.

Table 13. Hyperparameter setup of VectorFit for natural language generation tasks.

Dataset	learning rate	epochs	batch size	t_i	t_f	n_f	k
XSum	$1e-03$	30	64	35070	3100	10	5
CNN/Dailymail	$1e-03$	30	64	31500	4400	10	5

Table 14 presents the hyperparameters related to budget allocation of the baselines used for experiments with Xsum and CNN/Dailymail datasets for NLG task.

Table 14. Budget setup of baselines for NLG tasks.

Houlsby Adapter (d)	Pfeiffer Adapter (d)	LoRA (r)	AdaLoRA ($b^{(T)}$)	SVFT (d)
8	16	2	144	2

C.4 Image Classification

Table 15 gives the hyperparameters used for each dataset of image classification task. We experimented using the following learning rates ($1e-2, 1e-3, 1e-4, 3e-4, 5e-4$) and the best performing learning rates are given in the table.

Table 15. Hyperparameter setup of VectorFit for image classification tasks.

Dataset	learning rate	epochs	batch size	t_i	t_f	n_f	k
CIFAR10	$1e-03$	20	128	3600	300	4	5
GTSRB	$1e-03$	20	128	1900	170	4	5
MNIST	$1e-02$	20	128	4300	350	4	5
RESISC45	$1e-02$	20	128	2300	200	4	5

For the baselines, we use LoRA with $r = 2$, AdaLoRA with $b^{(T)} = 144$, and SVFT with $d = 2$.

C.5 Image Generation

Dreambooth fine-tuning for various subjects in the dataset were done using prior preservation loss with the weightage varying between 0.5 to 1.0 depending on the subject. We use 300 class images for each subject, a learning rate of $5e-5$, and a batch size of 4. We use the rank of 2 for fine-tuning with LoRA.

D Additional Experiments

D.1 Training Strength Ablation

Figure 7 shows the training strength heatmap of various trainable vectors for different variants of VectorFit. We can observe that VectorFit with AVF (top-right) achieves the most equitable training possible among the trainable vectors and hence maintains an overall lower training strength. We can also observe that as the number of trainable vectors is reduced, the training strength of the vectors increases to make up for the reduced number of trainable parameters.

D.2 NLU Tasks Ablation

Figure 8 shows the ablation graphs for the GLUE benchmark with all five variants of our method. The graphs show the efficacy of AVF where VectorFit with AVF gives a higher performance on all the datasets.

