GIVA: Gradient-Informed Bases for Vector-Based Adaptation

Neeraj Gangwar[†] **Rishabh Deshmukh**[§] **Michael Shavlovsky**[§] **Hancao Li**[§]
Vivek Mittal[§] **Lexing Ying**[¶] **Nickvash Kani**[†]

[†]University of Illinois Urbana-Champaign [§]Amazon [¶]Stanford University
gangwar2@illinois.edu, derishab@amazon.com

Abstract

As model sizes continue to grow, parameter-efficient fine-tuning has emerged as a powerful alternative to full fine-tuning. While LoRA is widely adopted among these methods, recent research has explored vector-based adaptation methods due to their extreme parameter efficiency. However, these methods typically require substantially higher ranks than LoRA to match its performance, leading to increased training costs. This work introduces GIVA, a gradient-based initialization strategy for vector-based adaptation. It achieves training times comparable to LoRA and maintains the extreme parameter efficiency of vector-based adaptation. We evaluate GIVA across diverse benchmarks, including natural language understanding, natural language generation, and image classification. Experiments show that our approach consistently outperforms or achieves performance competitive with existing vector-based adaptation methods and LoRA while reducing rank requirements by a factor of eight (8×).

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable success through large-scale pre-training, enabling them to learn rich representations of language and general world knowledge (e.g., OLMo et al., 2024; Liu et al., 2024a; Gemma Team et al., 2024; Achiam et al., 2023; Qwen Team, 2024; Abdin et al., 2024). However, to fully leverage their capabilities in real-world applications, these models typically need to be fine-tuned to adapt them effectively to various complex downstream tasks, such as mathematical and commonsense reasoning, code understanding and generation, etc. Fully fine-tuning these models for each application is extremely costly, especially as model sizes continue to grow, making it computationally expensive. Moreover, each fine-tuned copy requires the same amount of storage as the original model, which is further amplified by the need to store multiple checkpoints.

To address these challenges, parameter-efficient fine-tuning (PEFT) has emerged as a powerful alternative, offering a better trade-off between the computational cost and performance (e.g., Hu et al., 2022; Li and Liang, 2021; Lester et al., 2021; Houlsby et al., 2019; Sung et al., 2021). Recent extensions have further advanced this line of work (e.g., Zhang et al., 2023; Liu et al., 2024b; Kopiczko et al., 2024; Albert et al., 2025). Among these methods, LoRA (Hu et al., 2022) has become one of the most widely adopted approaches for adapting LLMs to various downstream tasks. It leverages the observation from Aghajanyan et al. (2021) that parameter updates often lie in a low-dimensional subspace and models them as a product of two low-rank matrices, significantly reducing the number of trainable parameters.

Vector-based adaptation methods (Figure 1) extend this idea by introducing scaling vectors that adapt a pair of frozen matrices (also referred to as bases) to guide the model's behavior (Kopiczko et al., 2024; Han et al., 2025). Since only the scaling vectors are trained, they reduce the number of trainable parameters to an even greater extent than LoRA-like approaches, making them extremely parameter- and storage-efficient. This efficiency is particularly appealing in resource-constrained applications, such as scenarios where model updates must be communicated over a network (e.g., federated learning or multi-device fine-tuning), and in mixture-of-experts (e.g., Wu et al., 2024). However, to match LoRA's performance, these methods typically require substantially higher ranks, which leads

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

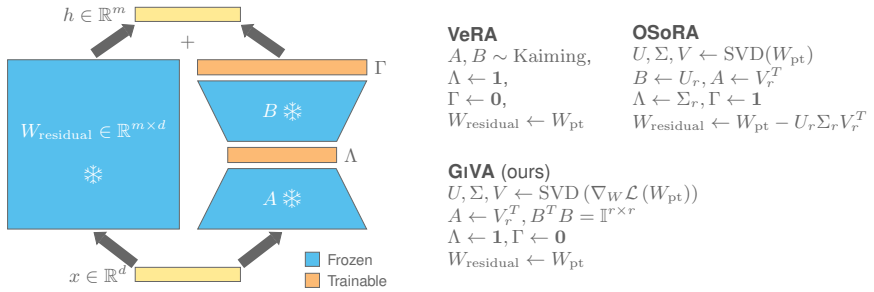

Figure 1: Overview of vector-based adaptation methods. Here, W_{pt} denotes the pre-trained weights, U_r and V_r denote the first r columns of U and V, respectively, and Σ_r corresponds to the top-left $r \times r$ submatrix of Σ.

to considerably longer training times as model sizes increase. For instance, fine-tuning Qwen 2 (0.5B) on 15K commonsense reasoning examples from Hu et al. (2023) using VeRA (Kopiczko et al., 2024) requires approximately 2.5× the runtime of LoRA to achieve comparable performance (see Section 4.4 for details). This additional overhead is primarily due to VeRA's higher rank—1024 versus 16 in LoRA. Motivated by this, we focus on the following question:

Can we reduce the required rank in vector-based adaptation methods to achieve training times comparable to LoRA while preserving their performance?

VeRA (Kopiczko et al., 2024) uses a pair of random matrices shared across layers as its bases, whereas OSoRA (Han et al., 2025) derives the bases directly from pre-trained weights. As these bases are not updated during fine-tuning and contain no information about the downstream task, both methods require higher ranks than LoRA to achieve comparable performance. The choice of bases may be particularly suboptimal for OSoRA, as the reliance on pre-trained weights makes it less flexible than VeRA. We hypothesize that carefully choosing the bases may reduce the required ranks while achieving performance comparable to these methods. To this end, we propose gradient-informed bases for vector-based adaptation (GIVA). Our approach constructs its bases using the first-step full fine-tuning gradient for the downstream task. Specifically, the bases are chosen such that the first-step gradient of GIVA closely approximates that of full fine-tuning. This strategy is analogous to those adopted in recent methods for improving LoRA (Wang et al., 2024, 2025). However, unlike these approaches, we fine-tune only the scaling vectors, thereby preserving the extreme parameter efficiency of vector-based adaptation methods.

We extensively evaluate GIVA across natural language understanding, natural language generation, and image classification tasks. Our experiments demonstrate that it consistently outperforms or achieves performance competitive with existing vector-based adaptation methods and LoRA while operating with substantially lower ranks. As a result, it attains runtimes on par with LoRA and requires an extremely small number of trainable parameters.

Contributions. The contributions of our work are as follows:

- We introduce GIVA, a vector-based adaptation method that derives the bases from the first-step full fine-tuning gradient, enabling efficient adaptation with reduced rank requirements. The reduced rank requirements result in training times comparable to those of LoRA-like methods.
- We evaluate our approach extensively across various benchmarks and model sizes, which show that GIVA outperforms or achieves performance competitive with existing methods.
- We examine three alternative strategies for initializing the bases, each leading to similar performance across benchmarks.

