GPART: END-TO-END ISOMETRIC FINE-TUNING VIA GLOBAL PARAMETER PARTITIONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adaptation (LoRA) has become a dominant paradigm for parameter-efficient fine-tuning (PEFT) of large-scale deep learning models. However, its bilinear parameterization induces a parameter-dependent geometry: the mapping from trainable parameters to weight updates is not generally distance-preserving. Related methods that project a low-dimensional vector into LoRA's parameter space, such as Uni-LoRA, improve parameter efficiency, but the subsequent bilinear map breaks end-to-end isometry. We propose GPart (Global Partition fine-tuning), a highly parameter-efficient fine-tuning method that maps a d-dimensional trainable vector directly into the full weight space through a sparse, isometric partition matrix. GPart retains a fixed global parameter-sharing prior while removing the additional low-rank reconstruction used by LoRA-based methods. This yields a simple parameterization with a single main hyperparameter (d), exact end-to-end isometry, and a minimal checkpoint representation consisting of the trainable vector and a random seed. GPart builds on the premise of effective fine-tuning within random low-dimensional subspaces of the full weight space *without requiring a low-rank matrix factorization*. Across natural language understanding, computer vision, and mathematical reasoning benchmarks, GPart matches or improves over existing PEFT methods at ultra-low parameter budgets. Beyond offering mathematical tractability and memory efficiency, the direct linear parameterization of GPart streamlines model selection and paves the way for compact adapter composition. Overall, GPart provides an elegant and competitive alternative for fine-tuning under small parameter budgets, with a fixed and predictable geometry between trainable coordinates and weight-space updates.

1 INTRODUCTION

Modern deep learning models for language, vision, and other modalities are often impractical to fine-tune conventionally due to a massive computational cost and time. Nonetheless, fine-tuning on downstream tasks remains an effective method that may outperform in-context learning Bucher & Martini (2024); Walsh et al. (2025). To overcome the challenge of updating the full model weights, parameter-efficient fine-tuning (PEFT) restricts updates to a low-dimensional trainable subspace. Among PEFT methods, LoRA (Hu et al., 2022) is the most widely used: for a weight matrix $W \in \mathbb{R}^{m\times n}$, it parameterizes the update as

$$\Delta W = BA, \qquad B \in \mathbb{R}^{m\times r},\ A \in \mathbb{R}^{r\times n}, \tag{1}$$

so that only $r(m+n)$ parameters are trained per layer.

LoRA is computationally efficient and effective, but its low-rank bilinear parameterization imposes additional, arbitrary structure on the trainable subspace. In particular, the map from trainable parameters (A, B) to the weight update BA is not an isometry, so Euclidean distances in the parameter space are not preserved in the weight space. As a result, the geometry seen by the optimizer in the trainable coordinates need not align with the geometry of the induced weight updates.

Recent work has pushed PEFT to even smaller trainable parameter budgets. VeRA (Kopiczko et al., 2024) freezes random matrices and trains only scaling vectors. Uni-LoRA (Li et al., 2025) further compresses the trainable space by optimizing a single d-dimensional vector $\boldsymbol{\theta}_d$, which is mapped into the LoRA parameter space $\mathbb{R}^D$ through a random partition matrix $P \in \mathbb{R}^{D\times d}$ satisfying $P^\top P = I_d$.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