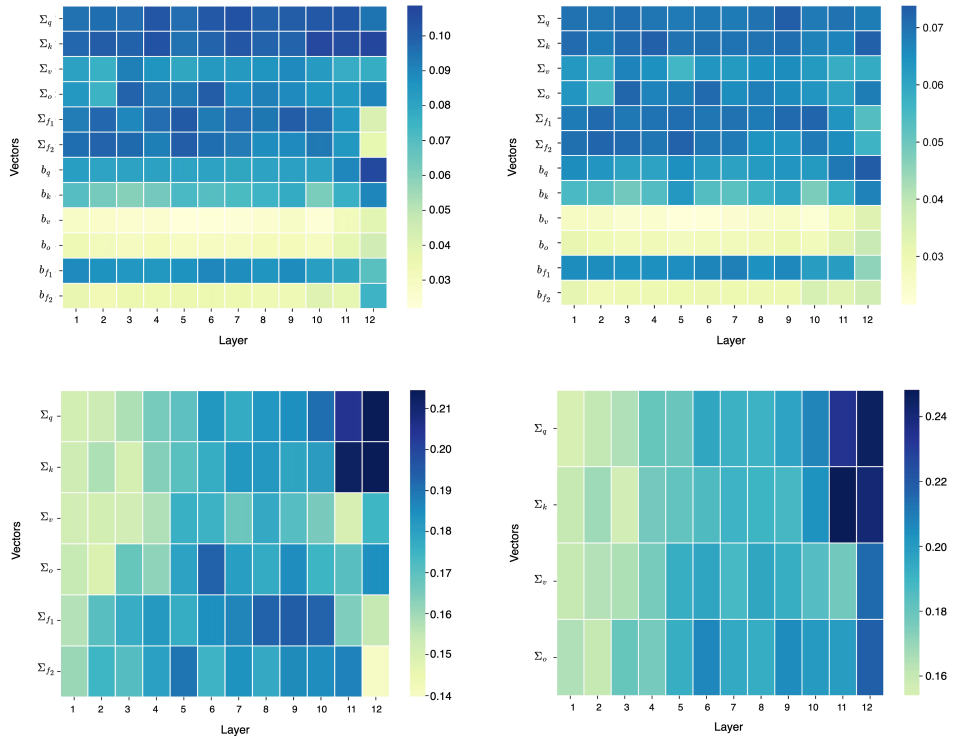

Figure 7. The training strength S_v of each trainable vector after fine-tuning of DeBERTaV3-base on the COLA dataset is shown for VectorFit without AVF (top-left), VectorFit with AVF (top-right), VectorFit (Σ) (bottom-left), and VectorFit (Σ_a) (bottom-right). The x-axis represents the layer index, while the y-axis corresponds to different types of trainable vectors.

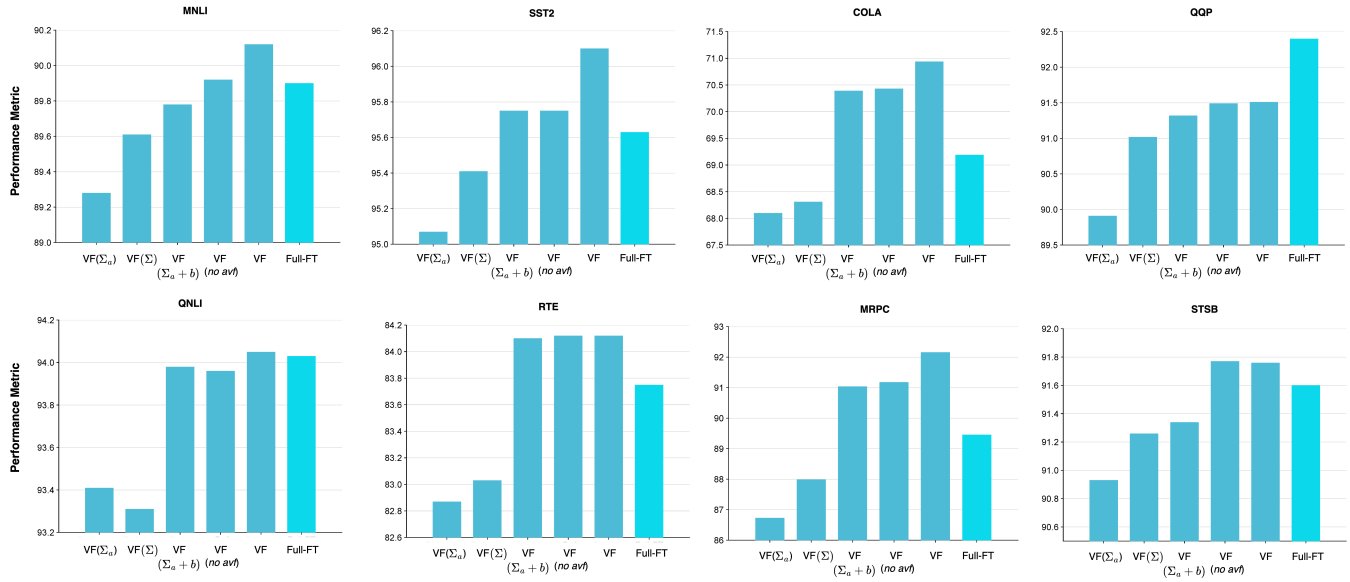

Figure 8. Ablation study about AVF and different trainable vectors configuration on the GLUE benchmark. We report the matched accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for the other tasks.