Our code is publicly available on GitHub.[1]

2 RELATED WORKS

Parameter Efficient Fine-Tuning. Early PEFT methods introduce a small set of trainable parameters and keep the original model frozen during training. One line of work inserts adapter modules between different layers of a pre-trained model (Houlsby et al., 2019; Lin et al., 2020; Pfeiffer et al., 2021), while other methods focus on adding learnable prompts to the model input. For example, AutoPrompt (Shin et al., 2020) searches for a token sequence that elicits the desired behavior. Prompt Tuning (Lester et al., 2021) and Prefix Tuning (Li and Liang, 2021) extend this idea by learning continuous prefix embeddings rather than

[1] `https://github.com/neerajgangwar/giva`

discrete tokens. Although effective, these methods increase the inference latency due to the additional layers or tokens. LoRA (Hu et al., 2022) addresses this limitation by modeling the parameter updates as a product of two low-rank matrices, which can be merged back into the base model after training, thus eliminating any additional inference cost. Several works have proposed alternative initialization schemes for LoRA (Büyükakyüz, 2024; Yang et al., 2024; Meng et al., 2024). Methods like AdaLoRA (Zhang et al., 2023), EVA (Paischer et al., 2024), GoRA (He et al., 2025), and GeLoRA (Ed-dib et al., 2024) decide the ranks for different layers in the model dynamically rather than using a fixed rank across layers. To achieve stable learning, LoRA+ (Hayou et al., 2024) uses different learning rates for the low-rank matrices, while Zhang and Pilanci (2024) propose the use of Riemannian preconditioning. Methods such as FLoRA (Hao et al., 2024) and XGBLoRA (Zhang et al., 2024), among others, have proposed full-rank variants of LoRA. Another line of research in parameter-efficient methods focuses on preserving the hyperspherical energy during fine-tuning. For example, Orthogonal Fine-Tuning (OFT; Qiu et al., 2023) targets text-to-image tasks. BOFT (Liu et al., 2024c) and MOFT (Wu et al., 2025) are its parameter- and memory-efficient extensions.

Improved Parameter Efficiency. Several methods have improved parameter efficiency by introducing learnable scaling vectors in LoRA-inspired frameworks. For example, NoLA (Koohpayegani et al., 2024) represents LoRA's low-rank matrices as a weighted combination of random bases, training only the weights. This strategy achieves comparable performance to LoRA while substantially reducing trainable parameters. VeRA (Kopiczko et al., 2024) employs trainable scaling vectors to adapt a pair of randomly initialized matrices to control the model behavior. Instead of using random bases, OSoRA (Han et al., 2025) derives its bases from pre-trained weights. RandLoRA (Albert et al., 2025) extends this strategy to support full-rank updates while preserving parameter efficiency. Differing from these methods, LoRA-XS (Bałazy et al., 2024) introduces a trainable weight matrix between the frozen low-rank matrices, which are derived from pre-trained weights.

Leveraging Gradients in PEFT. Recent studies have leveraged the full fine-tuning gradient for better initialization of the low-rank matrices. LoRA-GA (Wang et al., 2024) and LoRA-One (Zhang et al., 2025) use the first-step full fine-tuning gradient for initialization and achieve faster convergence across tasks. LoRA-SB (Ponkshe et al., 2024) extends LoRA-XS and uses gradients to initialize the frozen low-rank matrices. LoRA-Pro (Wang et al., 2025) adjusts the gradients of the low-rank matrices to better approximate those of full fine-tuning.

3 METHOD

3.1 Preliminaries

For a pre-trained weight matrix $W_{\text{pt}} \in \mathbb{R}^{m\times d}$, vector-based adaptation methods parameterize the weight update ΔW and the updated weight matrix $W' \in \mathbb{R}^{m\times d}$ as

$$\begin{aligned} \Delta W &= \Gamma B \Lambda A \\ W' &= W_{\text{pt}} + \Delta W = W_{\text{pt}} + \Gamma B \Lambda A \end{aligned} \tag{1}$$

Here, $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times d}$ are low-rank matrices; further $\Gamma = \text{diag}(\gamma_1 \dots \gamma_m)$ and $\Lambda = \text{diag}(\lambda_1 \dots \lambda_r)$. Rank r is generally chosen such that $r < \min(m, d)$. During training, only the diagonal matrices are updated, resulting in $m + r$ trainable parameters. Hence, these methods are extremely parameter-efficient.

VeRA (Kopiczko et al., 2024) initializes A and B as random matrices shared across layers, while OSoRA (Han et al., 2025) derives A and B from the singular-value decomposition (SVD) of W_{pt}.

3.2 Proposed Approach

In this section, we introduce GIVA. We hypothesize that carefully chosen A and B may allow for smaller values of r than existing vector-based adaptation methods while achieving comparable performance.

We employ a gradient-based initialization for A and B. In our method, A and B are chosen such that the first-step update closely approximates the first-step full fine-tuning update.

First-Step Update. The weight update ΔW in (1) can be written as

$$\begin{aligned} \Delta W &= (\Gamma_0 + \Delta\Gamma) B (\Lambda_0 + \Delta\Lambda) A \\ &= \Gamma_0 B \Lambda_0 A + \Gamma_0 B \Delta\Lambda A + \Delta\Gamma B \Lambda_0 A + \Delta\Gamma B \Delta\Lambda A \end{aligned}$$

Here, a subscript of zero denotes the initial value of a variable. We use $\Gamma_0 = \mathbf{0}^{m\times m}$ and $\Lambda_0 = \mathbb{I}^{r\times r}$, which reduces ΔW to

$$\Delta W = \Delta\Gamma B A + \Delta\Gamma B \Delta\Lambda A \tag{2}$$

The first-step update can be computed by substituting $\Delta\Gamma = \eta \nabla_\Gamma \mathcal{L}(\Gamma_0)$ and $\Delta\Lambda = \eta \nabla_\Lambda \mathcal{L}(\Lambda_0)$. Here, η and $\mathcal{L}$ denote the learning rate and the loss function, respectively. The second term in (2) contains two delta terms, resulting in a factor of η^2 and a product of

two gradients. We ignore this term and rewrite the first-step update as

$$\eta \nabla_\Gamma \mathcal{L}(\Gamma_0) BA$$

Computing A and B. Let $\eta \nabla_W \mathcal{L}(W_{\text{pt}})$ denote the first-step full fine-tuning update. We choose A and B such that

$$\arg\min_{A,B} \|\eta \nabla_\Gamma \mathcal{L}(\Gamma_0) BA - \eta \nabla_W \mathcal{L}(W_{\text{pt}})\|_F \quad (3)$$

$\nabla_\Gamma \mathcal{L}$ can be written in terms of $\nabla_{W'} \mathcal{L}$, resulting in

$$\nabla_\Gamma \mathcal{L}(\Gamma_0) = \nabla_{W'} \mathcal{L}(W_{\text{pt}}) A^T B^T$$

As $\nabla_{W'} \mathcal{L}(W_{\text{pt}}) = \nabla_W \mathcal{L}(W_{\text{pt}})$ (Lemma 3.1 from Wang et al., 2024), the optimization problem in (3) can be rewritten as

$$\arg\min_{A,B} \|\nabla_W \mathcal{L}(W_{\text{pt}}) A^T B^T BA - \nabla_W \mathcal{L}(W_{\text{pt}})\|_F \quad (4)$$

Theorem 1. *The optimization problem in* (4) *reaches its minimum when*

$$A = V_r^T, B^T B = \mathbb{I}^{r \times r}$$

V_r denotes the right singular vectors of $\nabla_W \mathcal{L}(W_{pt})$, corresponding to the r largest singular values.