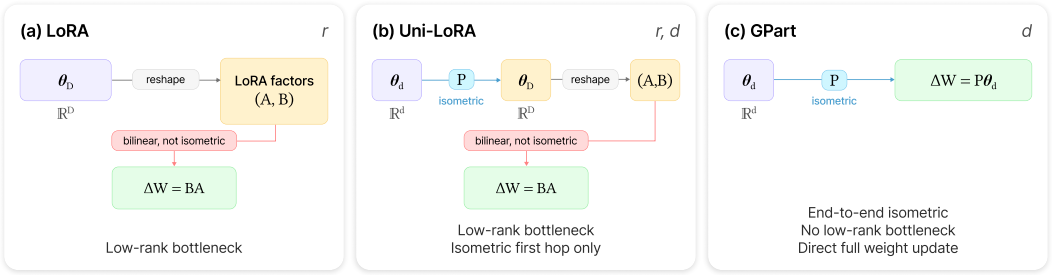

Figure 1: Comparison of PEFT parameterizations. LoRA and Uni-LoRA construct weight updates through the bilinear map $\Delta W = BA$, which breaks end-to-end distance preservation. GPart instead projects the trainable vector directly into full weight space with a seed-generated partition matrix P, yielding a one-step isometric parameterization with d as the only parameter-budget hyperparameter.

Despite this projection into LoRA parameter space is isometric, Uni-LoRA still requires the LoRA bilinear map, so the overall map from $\boldsymbol{\theta}_d$ to weight space is not isometric (Figure 1b).

We propose **GPart** (Global Partition fine-tuning), which removes the intermediate low-rank parameterization entirely. Instead of projecting into LoRA space, GPart maps the trainable vector directly into the full weight space:

$$W = W_0 + P\boldsymbol{\theta}_d, \qquad P \in \mathbb{R}^{N\times d}, \qquad P^\top P = I_d. \tag{2}$$

This yields a single linear map $\mathbb{R}^d \xrightarrow{P,\ \text{isometry}} \mathbb{R}^N$, so the trainable coordinates are connected to the weight update through an end-to-end isometric parameterization of the optimized subspace. In a comprehensive evaluation ranging from natural language understanding to computer vision, GPart demonstrates superior or comparable performance at ultra-low parameter budgets.

Furthermore, mathematical tractability of GPart simplifies utilization of adapters downstream. The direct linear parameterization supports compact adapter composition. For adapters sharing the same projection P, additive merging in the trainable coordinates exactly reproduces additive merging of their weight updates: $P(\boldsymbol{\theta}_d^{(1)} + \boldsymbol{\theta}_d^{(2)}) = P\boldsymbol{\theta}_d^{(1)} + P\boldsymbol{\theta}_d^{(2)}$, as analyzed in Appendix H. This parameterization also simplifies model selection. In LoRA, the rank r controls both the size and the structure of the low-rank update. In Uni-LoRA, the trainable budget is controlled by d, but r must still be chosen because the method retains the LoRA factorization. GPart removes this extra choice: d, the number of partition groups, is the only hyperparameter controlling subspace size (Figure 1).

Conceptually, GPart reconnects PEFT to intrinsic-dimension results of Aghajanyan et al. (2021), which showed that strong fine-tuning performance can emerge from optimization in random low-dimensional subspaces of the full parameter space. Unlike their structured Fastfood projection, GPart uses a sparse, seed-reconstructible partition that is an exact isometry for every realized projection. Importantly, Fastfood requires $\sim 60\%$ more training time than GPart at the training protocol. GPart therefore follows the intrinsic-dimension perspective directly while retaining the compact representation pioneered by VeRA (Kopiczko et al., 2024) and Uni-LoRA (Li et al., 2025).

Our contributions are as follows:

- We introduce GPart, a PEFT method that maps a d-dimensional trainable vector directly into the full weight space via a random partition matrix, eliminating the intermediate low-rank factorization used in LoRA-based approaches.
- We show theoretically that GPart preserves Euclidean geometry within the trainable subspace, whereas the LoRA bilinear map does not. This direct linear parameterization results in straightforward hyper-parameter selection and compact adapter composition.
- We empirically show that GPart is a competitive alternative to Uni-LoRA and other PEFT methods across encoder, decoder, and vision settings. At matched parameter budgets, GPart matches or improves over Uni-LoRA on average, with the clearest gains on natural language understanding and computer vision tasks.