D.3 QA Tasks Ablation

Table 16 presents the performance of various VectorFit variants. Notably, the most parameter-efficient version, VectorFit(Σ_a), which uses only 0.01% of trainable parameters, achieves up to 98% of the F1 score obtained with Full-FT.

Table 16. Ablation study on QA.

Model	# Params	Squad v1.1 (EM/F1)	Squad v2.0 (EM/F1)
VectorFit (Σ_a)	0.01%	83.8/91.0	80.2/83.9
VectorFit (Σ)	0.02%	84.9/91.9	81.6/85.0
VectorFit ($\Sigma_a + b$)	0.07%	86.4/92.6	83.7/86.6
VectorFit (*no avf*)	0.08%	86.7/93.0	84.2/87.1
VectorFit	0.08%	87.0/93.2	84.4/87.6

D.4 Rank Analysis Continued

Figure 10 presents the singular value distributions of the Δ^* matrices discussed in Section 6.2. For the DeBERTaV3-base model, each singular vector is 768-dimensional, and all 768 singular values are plotted. The graphs reveal that Δ^* for Full-FT is not inherently low-rank, as even the smallest singular values remain non-zero for many weight matrices. Additionally, the plots demonstrate that VectorFit achieves high-rank adaptation, closely approximating Full-FT for several weight matrices. Figure 9 shows the singular values of Δ^* of ViT-base model fine-tuned with VectorFit on CIFAR10 dataset.

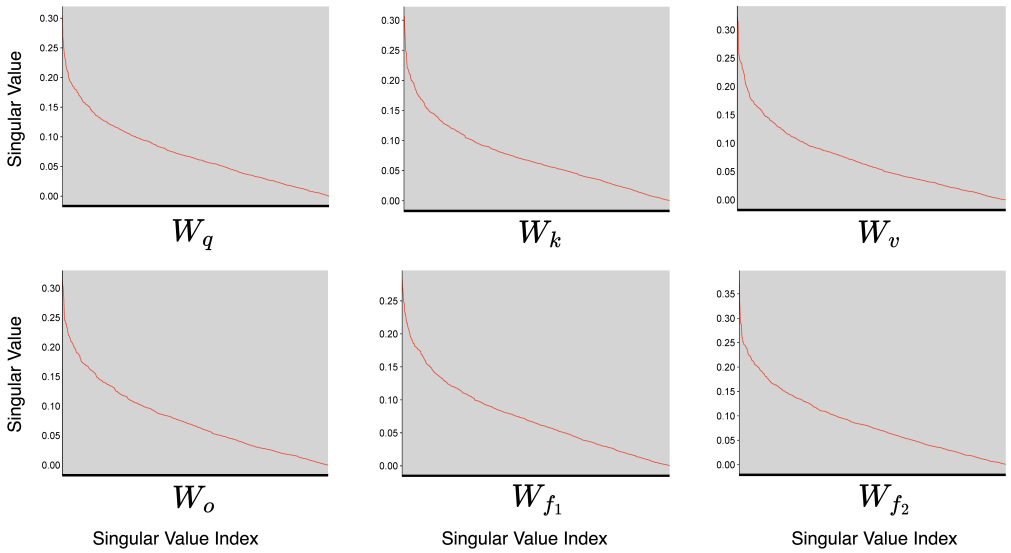

Figure 9. Singular value graphs of Δ^* for all the modules of a randomly picked layer (layer 6) of ViT-base model fine-tuned with VectorFit on CIFAR10 dataset. X-axis represents the singular value position/index and Y-axis represents the singular value.