Proof. See the supplementary material. □

Using Theorem 1, we initialize the training formulation in (1) as follows

$$\begin{aligned} &\Gamma = \mathbf{0}^{m \times m}, \\ &\Lambda = \mathbb{I}^{r \times r}, \\ &A = V_r^T, \text{ and} \\ &B \text{ such that } B^T B = \mathbb{I}^{r \times r} \end{aligned} \quad (5)$$

Efficiency Trade-Offs. In contrast to existing vector-based adaptation methods, our approach introduces some additional overheads: (1) storage, since matrices A and B must be saved, (2) gradient computation, and (3) performing SVD to obtain A and B. The low-rank matrices need to be stored only once per task, and subsequent checkpoints remain as lightweight as those in existing vector-based adaptation methods. To keep the training pipeline efficient, we compute the first-step full fine-tuning gradient using a single batch of examples and use the low-rank SVD algorithm.[2]

[2]PyTorch implementation: `torch.svd_lowrank`

4 EXPERIMENTS

We conduct extensive experiments to evaluate GIVA on natural language understanding (Section 4.1), natural language generation (Section 4.2), and image classification (Section 4.3). The corresponding sections provide details about the models and datasets used. We conclude by discussing the training times for different adaptation methods in Section 4.4.

For our baselines, we primarily consider vector-based adaptation methods: VeRA (Kopiczko et al., 2024), OSoRA (Han et al., 2025), and RandLoRA (Albert et al., 2025). We additionally include LoRA (Hu et al., 2022), which serves as a widely accepted reference point in PEFT methods. While not all baselines are evaluated across every dataset, we include each method for all tasks where results were reported in its original publication.

We initialize A following (5). As B may be initialized as any matrix with orthonormal columns, we consider the following initialization strategies:

TopSV-B. Left singular vectors of the first-step full fine-tuning gradient corresponding to the r largest singular values. Zhang et al. (2025) use a similar strategy for LoRA.

SecSV-B. Left singular vectors of the first-step full fine-tuning gradient corresponding to the $r+1$ to $2r$ largest singular values. Wang et al. (2024) use a similar strategy for LoRA.

Rand-B. By performing a QR decomposition on a random matrix $X \sim \mathcal{N}(0,1)^{m \times r}$.

4.1 Natural Language Understanding

We begin by evaluating GIVA on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019). Our experiments use both the base and large versions of RoBERTa (Liu et al., 2019). We generally follow the training setup of Albert et al. (2025) and additionally perform a learning rate sweep. Adapter modules are added to all linear projections within the attention and MLP layers. We initialize the training from pre-trained checkpoints and omit the MNLI-trick from Hu et al. (2022) due to resource limitations.

Models are fine-tuned for 10 epochs and evaluated at the end of each epoch. We report the best validation performance. Further details on the training setup are provided in Section B.1 of the supplementary material.

Table 1 shows the performance of various adaptation methods on the validation set. The reported number of trainable parameters does not include the classification

Table 1: Performance of different adaptation methods on GLUE. We report Matthew's correlation for CoLA, Pearson's correlation for STS-B, and accuracy for the remaining tasks. Experiments are repeated three times, and the averages are reported.

Method	**Params** (M)	**SST-2** (Acc.)	**MRPC** (Acc.)	**CoLA** (MCC)	**QNLI** (Acc.)	**RTE** (Acc.)	**STS-B** (PCC)	**Avg.**
				RoBERTa-Base				
Full Fine-Tuning	124.06	94.8±0.2	89.6±0.7	62.1±0.8	92.8±0.1	75.8±1.4	90.9±0.1	84.3
LoRA ($r = 4$)	0.66	94.7±0.1	87.7±0.6	59.2±0.8	92.7±0.1	74.8±0.6	90.5±0.0	83.3
VeRA ($r = 1024$)	0.16	94.5±0.1	88.2±0.9	62.3±0.5	92.5±0.1	73.6±1.3	90.6±0.1	83.6
RandLoRA ($r = 64$)	0.72	94.9±0.1	88.8±0.4	62.6±0.9	92.8±0.2	74.8±1.5	90.6±0.1	84.1
GIVA ($r = 8$, TopSV-B)	0.08	94.2±0.2	88.7±0.5	59.1±0.8	92.3±0.2	75.5±3.9	90.4±0.1	83.4
GIVA ($r = 8$, SecSV-B)	0.08	94.2±0.2	89.4±0.9	60.0±2.3	92.2±0.2	73.4±1.9	90.5±0.0	83.3
GIVA ($r = 8$, Rand-B)	0.08	94.5±0.2	89.3±1.6	61.6±0.3	92.3±0.1	74.8±3.0	90.5±0.2	83.8
				RoBERTa-Large				
Full Fine-Tuning	354.31	96.4±0.1	89.5±0.6	67.5±1.5	94.7±0.1	84.7±1.8	92.1±0.1	87.5
LoRA ($r = 4$)	1.77	96.2±0.2	89.7±0.4	66.6±0.5	94.7±0.0	85.2±1.0	92.2±0.1	87.4
VeRA ($r = 256$)	0.26	96.1±0.1	89.8±0.4	68.6±1.8	94.8±0.2	83.2±2.2	92.0±0.2	87.4
RandLoRA ($r = 100$)	1.78	96.5±0.5	90.1±0.4	67.7±1.4	94.7±0.3	85.3±0.9	92.3±0.2	87.8
GIVA ($r = 8$, TopSV-B)	0.22	96.2±0.2	90.3±0.7	67.5±1.0	94.7±0.1	84.5±1.1	91.6±0.5	87.5
GIVA ($r = 8$, SecSV-B)	0.22	96.1±0.4	89.4±0.4	68.0±2.2	95.0±0.1	82.3±2.8	91.6±0.2	87.1
GIVA ($r = 8$, Rand-B)	0.22	96.1±0.3	89.8±1.2	67.0±0.9	94.7±0.2	85.3±0.6	92.0±0.4	87.5

heads. These results demonstrate that GIVA achieves performance comparable to other methods. Notably, GIVA outperforms VeRA and uses a rank of eight, substantially reducing the computational overhead during training. All three initializations for GIVA yield comparable results on GLUE, but Rand-B achieves the highest average performance across model sizes.

4.2 Natural Language Generation

We next evaluate GIVA on natural language generation tasks. For this purpose, we consider four benchmarks: (1) commonsense reasoning, (2) mathematical reasoning, (3) code generation, and (4) instruction tuning.