2 Related Work

Low-rank adaptation. LoRA (Hu et al., 2022) parameterizes weight updates as a low-rank product, $\Delta W = BA$, and has become a de facto standard to parameter-efficient fine-tuning (PEFT). A number of follow-up methods modify this parameterization. For example, DoRA (Liu et al., 2024) separates magnitude and direction, while other variants such as AdaLoRA (Zhang et al., 2023) adapt the rank allocation across layers. Despite these differences, these methods retain the low-rank bilinear structure that maps trainable parameters to weight updates through BA.

Hyperefficiency in PEFT. Following the introduction of LoRA, related PEFT methods are introduced to further reduce the number of trainable parameters. BitFit (Ben Zaken et al., 2022) updates only bias parameters, showing that strong adaptation can be achieved with extremely small trainable subsets. VeRA (Kopiczko et al., 2024) freezes random matrices and learns only scaling vectors, enabling compact storage through the learned parameters and a random seed. FourierFT (Gao et al., 2024) reparameterizes weight updates in the frequency domain, learning a small number of Fourier coefficients per layer to reconstruct the full update via the inverse discrete Fourier transform. Uni-LoRA (Li et al., 2025) trains a single low-dimensional vector and projects it into the LoRA parameter space using a random partition matrix. GPart is closest in spirit to VeRA and Uni-LoRA in its use of seed-generated random projections, but differs in that it maps directly into the full weight space rather than into an intermediate LoRA parameterization.

Intrinsic dimensionality and random subspaces. Our work is also closely related to intrinsic-dimension approaches to fine-tuning. Aghajanyan et al. (2021) showed that effective fine-tuning can often be achieved by optimizing within a random low-dimensional subspace of the full parameter space. Their method projects a d-dimensional trainable vector into $\mathbb{R}^N$ using the Fastfood transform (Le et al., 2013), thereby directly parameterizing updates in the ambient weight space. GPart follows the same high-level random-subspace perspective, but uses a sparse, seed-reconstructible partition projection that is an exact isometry for each realized map, rather than a structured dense transform.

Hash-based parameter sharing. GPart is also related to earlier work on parameter sharing through hashing. HashedNet (Chen et al., 2015) compresses neural networks by assigning weights to shared hash buckets, so that multiple weights reuse the same learned parameter. This mechanism is algebraically similar to the partition matrix used in GPart, where each parameter index is assigned to one of d groups. The setting, however, is different: HashedNet was introduced for model compression and training compact models, whereas GPart uses the same type of sharing structure to parameterize fine-tuning updates around a pretrained model.

3 Method

We begin by describing GPart in the vectorized weight space of the pretrained model and comparing it with the LoRA-based factor space; we then formalize the construction of the partition matrix and the resulting forward and backward passes. Figure 2 provides an overview of the full pipeline.

3.1 Vectorized weight space

Consider a pretrained model with L adapted weight matrices $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$, possibly of different shapes. We flatten each matrix and concatenate the results into a single parameter vector

$$w_0 = \mathrm{Concat}\Big(\mathrm{vec}\big(W^{(1)}\big),\ \mathrm{vec}\big(W^{(2)}\big),\ \ldots,\ \mathrm{vec}\big(W^{(L)}\big)\Big) \in \mathbb{R}^N, \tag{3}$$

where $N = \sum_{\ell=1}^{L} m_\ell n_\ell$ is the total number of adapted parameters.

For comparison, Uni-LoRA parameterizes updates in the LoRA factor space rather than in $\mathbb{R}^N$. For each adapted layer ℓ, LoRA introduces factors $B^{(\ell)} \in \mathbb{R}^{m_\ell \times r}$ and $A^{(\ell)} \in \mathbb{R}^{r \times n_\ell}$. Flattening and concatenating these variables across layers gives

$$\boldsymbol{\theta}_D = \mathrm{Concat}\Big(\mathrm{vec}\big(B^{(1)}\big),\ \mathrm{vec}\big(A^{(1)}\big),\ \ldots,\ \mathrm{vec}\big(B^{(L)}\big),\ \mathrm{vec}\big(A^{(L)}\big)\Big) \in \mathbb{R}^D, \tag{4}$$