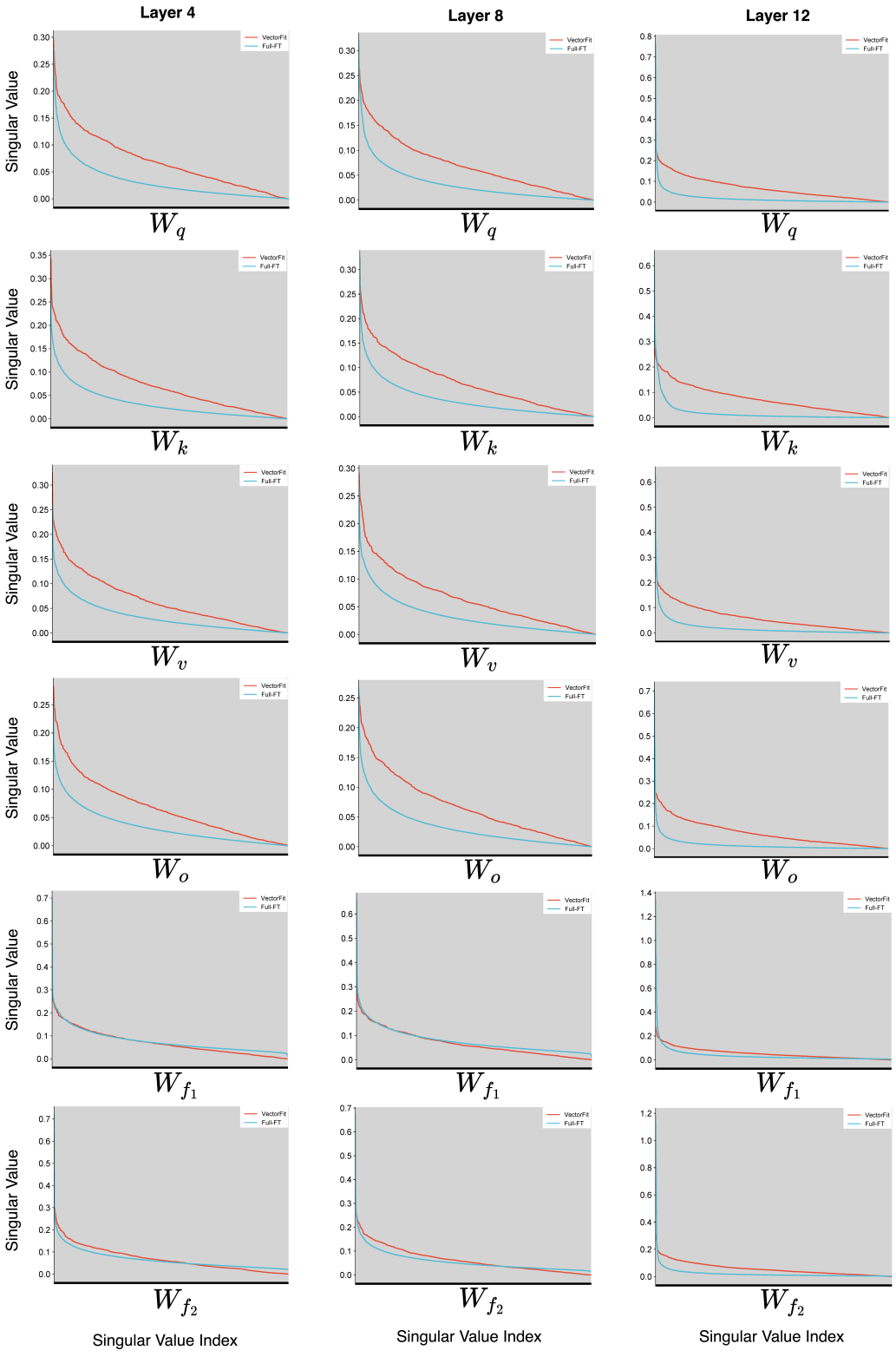

Figure 10. Singular value graphs of Δ^* for all the modules of randomly picked layers in case of VectorFit and Full-FT. X-axis represents the singular value position/index and Y-axis represents the singular value.

D.5 Weight Transformation During VectorFit Fine-Tuning

The heatmap of variation of the first 64 singular values before and after full fine-tuning in each singular vector of randomly selected layers is displayed in Figure 11. This depicts the weight matrix's stretching in its multi-dimensional hyper-space. It should be noted that upon

fine-tuning, even the least significant singular directions might become the principal singular directions. This shows that VectorFit is highly expressive.