Commonsense Reasoning. We fine-tune Qwen 2 (0.5B; Qwen Team, 2024), Phi 3 (3.8B; Abdin et al., 2024), and OLMo 2 (7B; OLMo et al., 2024) on a subset of 15K commonsense reasoning examples from Hu et al. (2023). We largely follow the training setup of Albert et al. (2025), with the addition of a learning rate sweep. The official implementation of RandLoRA suggests that Albert et al. (2025) compute the training loss over both the input and output tokens. In our experiments, we restrict the loss computation to the output tokens, which yields better performance across methods. Adapter modules are added to the attention layers' query, key, and value projections, as well as the up and down projections of the MLP layers. For Phi 3, they are also added to the MLP gate projections.

We fine-tune the models for three epochs. During training, the models are evaluated four times per epoch, and the best checkpoint is selected based on the validation loss. See Section B.2 of the supplementary material for further details on the training setup.

The fine-tuned models are evaluated on the following commonsense reasoning datasets: (1) BoolQ (Clark et al., 2019), (2) PIQA (Bisk et al., 2020), (3) Social IQa (Sap et al., 2019), (4) HellaSwag (Zellers et al., 2019), (5) WinoGrande (Sakaguchi et al., 2020), (6) ARC (Clark et al., 2018), and (7) OpenBookQA (Mihaylov et al., 2018). We use beam search with a beam size of four and generate a maximum of 32 tokens.

Table 2 presents the performance of various adaptation methods. The parameter percentages are shown relative to the total model size. These results demonstrate that GIVA achieves performance comparable to other methods and surpasses both VeRA and OSoRA despite using a rank of 64 across models. In addition, the three initialization strategies for GIVA yield broadly comparable results on commonsense reasoning tasks.

Unlike Albert et al. (2025), we do not observe a significant performance gap between VeRA and other methods. We believe the larger performance gap observed by Albert et al. (2025) for VeRA can be explained by the choice of the d_{initial} hyperparameter, in addition to the difference in the training loss computation.

Mathematical Reasoning. We fine-tune OLMo 2 (7B) on 100K examples from MetaMathQA (Yu et al., 2024) and use an additional 10K examples for validation. Adapter modules are added to the attention layers' query, key, and value projections, as well as

Table 2: Performance of different adaptation methods on commonsense reasoning datasets. We report accuracy for all tasks. The percentages are computed relative to the total number of parameters. Experiments are repeated three times, and their averages are reported.

Method	**Params** (%)	**BoolQ**	**PIQA**	**SIQA**	**HS**	**WG** (Acc.)	**ARC-e**	**ARC-c**	**OBQA**	**Avg.**
					Qwen2-0.5B-Instruct					
LoRA ($r = 16$)	1.180	62.0±0.3	66.1±1.6	61.3±0.1	54.0±2.1	51.2±1.8	66.6±0.2	49.1±2.3	60.7±1.0	58.9
VeRA ($r = 1024$)	0.058	62.4±0.4	65.5±0.2	60.5±0.4	47.6±1.0	53.3±1.3	64.6±0.5	45.7±0.7	57.1±1.4	57.1
OSoRA ($r = 1024$)	0.048	62.0±0.3	62.0±0.6	58.5±1.0	40.4±1.3	50.9±0.8	62.7±0.6	43.9±0.8	57.3±1.0	54.7
RandLoRA ($r = 10$)	1.191	61.4±1.8	65.6±1.7	57.6±1.9	44.1±5.5	50.2±0.4	64.6±0.9	45.9±2.9	55.8±2.8	55.7
GIVA ($r = 64$, TopSV-B)	0.035	62.6±0.3	66.2±0.7	59.3±0.2	48.8±0.6	51.6±0.5	66.4±0.1	47.6±1.6	54.7±0.4	57.1
GIVA ($r = 64$, SecSV-B)	0.035	61.7±1.0	66.0±0.8	60.2±1.0	50.4±0.7	50.6±0.5	65.8±0.6	48.5±0.3	57.8±1.0	57.6
GIVA ($r = 64$, Rand-B)	0.035	62.2±0.0	65.6±0.7	60.4±0.4	50.7±1.3	51.9±0.5	65.7±1.0	48.7±0.5	55.7±0.9	57.6
					Phi3-Mini					
LoRA ($r = 16$)	0.573	69.9±0.6	86.4±0.1	77.2±0.5	90.4±0.5	80.4±0.3	96.1±0.4	88.1±0.5	87.7±0.6	84.5
VeRA ($r = 1024$)	0.027	69.2±1.0	85.8±0.2	76.5±0.1	89.1±1.7	79.4±1.2	95.5±0.4	86.9±0.3	86.1±1.6	83.6
OSoRA ($r = 1024$)	0.027	68.4±0.5	85.8±0.5	77.8±0.2	88.7±0.4	78.2±0.1	95.7±0.2	87.1±0.3	88.9±1.2	83.8
RandLoRA ($r = 40$)	0.598	69.8±0.1	86.7±0.3	76.5±0.6	90.1±1.0	80.1±0.2	95.6±0.3	87.7±0.3	87.1±0.8	84.2
GIVA ($r = 64$, TopSV-B)	0.024	68.5±0.8	87.0±0.3	76.3±1.0	90.1±0.8	78.8±0.6	95.6±0.1	87.6±0.5	88.1±1.6	84.0
GIVA ($r = 64$, SecSV-B)	0.024	69.4±0.2	85.4±0.3	76.8±0.1	89.7±0.9	79.0±0.9	95.3±0.2	87.1±0.7	87.5±0.1	83.8
GIVA ($r = 64$, Rand-B)	0.024	68.5±1.1	86.3±0.5	76.0±1.1	89.5±1.5	78.0±1.2	95.5±0.2	87.4±0.3	86.7±1.8	83.5
					OLMo2-7B					
LoRA ($r = 16$)	0.383	71.0±0.2	86.7±0.1	78.4±0.6	92.3±0.4	80.0±0.2	93.0±0.2	83.0±0.1	85.3±0.5	83.7
VeRA ($r = 1024$)	0.014	70.5±0.8	85.8±0.6	77.7±0.4	91.0±0.2	77.8±0.6	92.7±0.1	81.7±0.5	85.3±0.9	82.8
OSoRA ($r = 1024$)	0.014	69.4±1.1	85.7±0.9	78.1±0.7	90.8±1.1	78.2±1.3	92.0±0.2	81.3±0.4	85.3±1.3	82.6
RandLoRA ($r = 100$)	0.376	70.1±1.6	86.3±0.2	77.5±0.8	89.7±2.8	78.3±2.9	92.8±0.2	83.7±0.1	85.9±0.6	83.0
GIVA ($r = 64$, TopSV-B)	0.012	71.0±0.4	86.8±0.3	78.3±0.6	91.2±0.6	78.7±0.9	93.2±0.4	82.5±0.4	84.1±0.9	83.2
GIVA ($r = 64$, SecSV-B)	0.012	70.8±1.1	86.1±0.9	79.0±0.6	91.3±1.4	78.5±1.3	93.1±0.2	82.8±0.5	84.7±0.6	83.3
GIVA ($r = 64$, Rand-B)	0.012	71.2±0.5	86.7±0.4	78.1±1.1	91.0±0.6	78.7±0.7	92.7±0.4	82.5±0.7	84.4±2.3	83.2

the up and down projections of the MLP layers. The models are fine-tuned for one epoch. During training, the validation is performed 10 times, and the best checkpoint is selected based on the validation loss. Further details on the training setup are provided in Section B.3 of the supplementary material.