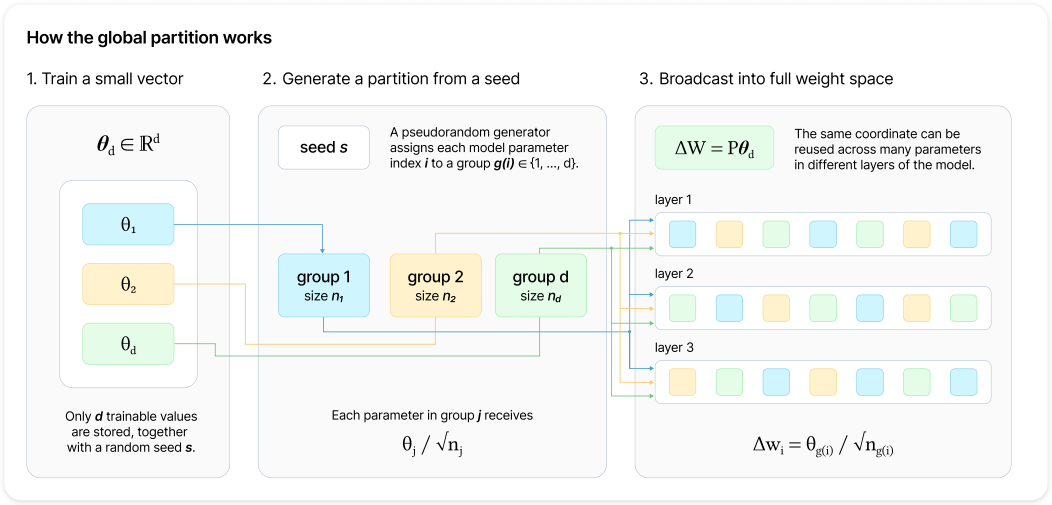

Figure 2: **Overview of GPart.** A d-dimensional trainable vector $\boldsymbol{\theta}_d$ is broadcast into the full weight space via a random partition generated from a seed s. Each model parameter w_i is assigned to a group $g(i) \in \{1, \ldots, d\}$ and updated as $\Delta w_i = \theta_{g(i)}/\sqrt{n_{g(i)}}$, preserving isometry. A minimal GPart checkpoint consists of only the d trainable values and the partition seed.

with $D = \sum_{\ell=1}^{L} r(m_\ell + n_\ell)$ typically satisfying $D \ll N$.

Unlike Uni-LoRA, GPart operates directly in the model weight space $\mathbb{R}^N$. It therefore requires neither low-rank factors nor a bilinear map from factor space back to model space.

3.2 Partition Matrix

Let $d \ll N$ denote the dimension of the trainable subspace, where N is the total number of adapted parameters. GPart defines a sparse matrix $P \in \mathbb{R}^{N \times d}$ by first flattening these adapted parameters into a global vector of length N, applying a seed-dependent pseudorandom permutation, and then splitting the permuted sequence into d disjoint groups. This yields a global assignment map $g : \{1, \ldots, N\} \to \{1, \ldots, d\}$,such that each parameter belongs to exactly one group and every group is nonempty.

For each group j, define $n_j = \big|\{i : g(i) = j\}\big|$. The matrix P is then given entrywise by

$$P_{ij} = \begin{cases} \frac{1}{\sqrt{n_j}}, & \text{if } g(i) = j, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

By construction, each row of P contains exactly one nonzero entry, while distinct columns have disjoint supports. Since every group is nonempty, each column has unit norm, and therefore

$$P^\top P = I_d. \tag{6}$$

Hence, P is an isometric embedding from $\mathbb{R}^d$ into $\mathbb{R}^N$.