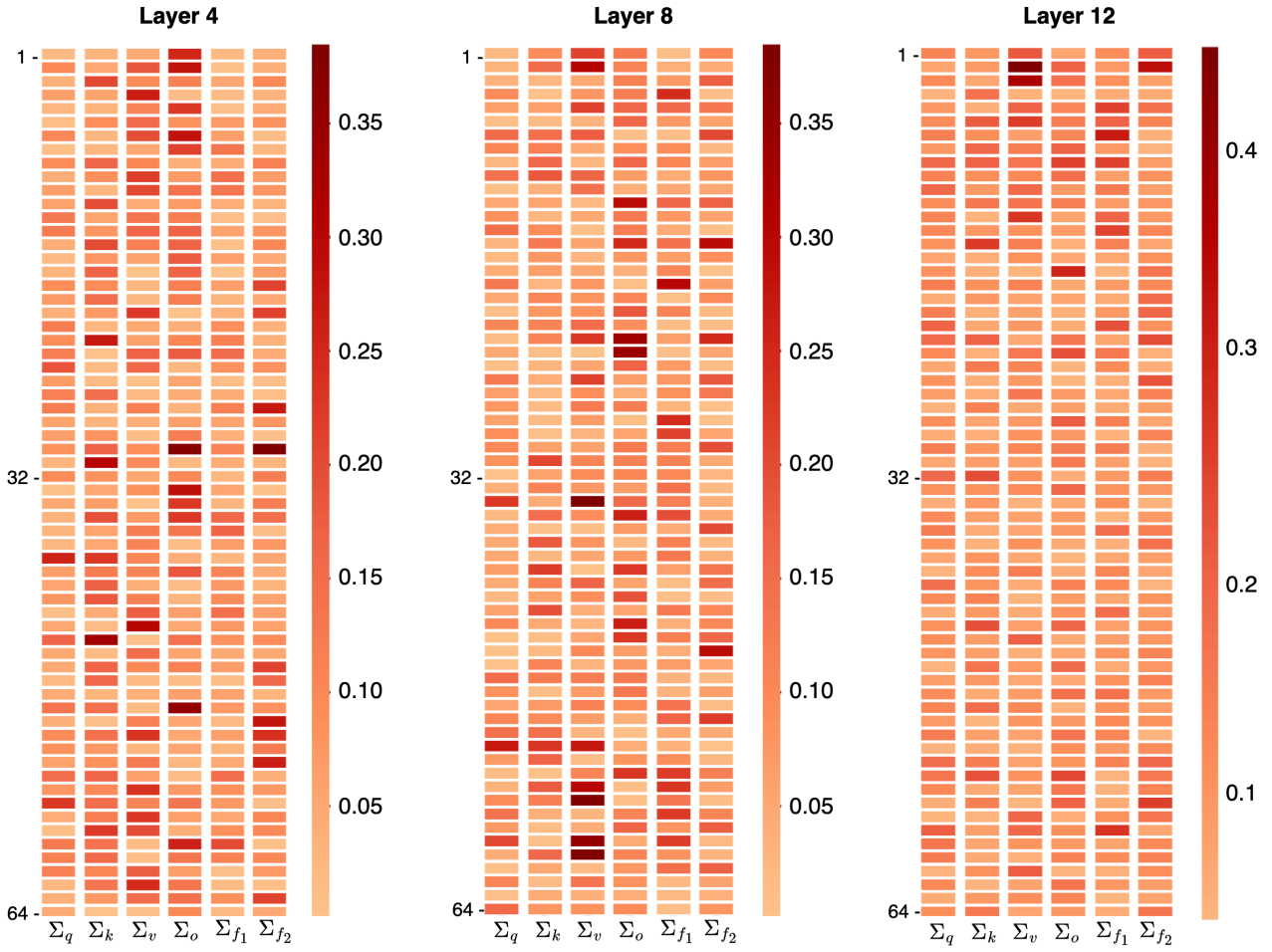

Figure 11. Heatmap representing the variations in first 64 singular values of different singular vectors before and after fine-tuning. The heatmaps are generated with randomly picked layers of DeBERTaV3-base model fine-tuned using VectorFit on COLA dataset.