The fine-tuned models are evaluated on the GSM8k test set (Cobbe et al., 2021). We use greedy decoding and generate a maximum of 256 tokens. Accuracy is computed by matching the final answer with the ground truth.

Results for these experiments are presented in Table 3. These results show that GIVA surpasses both VeRA and OSoRA despite using a rank of only 64. Similar to earlier experiments, the three initializations for GIVA achieve comparable performance, with Rand-B performing the best.

Code Generation. We fine-tune OLMo 2 (7B) on 100K examples from Code-Feedback (Zheng et al., 2024) and use an additional 10K examples for validation. We use the same training setup as used for mathematical reasoning experiments.

The fine-tuned models are evaluated on HumanEval (Chen et al., 2021). We use beam search with a beam size of four and generate a maximum of 256 tokens. For scoring the model generations, we use the official implementation.[3]

Table 3 shows the performance of various adaptation methods. The results indicate that GIVA matches the performance of VeRA and OSoRA while requiring substantially lower ranks. For code generation, Rand-B achieves the strongest results, while the other two initializations outperform VeRA. Additionally, we observe that LoRA outperforms vector-based adaptation methods by a considerable margin.

Instruction Tuning. Instruction tuning adapts pre-trained language models to follow instructions more effectively. For our experiments, we fine-tune Mistral (7B; Jiang et al., 2023) on the Alpaca dataset (Taori et al., 2023). We largely follow the training setup of Kopiczko et al. (2024) and additionally perform a learning rate sweep. We use a single random seed for all methods. Adapter modules are added to all linear

[3] `https://github.com/openai/human-eval`

Table 3: Performance of fine-tuned OLMo 2 (7B) on mathematical reasoning and code generation with different adaptation methods. We report accuracy for GSM8k and pass@1 for HumanEval. Experiments are repeated three times, and the averages are reported.

Method	Params (M)	GSM8k (Acc.)	HEval (pass@1)
LoRA ($r = 8$)	14.02	70.9$\pm$0.7	27.6$\pm$2.3
VeRA ($r = 512$)	0.96	70.4$\pm$0.8	20.1$\pm$1.8
OSoRA ($r = 512$)	0.96	69.7$\pm$0.6	23.4$\pm$0.9
GiVA ($r = 64$, TopSV-B)	0.89	70.3$\pm$0.9	22.0$\pm$1.6
GiVA ($r = 64$, SecSV-B)	0.89	70.6$\pm$0.4	22.2$\pm$1.3
GiVA ($r = 64$, Rand-B)	0.89	70.8$\pm$0.8	23.2$\pm$0.6

projections within the attention and MLP layers. Models are fine-tuned for one epoch without any validation, and the model at the end of the epoch is used for evaluation. For GiVA, we only use Rand-B, as it is the most generic initialization out of the three strategies. Due to the costs associated with evaluation, we use the GPT-4 reference answers from MT-Bench and compute the loss on these answers to select the best learning rate for each method. Further details on the training setup are provided in Section B.4 of the supplementary material.

We test the fine-tuned models on MT-Bench (Zheng et al., 2023). The best model for each method is evaluated using GPT-4.[4]

Table 4 presents the performance of different adaptation methods. These results demonstrate that GiVA achieves performance comparable to both LoRA and VeRA despite using a rank of 32 and even outperforms both on the first turn questions.

4.3 Image Classification

Our final evaluation focuses on image classification. We adapt the vision backbones of DinoV2 ViT-B/14 (Oquab et al., 2023) and CLIP ViT-L/14 (Radford et al., 2021) across four image classification datasets: (1) CIFAR100 (Krizhevsky et al., 2009), (2) Food101 (Bossard et al., 2014), (3) Flowers102 (Nilsback and Zisserman, 2008), and (4) RESISC45 (Cheng et al., 2017). Adapter modules are added to the attention layers' query, key, and value projections, as well as the linear projections in the MLP layers. We largely follow the training setup of Albert et al. (2025) and additionally perform a learning rate sweep. Further details on the training setup are provided in Section B.5 of the supplementary material.

Table 5 reports the results across different adaptation methods. Consistent with earlier experiments, GiVA maintains competitive performance with other adaptation methods despite using a rank of 32. Across all datasets and models, the three initialization strategies for GiVA achieve broadly similar performance.

[4] `gpt-4-0613` as of Nov 25, 2025.

Table 4: Average scores on MT-Bench given by GPT-4 to the responses from Mistral (7B) fine-tuned on the Alpaca dataset. For GiVA, Rand-B is used.

Method	Params (M)	Score Turn 1	Turn 2	Avg.
Pre-trained	-	3.825	2.247	3.051
LoRA ($r = 64$)	167.8	6.906	5.700	6.303
VeRA ($r = 1024$)	1.6	6.956	5.375	6.166
GiVA ($r = 32$)	1.4	6.963	5.263	6.113

4.4 Training Time Comparison

In this section, we quantify the training time of different adaptation methods. The time excludes the duration of validation loops. We report the average times for fine-tuning Qwen 2 (0.5B) on Commonsense-15K, OLMo 2 (7B) on MetaMathQA, and DINOv2-ViT-B/14 (87M) and CLIP-ViT-L/14 (303M) on the four image classification datasets.

Table 6 presents the training times for various adaptation methods across models of different sizes. The results show that while GiVA is only slightly better than the other methods on DINOv2-ViT-B/14, it gradually closes the gap with LoRA as the model size increases, proving to be significantly faster than the other vector-based adaptation methods.

5 DISCUSSION

In our experiments, we estimate the first-step full fine-tuning gradient using a single batch of examples. The results indicate that this approach works well across a range of tasks, and using different sets of examples for the first-step full fine-tuning gradient computation, controlled by varying the seed, has little effect on performance. Furthermore, the three initialization strategies for GiVA perform comparably across tasks and models, providing further support for the solution derived in (5).

We also investigate whether incorporating more batches into the gradient estimation and taking an average improves the model performance. We repeat the commonsense reasoning experiments for GiVA with SecSV-B and estimate the gradient by averaging over multiple batches. The training setup is the same as in Section 4.2. Figure 2 shows the average performance

Table 5: Performance of various adaptation methods on image classification datasets. We report accuracy for all tasks. Experiments are repeated three times, and their averages are reported.