The assignment is global: parameters are grouped across the entire model rather than separately within each layer. In practice, GPart is implemented through the seed-defined assignment $g(\cdot)$ and the group sizes $\{n_j\}_{j=1}^d$, without explicitly materializing P. The assignments may either be materialized as integer indices or generated statelessly from the seed and parameter positions, avoiding an $O(N)$ assignment buffer. Figure 2 illustrates both the partition construction and the induced broadcast update.

3.3 FORWARD AND BACKWARD PASSES

Given the pretrained weight vector $w_0 \in \mathbb{R}^N$ and trainable parameters $\boldsymbol{\theta}_d \in \mathbb{R}^d$, GPart defines the adapted weights as

$$w = w_0 + \Delta w, \qquad \Delta w = P\boldsymbol{\theta}_d. \tag{7}$$

Equivalently, each parameter i receives the update

$$\Delta w_i = \frac{\theta_{g(i)}}{\sqrt{n_{g(i)}}}. \tag{8}$$

Thus, all parameters assigned to the same group share one trainable value, normalized by the group size. Computing $P\boldsymbol{\theta}_d$ requires only a single pass over the N parameters and no explicit matrix multiplication.

For a loss $\mathcal{L}(w)$, the gradient with respect to $\boldsymbol{\theta}_d$ is $\nabla_{\boldsymbol{\theta}_d}\mathcal{L} = P^\top \nabla_w \mathcal{L}$. The backward pass therefore reduces to accumulating normalized gradient sums within each group, which again requires $O(N)$ work.

We initialize the trainable vector $\boldsymbol{\theta}_d$ at zero. Since GPart is linear in $\boldsymbol{\theta}_d$, this gives $\Delta w = P\boldsymbol{\theta}_d = 0$ at initialization, so optimization starts exactly from the pretrained model w_0. Unlike bilinear parameterizations such as LoRA, no symmetry-breaking random initialization is required: from Equation 3.3, the gradient with respect to $\boldsymbol{\theta}_d$ is generally nonzero even when $\boldsymbol{\theta}_d = 0$.

3.4 STORAGE AND HYPERPARAMETER

A minimal GPart checkpoint is fully specified by the seed s, which regenerates the partition, and the trainable vector $\boldsymbol{\theta}_d \in \mathbb{R}^d$, requiring only $d + 1$ stored values. This matches the minimal checkpoint representation of Uni-LoRA while requiring fewer trainable adapter parameters than LoRA ($O(D)$) and full fine-tuning ($O(N)$). At runtime, a materialized implementation additionally requires group-assignment indices and normalization scales; these assignment indices can instead be generated statelessly from the seed and parameter positions, eliminating the $O(N)$ index buffer. We discuss the resulting runtime memory and computational trade-offs in Section 5.

GPart is controlled by a single hyperparameter, the subspace dimension d. Setting $d = 1$ collapses all parameters to one shared scalar, while $d = N$ recovers full fine-tuning; intermediate values interpolate between these extremes. Unlike LoRA-based methods, GPart does not require choosing r, and uses d alone to control the trade-off between parameter efficiency and expressiveness.

4 THEORETICAL ANALYSIS

We analyze the geometry induced by GPart and contrast it with that of LoRA-based parameterizations. Our central observation is that GPart defines a linear isometric embedding from the trainable space into the full weight space, whereas LoRA-based methods map trainable parameters to weight updates through a bilinear transformation whose local geometry depends on the current parameter values. As a result, GPart provides fixed orthonormal coordinates for optimization within its trainable subspace, whereas the geometry induced by LoRA-based parameterizations varies throughout training.

4.1 END-TO-END ISOMETRY OF GPART

We begin with the basic geometric property of GPart.