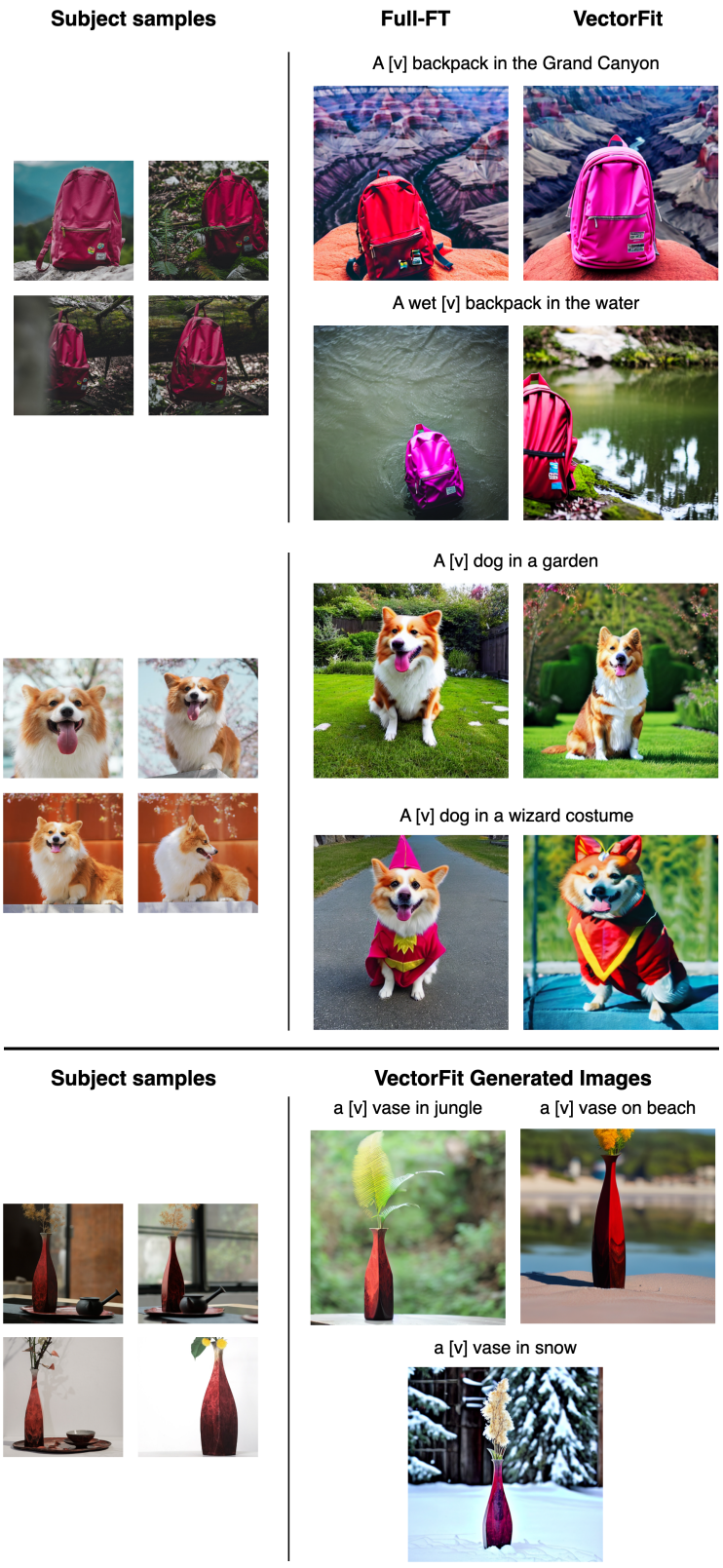

Figure 12. Visual comparison of images generated by Stable Diffusion v1.4 fine-tuned with VectorFit and Full-FT methods using Dreambooth approach for Subject-driven Image generation.