Method	**Params** (M)	**CIFAR100**	**Food101**	**Flowers102**	**RESISC45**
			(Acc.)		
			DINOv2-ViT-B/14		
Full Fine-Tuning	86.73	93.6±0.2	93.9±0.0	99.1±0.2	97.2±0.2
LoRA ($r = 32$)	4.72	93.7±0.1	93.8±0.0	99.6±0.0	96.6±0.2
VeRA ($r = 256$)	0.09	93.0±0.3	93.2±0.1	99.3±0.1	96.0±0.1
RandLoRA ($r = 10$)	3.59	93.7±0.1	94.1±0.0	99.6±0.0	96.9±0.2
GIVA ($r = 32$, TopSV-B)	0.08	93.1±0.3	93.1±0.1	99.2±0.1	96.2±0.0
GIVA ($r = 32$, SecSV-B)	0.08	93.0±0.1	93.1±0.1	99.3±0.1	95.9±0.1
GIVA ($r = 32$, Rand-B)	0.08	93.0±0.1	93.2±0.1	99.2±0.2	95.8±0.1
			CLIP-ViT-L/14		
Full Fine-Tuning	303.28	92.8±0.2	95.0±0.1	98.3±0.2	97.3±0.1
LoRA ($r = 32$)	12.58	93.1±0.1	95.2±0.2	98.3±0.1	97.2±0.1
VeRA ($r = 256$)	0.23	92.4±0.1	95.4±0.1	98.7±0.0	96.9±0.1
RandLoRA ($r = 10$)	12.78	93.0±0.1	95.4±0.2	98.2±0.3	97.4±0.1
GIVA ($r = 32$, TopSV-B)	0.20	92.4±0.2	95.4±0.0	98.4±0.2	96.9±0.3
GIVA ($r = 32$, SecSV-B)	0.20	92.3±0.2	95.4±0.1	98.5±0.2	97.0±0.1
GIVA ($r = 32$, Rand-B)	0.20	92.4±0.2	95.3±0.0	98.7±0.2	97.0±0.1

Table 6: Time taken for fine-tuning Qwen 2 on Commonsense-15K, OLMo 2 on MetaMathQA, and DINOv2-ViT-B/14 and CLIP-ViT-L/14 on the four image classification datasets. For each model, the training configuration is consistent across methods.

Method	**DINOv2** (87M)	**CLIP** (303M)	**Qwen 2** (494M)	**OLMo 2** (7B)
LoRA	1h21m	4h59m	11m	2h54m
VeRA	1h55m	7h51m	27m	3h26m
OSoRA	-	-	23m	3h25m
RandLoRA	2h05m	7h55m	23m	-
GIVA	1h49m	6h53m	12m	2h57m

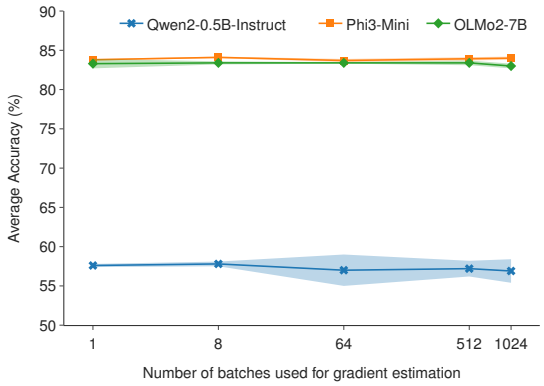

Figure 2: Average commonsense reasoning performance for GIVA (SecSV-B) when multiple batches of examples are used to estimate the first-step full fine-tuning gradient.

for three models with different numbers of examples used for gradient estimation. These results suggest that incorporating more batches in the gradient computation does not improve model performance.

6 LIMITATIONS

In this section, we note the limitations and potential extensions of our work. Although our method performs well on the datasets evaluated in this paper, its reliance on fixed bases computed from a single batch of examples may not fully capture the diversity of datasets composed of mixtures of multiple supervised fine-tuning sources, for example, Tulu 3 SFT mixture (Lambert et al., 2024), and may require further adaptation. In such cases, more sophisticated initialization strategies, such as incorporating gradients from multiple tasks within the mixture or combining random and gradient-based bases, could prove beneficial. Strategies for selecting the most representative examples for a dataset could improve performance across benchmarks. We leave these avenues for future work.

7 CONCLUSION

In this work, we introduced GIVA, a parameter-efficient adaptation method for large language models. Unlike previous vector-based approaches, GIVA constructs its bases from the first-step full fine-tuning gradient. Because these bases are derived from the fine-tuning data, GIVA reduces the rank requirements needed to match the performance of existing vector-based

adaptation methods. This reduction in required ranks leads to faster training times comparable to LoRA-like methods while preserving the extreme parameter efficiency of vector-based adaptation. We evaluated the proposed method across tasks in natural language understanding, natural language generation, and image classification, as well as on models of varying sizes. The results demonstrate that GIVA outperforms or matches existing parameter-efficient methods while using significantly fewer trainable parameters and reducing rank requirements by a factor of eight compared to prior vector-based adaptation methods.

Acknowledgements

This work was supported in part by the Illinois Campus Cluster Program and the Delta Advanced Computing and Data Resource. Delta is supported by the National Science Foundation (award OAC 2005572) and the State of Illinois and is a joint effort of the University of Illinois Urbana-Champaign and its National Center for Supercomputing Applications. We also thank the anonymous reviewers for their feedback.

References

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H. H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H. S., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chopra, P., Giorno, A. D., de Rosa, G., Dixon, M., Eldan, R., Iter, D., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Kim, Y. J., Khademi, M., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Liang, C., Liu, W., Lin, E., Lin, Z., Madan, P., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Ren, L., Rosset, C., Roy, S., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Song, X., Ruwase, O., Vaddamanu, P., Wang, X., Ward, R., Wang, G., Witte, P. A., Wyatt, M., Xu, C., Xu, J., Yadav, S., Yang, F., Yang, Z., Yu, D., Zhang, C.-Y., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhou, X., and Yang, Y. (2024). Phi-3 technical report: A highly capable language model locally on your phone. *ArXiv*, abs/2404.14219.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aghajanyan, A., Gupta, S., and Zettlemoyer, L. (2021). Intrinsic dimensionality explains the effectiveness of language model fine-tuning. 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)*, pages 7319–7328.

Albert, P., Zhang, F. Z., Saratchandran, H., Rodriguez-Opazo, C., van den Hengel, A., and Abbasnejad, E. (2025). Randlora: Full rank parameter-efficient fine-tuning of large models. In *The Thirteenth International Conference on Learning Representations*.

Bałazy, K., Banaei, M., Aberer, K., and Tabor, J. (2024). Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*.

Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. (2020). Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*.

Bossard, L., Guillaumin, M., and Van Gool, L. (2014). Food-101–mining discriminative components with random forests. In *European Conference on Computer Vision*, pages 446–461. Springer.

Büyükakyüz, K. (2024). Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating large language models trained on code.

Cheng, G., Han, J., and Lu, X. (2017). Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think

you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218.

Ed-dib, A., Datbayev, Z., and Aboussalah, A. M. (2024). Gelora: Geometric adaptive ranks for efficient lora fine-tuning. *arXiv preprint arXiv:2412.09250*.

Gemma Team, G., Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., et al. (2024). Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Han, J., Zhang, S., and Zhang, K. (2025). Osora: Output-dimension and singular-value initialized low-rank adaptation. *ArXiv*, abs/2505.14350.

Hao, Y., Cao, Y., and Mou, L. (2024). Flora: Low-Rank Adapters Are Secretly Gradient Compressors. arXiv:2402.03293 [cs].

Hayou, S., Ghosh, N., and Yu, B. (2024). Lora+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, pages 17783–17806. PMLR.