Proposition 1 (GPart isometry)**.** *Let $P \in \mathbb{R}^{N\times d}$ be the partition matrix defined in Section 3, satisfying $P^\top P = I_d$. Then for any $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^d$,*

$$\|P\boldsymbol{\theta} - P\boldsymbol{\theta}'\|_2 = \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2. \tag{9}$$

Proof. By linearity,

$$\|P\boldsymbol{\theta} - P\boldsymbol{\theta}'\|_2^2 = \|P(\boldsymbol{\theta} - \boldsymbol{\theta}')\|_2^2 = (\boldsymbol{\theta} - \boldsymbol{\theta}')^\top P^\top P(\boldsymbol{\theta} - \boldsymbol{\theta}') = (\boldsymbol{\theta} - \boldsymbol{\theta}')^\top(\boldsymbol{\theta} - \boldsymbol{\theta}') = \|\boldsymbol{\theta} - \boldsymbol{\theta}'\|_2^2.$$

Taking square roots gives the result. □

Proposition 1 shows that GPart preserves Euclidean geometry exactly within the trainable subspace. Equivalently, the map $\boldsymbol{\theta} \mapsto P\boldsymbol{\theta}$ preserves norms, distances, and inner products on $\mathbb{R}^d$. Thus, optimization in $\boldsymbol{\theta}$-space is exactly optimization over the subspace $\mathrm{image}(P) \subset \mathbb{R}^N$ expressed in orthonormal coordinates.

4.2 LoRA INDUCES A PARAMETER-DEPENDENT METRIC

We now contrast this with the LoRA parameterization. For a single layer, LoRA represents the weight update as

$$\phi(A, B) = BA, \qquad B \in \mathbb{R}^{m\times r}, \quad A \in \mathbb{R}^{r\times n}. \tag{10}$$

Unlike GPart, this map is bilinear rather than linear. Its local behavior is therefore governed by a Jacobian that depends on the current values of A and B.

Using $\mathrm{vec}(XYZ) = (Z^\top \otimes X)\,\mathrm{vec}(Y)$, we obtain $\mathrm{vec}(BA) = (I_n \otimes B)\,\mathrm{vec}(A)$ and $\mathrm{vec}(BA) = (A^\top \otimes I_m)\,\mathrm{vec}(B)$. Hence,

$$\frac{\partial\ \mathrm{vec}(BA)}{\partial\ \mathrm{vec}(A)} = I_n \otimes B, \qquad \frac{\partial\ \mathrm{vec}(BA)}{\partial\ \mathrm{vec}(B)} = A^\top \otimes I_m. \tag{11}$$

These Jacobian blocks depend explicitly on A and B. Consequently, the metric induced on the trainable coordinates by the map $(A, B) \mapsto BA$ varies with the current point in parameter space. In particular, there is no fixed orthonormal coordinate system in (A, B)-space whose Euclidean geometry is preserved by the LoRA map throughout training. Equal-norm perturbations in trainable coordinates can therefore produce different update magnitudes depending on the values of A and B.

4.3 CONNECTION TO INTRINSIC DIMENSIONALITY

The motivation for GPart is closely related to the intrinsic-dimensionality view of fine-tuning introduced by Aghajanyan et al. (2021), where strong adaptation can emerge from optimization in random low-dimensional subspaces of the full parameter space. GPart follows this viewpoint by optimizing directly in a random d-dimensional subspace of $\mathbb{R}^N$. Its partition matrix satisfies $P^\top P = I_d$ for every realization, giving exact norm and inner-product preservation. This contrasts with structured random projections such as Fastfood (Le et al., 2013), which are isotropic in expectation but are not generally exact isometries for each realized projection. The key distinction from Uni-LoRA is the location of the random subspace: GPart operates directly in full weight space, whereas Uni-LoRA projects into LoRA parameter space before applying the bilinear map to weight space. Thus, the former preserves the geometry of the restricted problem exactly, whereas the latter does not.

5 EXPERIMENTS

We evaluate GPart across encoder-only and decoder-only models on three benchmark families: natural language understanding, mathematical reasoning, and computer vision. We compare against both non-PEFT baselines—Linear Probing (LP), which updates only the task-specific head, and Full Fine-tuning (FF), which updates all model parameters—and standard PEFT baselines, including LoRA (Hu et al., 2022), BitFit (Ben Zaken et al., 2022), VeRA (Kopiczko et al., 2024), FourierFT (Gao et al., 2024), and Uni-LoRA (Li et al., 2025), where applicable. The reported trainable-parameter counts exclude the task-specific head and include only backbone parameters for FF and adapter parameters for PEFT methods. Detailed training hyperparameters for each setup are provided in Appendix E.