He, H., Ye, P., Ren, Y., Yuan, Y., Zhou, L., Ju, S., and Chen, L. (2025). Gora: Gradient-driven adaptive low rank adaptation. *arXiv preprint arXiv:2502.12171*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Hu, J. E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2022). Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hu, Z., Wang, L., Lan, Y., Xu, W., Lim, E.-P., Bing, L., Xu, X., Poria, S., and Lee, R. (2023). Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.

Koohpayegani, S. A., Navaneet, K., Nooralinejad, P., Kolouri, S., and Pirsiavash, H. (2024). Nola: Compressing lora using linear combination of random basis. In *International Conference on Learning Representations*.

Kopiczko, D. J., Blankevoort, T., and Asano, Y. M. (2024). Vera: Vector-based random matrix adaptation. In *International Conference on Learning Representations*.

Krizhevsky, A. et al. (2009). Learning multiple layers of features from tiny images.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., et al. (2024). Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Li, X. L. and Liang, P. (2021). 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)*, pages 4582–4597.

Lin, Z., Madotto, A., and Fung, P. (2020). Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 441–459.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. (2024a). Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Liu, S.-y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. (2024b). Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*.

Liu, W., Qiu, Z., Feng, Y., Xiu, Y., Xue, Y., Yu, L., Feng, H., Liu, Z., Heo, J., Peng, S., et al. (2024c). Parameter-efficient orthogonal finetuning via butterfly factorization. In *ICLR*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., and Bossan, B. (2022). PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Meng, F., Wang, Z., and Zhang, M. (2024). Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. (2018). Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *2008 Sixth Indian conference on computer vision, graphics & image processing*, pages 722–729. IEEE.

OLMo, T., Walsh, P., Soldaini, L., Groeneveld, D., Lo, K., Arora, S., Bhagia, A., Gu, Y., Huang, S., Jordan, M., et al. (2024). 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.-Y., Li, S.-W., Misra, I., Rabbat, M., Sharma, V., Synnaeve, G., Xu, H., Jegou, H., Mairal, J., Labatut, P., Joulin, A., and Bojanowski, P. (2023). Dinov2: Learning robust visual features without supervision.

Paischer, F., Hauzenberger, L., Schmied, T., Alkin, B., Deisenroth, M. P., and Hochreiter, S. (2024). Parameter efficient fine-tuning via explained variance adaptation. *arXiv preprint arXiv:2410.07170*.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2021). Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503.

Ponkshe, K., Singhal, R., Gorbunov, E., Tumanov, A., Horvath, S., and Vepakomma, P. (2024). Initialization using update approximation is a silver bullet for extremely efficient low-rank fine-tuning. *arXiv preprint arXiv:2411.19557*.

Qiu, Z., Liu, W., Feng, H., Xue, Y., Feng, Y., Liu, Z., Zhang, D., Weller, A., and Schölkopf, B. (2023). Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362.

Qwen Team, Q. (2024). Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Sakaguchi, K., Le Bras, R., Bhagavatula, C., and Choi, Y. (2020). Winogrande: An adversarial winograd schema challenge at scale. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 8732–8740.

Sap, M., Rashkin, H., Chen, D., Le Bras, R., and Choi, Y. (2019). Social IQa: Commonsense reasoning about social interactions. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.

Shin, T., Razeghi, Y., Logan IV, R. L., Wallace, E., and Singh, S. (2020). Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

Sung, Y.-L., Nair, V., and Raffel, C. A. (2021). Training neural networks with fixed sparse masks. *Advances in Neural Information Processing Systems*, 34:24193–24205.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Wang, S., Yu, L., and Li, J. (2024). Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931.

Wang, Z., Liang, J., He, R., Wang, Z., and Tan, T. (2025). Lora-pro: Are low-rank adapters properly optimized? In *The Thirteenth International Conference on Learning Representations*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2019). Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Wu, F., Hu, J., Min, G., and Wang, S. (2025). Memory-efficient orthogonal fine-tuning with principal subspace adaptation. *arXiv preprint arXiv:2505.11235*.

Wu, X., Huang, S., and Wei, F. (2024). Mixture of lora experts. In *International Conference on Learning Representations*.

Yang, Y., Li, X., Zhou, Z., Song, S., Wu, J., Nie, L., and Ghanem, B. (2024). Corda: Context-oriented decomposition adaptation of large language models for task-aware parameter-efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:71768–71791.

Yu, L., Jiang, W., Shi, H., YU, J., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., and Liu, W. (2024). Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Zhang, F. and Pilanci, M. (2024). Riemannian preconditioned lora for fine-tuning foundation models. In *International Conference on Machine Learning*, pages 59641–59669. PMLR.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. (2023). Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Zhang, Y., Liu, F., and Chen, Y. (2025). Lora-one: One-step full gradient could suffice for fine-tuning large language models, provably and efficiently. In *Forty-second International Conference on Machine Learning*.

Zhang, Y., Zhu, H., Liu, A., Yu, H., Koniusz, P., and King, I. (2024). Less is more: Extreme gradient boost rank-1 adaption for efficient finetuning of llms. *arXiv preprint arXiv:2410.19694*.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Zheng, T., Zhang, G., Shen, T., Liu, X., Lin, B. Y., Fu, J., Chen, W., and Yue, X. (2024). Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12834–12859.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes, Section 3]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes, Section 4.4]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes, Section A in the supplementary material]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [`https://github.com/neerajgangwar/giva`]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes, Section B in the supplementary material]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes, Section B in the supplementary material]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator if your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable, we have used publicly available models and datasets]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable, we have used publicly available models and datasets]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

   (a) The full text of instructions given to participants and screenshots. [Not Applicable]

   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

GIVA: Gradient-Informed Bases for Vector-Based Adaptation Supplementary Materials

A MISSING PROOFS

Theorem 1. *Consider the following optimization problem*

$$\arg\min_{A,B} \|\nabla_W \mathcal{L}(W_{pt}) A^T B^T BA - \nabla_W \mathcal{L}(W_{pt})\|_F$$

Here, $\nabla_W \mathcal{L}(W_{pt}) \in \mathbb{R}^{m\times d}$*,* $B \in \mathbb{R}^{m\times r}$*,* $A \in \mathbb{R}^{r\times d}$*, and* $r < \min(m, d)$*. The optimization problem reaches its minimum when*

$$A = V_r^T, B^T B = \mathbb{I}^{r\times r}$$

V_r *denotes the right singular vectors of* $\nabla_W \mathcal{L}(W_{pt})$*, corresponding to the* r *largest singular values.*

Proof. The rank of the first term in the optimization problem can be computed as

$$\operatorname{rank}\left(\nabla_W \mathcal{L}(W_{\text{pt}}) A^T B^T BA\right) \leq \min\left(\operatorname{rank}\left(\nabla_W \mathcal{L}(W_{\text{pt}})\right), \operatorname{rank}\left(A^T B^T BA\right)\right) \leq r$$

Using the Eckart-Young theorem, the optimization problem reaches its minimum when $\nabla_W \mathcal{L}(W_{\text{pt}}) A^T B^T BA$ is the best rank-r approximation of $\nabla_W \mathcal{L}(W_{\text{pt}})$ (Eckart and Young, 1936).