5.1 NATURAL LANGUAGE UNDERSTANDING

We evaluate GPart on GLUE (Wang et al., 2019) using RoBERTa-base and RoBERTa-large (Liu et al., 2019). In Table 1, we report results on CoLA, SST-2, MRPC, STS-B, QNLI, and RTE, using Matthews correlation for CoLA, Pearson correlation for STS-B, and accuracy for the remaining tasks, following standard practice (Wang et al., 2019; Hu et al., 2022; Li et al., 2025). For each task, we reserve a portion of the training set as a development set for checkpoint selection and report the median and standard deviation across three runs on the validation set. For GPart, these runs vary both the training seed and the projection seed, and therefore also sample independent random partitions.

Table 1: GLUE validation results for RoBERTa-base and RoBERTa-large. Bold indicates the best result among PEFT methods, underlined entries indicate the second-best result, and highlighted entries mark cases where GPart outperforms Uni-LoRA.

Model	Method	# Params	CoLA	SST-2	MRPC	STS-B	QNLI	RTE	Avg.
RoBERTa-Base	LP	0	$43.1_{\pm 2.6}$	$84.3_{\pm 0.5}$	$72.5_{\pm 0.5}$	$66.3_{\pm 0.5}$	$70.6_{\pm 0.1}$	$59.2_{\pm 1.2}$	66.0
	FF	124M	$60.6_{\pm 2.1}$	$94.3_{\pm 0.5}$	$87.8_{\pm 0.9}$	$90.5_{\pm 0.1}$	$92.2_{\pm 0.3}$	$77.3_{\pm 0.5}$	83.8
	LoRA (r=8)	294K	$60.5_{\pm 1.2}$	$94.0_{\pm 0.6}$	$87.8_{\pm 0.5}$	$\mathbf{90.8}_{\pm 0.2}$	$\mathbf{92.8}_{\pm 0.1}$	$75.4_{\pm 1.0}$	$\underline{83.5}$
	BitFit	102K	$58.5_{\pm 1.0}$	$93.6_{\pm 0.2}$	$\underline{88.0}_{\pm 1.2}$	$90.2_{\pm 0.04}$	$91.8_{\pm 0.1}$	$\underline{78.7}_{\pm 6.1}$	$\underline{83.5}$
	VeRA (r=1024)	43K	$\mathbf{60.8}_{\pm 0.7}$	$94.0_{\pm 0.2}$	$87.5_{\pm 0.2}$	$90.2_{\pm 0.1}$	$91.8_{\pm 0.1}$	$76.2_{\pm 2.9}$	83.4
	Uni-LoRA	23K	$58.1_{\pm 0.7}$	$\underline{94.2}_{\pm 0.5}$	$86.5_{\pm 0.8}$	$90.3_{\pm 0.1}$	$\underline{92.0}_{\pm 0.3}$	$76.9_{\pm 1.7}$	83.0
	GPart *(Ours)*	23K	$\underline{60.6}_{\pm 1.9}$	$\mathbf{94.3}_{\pm 0.5}$	$\mathbf{88.5}_{\pm 0.6}$	$\underline{90.4}_{\pm 0.1}$	$91.1_{\pm 0.1}$	$\mathbf{79.8}_{\pm 1.4}$	**84.1**
RoBERTa-Large	LP	0	$44.9_{\pm 1.1}$	$86.8_{\pm 0.6}$	$72.3_{\pm 1.1}$	$58.8_{\pm 0.3}$	$66.8_{\pm 0.7}$	$58.8_{\pm 0.6}$	64.7
	FF	355M	$66.3_{\pm 0.5}$	$95.8_{\pm 0.3}$	$89.5_{\pm 0.2}$	$92.0_{\pm 0.4}$	$94.6_{\pm 0.2}$	$83.4_{\pm 1.2}$	86.9
	LoRA (r=8)	786K	$\underline{65.3}_{\pm 1.6}$	$\underline{95.6}_{\pm 0.1}$	$\underline{88.0}_{\pm 0.6}$	$91.3_{\pm 0.2}$	$\mathbf{94.8}_{\pm 0.2}$	$\underline{85.4}_{\pm 0.2}$	**86.7**
	BitFit	271K	$\mathbf{65.4}_{\pm 0.6}$	$\mathbf{95.9}_{\pm 0.2}$	$87.8_{\pm 1.2}$	$89.9_{\pm 0.4}$	$94.0_{\pm 0.2}$	$82.3_{\pm 12.5}$	85.9
	VeRA (r=256)	61K	$59.1_{\pm 3.6}$	$\mathbf{95.9}_{\pm 0.2}$	$87.8_{\pm 0.5}$	$91.0_{\pm 0.2}$	$\underline{94.1}_{\pm 0.5}$	$\mathbf{87.2}_{\pm 0.9}$	85.8
	Uni-LoRA	23K	$65.0_{\pm 7.3}$	$95.4_{\pm 0.1}$	$\mathbf{88.7}_{\pm 0.5}$	$\underline{91.5}_{\pm 0.4}$	$92.9_{\pm 1.0}$	$81.8_{\pm 3.1}$	85.9
	GPart *(Ours)*	23K	$64.2_{\pm 0.1}$	$95.4_{\pm 0.2}$	$87.2_{\pm 0.8}$	$\mathbf{91.7}_{\pm 0.2}$	$94.0_{\pm 0.2}$	$85.2_{\pm 0.9}$	$\underline{86.3}$