We prove the theorem by substituting the optimal A and B in the first term of the optimization problem. Let U_r and V_r denote the left and right singular vectors of $\nabla_W \mathcal{L}(W_{\text{pt}})$ corresponding to the r largest singular values $\{\sigma_1 \dots \sigma_r\}$. Define $\Sigma_r = \operatorname{diag}(\sigma_1 \dots \sigma_r)$, and let V_{d-r} denote the remaining right singular vectors.

$$\begin{aligned}
\nabla_W \mathcal{L}(W_{\text{pt}}) A^T B^T BA &= U\Sigma V^T V_r V_r^T \\
&= U\Sigma \begin{bmatrix} V_r^T \\ V_{d-r}^T \end{bmatrix} V_r V_r^T \\
&= U\Sigma \begin{bmatrix} \mathbb{I}^{r\times r} \\ \mathbf{0}^{d-r\times r} \end{bmatrix} V_r^T \\
&= U_r \Sigma_r V_r^T
\end{aligned}$$

which is the best rank-r approximation of $\nabla_W \mathcal{L}(W_{\text{pt}})$. □

B TRAINING DETAILS

Our experiments use the `transformers` (Wolf et al., 2019) and `peft` (Mangrulkar et al., 2022) libraries from HuggingFace. For GLUE, we use a single A10 GPU (24GB VRAM), and the remaining experiments are conducted on a single A100 GPU (40GB VRAM) or a single L40S GPU (48GB VRAM). Gradient accumulation is applied when a full batch does not fit into memory.

B.1 GLUE

Table 7 shows the common hyperparameters used for fine-tuning RoBERTa on various GLUE tasks. Additionally, we search for the best learning rate over $\left\{10^{-5}, 5\times10^{-5}, 10^{-4}\right\}$ for full fine-tuning, over $\left\{5\times10^{-5}, 10^{-4}, 2\times10^{-4}\right\}$ for LoRA and RandLoRA, and over $\left\{10^{-3}, 5\times10^{-3}, 10^{-2}\right\}$ for VeRA and GIVA. We do not use dropout or bias for any adapter modules. Finally, we use $\alpha = 2r$ for LoRA and RandLoRA and $d_{\text{initial}} = 0.1$ for VeRA.

Table 7: Common hyperparameters for fine-tuning RoBERTa on the GLUE tasks.

Optimizer	**Weight Decay**	**Warmup Steps**	**Epochs**
AdamW	10^{-2}	0.06	10
Gradient Clipping	**Input Length**	**Batch Size**	**LR Scheduler**
Norm-based at 1.0	128	64	Linear

Table 8: Common hyperparameters for fine-tuning Qwen 2, Phi 3, and OLMo 2 on commonsense reasoning.

Optimizer	**Weight Decay**	**Warmup Steps**	**Epochs**
AdamW	0	100	3
Gradient Clipping	**Max Input Tokens**	**Batch Size**	**LR Scheduler**
Norm-based at 1.0	512	16	Linear

Table 9: Common hyperparameters for fine-tuning OLMo 2 on MetaMathQA and Code-Feedback.

Optimizer	**Weight Decay**	**Warmup Steps**	**Epochs**
AdamW	0	0.03	1
Gradient Clipping	**Max Input Tokens**	**Batch Size**	**LR Scheduler**
Norm-based at 1.0	1024	32	Cosine

Table 10: Common hyperparameters for fine-tuning Mistral on Alpaca.

Optimizer	**Weight Decay**	**Warmup Steps**	**Epochs**
AdamW	10^{-4}	0.1	1
Gradient Clipping	**Max Input Tokens**	**Batch Size**	**LR Scheduler**
Norm-based at 1.0	1024	16	Cosine

Table 11: Common hyperparameters for fine-tuning the vision backbones of DINOv2-ViT-B/14 and CLIP-ViT-L/14 on CIFAR100, Food101, Flowers102, and RESISC45.

Optimizer	**Weight Decay**	**Warmup Steps**	**Gradient Clipping**	**Batch Size**	**LR Scheduler**
AdamW	10^{-1}	0.0	Norm-based at 1.0	128	Cosine

B.2 Commonsense Reasoning

Table 8 shows the common hyperparameters used for fine-tuning models on commonsense reasoning. Additionally, we search for the best learning rate over $\left\{10^{-5}, 5\times10^{-5}, 10^{-4}, 2\times10^{-4}\right\}$ for LoRA and RandLoRA, over $\left\{5\times10^{-4}, 10^{-3}, 5\times10^{-3}, 10^{-2}\right\}$ for OSoRA, and over $\left\{10^{-3}, 5\times10^{-3}, 10^{-2}\right\}$ for VeRA and GIVA. We use a dropout of 0.05 and no bias for adapter modules. Finally, we use $\alpha = 2r$ for LoRA, $\alpha = \frac{2r}{\sqrt{n}}$ for RandLoRA with n being the number of bases, and $d_{\text{initial}} = 1.0$ for VeRA.

B.3 Mathematical Reasoning and Code Generation

Table 9 shows the common hyperparameters used for fine-tuning OLMo 2 on MetaMathQA and Code-Feedback. Additionally, we search for the best learning rate over $\{5\times10^{-5}, 10^{-4}, 2\times10^{-4}, 5\times10^{-4}\}$ for LoRA and over $\{10^{-3}, 5\times10^{-3}, 10^{-2}, 2\times10^{-2}, 5\times10^{-2}\}$ for VeRA, OSoRA, and GIVA. We do not use dropout or bias for any adapter modules. Finally, we use $\alpha = 2r$ for LoRA and $d_{\text{initial}} = 1.0$ for VeRA.

B.4 Instruction Tuning

Table 10 shows the common hyperparameters used for fine-tuning. Additionally, we search for the best learning rate over $\{5\times10^{-6}, 10^{-5}, 5\times10^{-5}, 10^{-4}, 4\times10^{-4}\}$ for LoRA and over $\{10^{-4}, 5\times10^{-4}, 10^{-3}, 4\times10^{-3}, 5\times10^{-3}, 10^{-2}\}$ for VeRA and GIVA. We do not use dropout or bias for any adapter module. Finally, we use $\alpha = 2r$ for LoRA and $d_{\text{initial}} = 1.0$ for VeRA.

B.5 Image Classification

Table 11 shows the common hyperparameters used for image classification experiments. Additionally, we search for the best learning rate over $\left\{10^{-5}, 2\times10^{-5}, 5\times10^{-5}\right\}$ for full fine-tuning, over $\left\{5\times10^{-4}, 10^{-3}, 2\times10^{-3}\right\}$ for LoRA and RandLoRA, and over $\left\{5\times10^{-3}, 10^{-2}, 2\times10^{-2}\right\}$ for VeRA and GIVA. We use 10 epochs for CIFAR100, 15 epochs for Food101 and RESISC45, and 40 epochs for Flowers102. No dropout or bias is used for any adapter module. Finally, we use $\alpha = 1$ for both LoRA and RandLoRA and $d_{\text{initial}} = 1.0$ for VeRA.