Table 2: Mathematical reasoning performance after fine-tuning on MetaMathQA and evaluation on GSM8K and MATH. We report mean test accuracy over three seeds, with standard deviations shown as subscripts. Bold denotes the best result for each model and benchmark. In the Params column, the left value denotes the number of trainable parameters for Uni-LoRA and GPart, while the right value denotes the number of trainable parameters for rank-1 LoRA.

		GSM8K			MATH		
Model	Params	Uni-LoRA	GPart	LoRA	Uni-LoRA	GPart	LoRA
Qwen-2.5-0.5B	131K/135K	$45.49_{\pm 0.39}$	$\mathbf{46.25}_{\pm 0.32}$	$45.99_{\pm 0.31}$	$\mathbf{19.77}_{\pm 0.85}$	$19.76_{\pm 0.99}$	$19.03_{\pm 0.25}$
Llama-3.1-8B	524K/851K	$68.83_{\pm 0.56}$	$\mathbf{69.45}_{\pm 0.13}$	$68.99_{\pm 1.18}$	$\mathbf{22.72}_{\pm 0.30}$	$22.39_{\pm 0.18}$	$22.53_{\pm 0.31}$
Gemma-7B	524K/802K	$\mathbf{72.37}_{\pm 0.93}$	$71.68_{\pm 0.88}$	$70.18_{\pm 0.13}$	$\mathbf{25.90}_{\pm 0.36}$	$25.10_{\pm 1.33}$	$23.90_{\pm 0.35}$
Qwen-2.5-3B	524K/460K	$78.64_{\pm 0.38}$	$78.65_{\pm 0.48}$	$\mathbf{78.92}_{\pm 0.41}$	$38.90_{\pm 1.48}$	$\mathbf{39.55}_{\pm 1.31}$	$38.73_{\pm 0.18}$
Qwen-2.5-7B	524K/630K	$83.07_{\pm 1.00}$	$82.92_{\pm 1.06}$	$\mathbf{83.83}_{\pm 0.31}$	$45.39_{\pm 1.28}$	$\mathbf{47.87}_{\pm 1.80}$	$44.14_{\pm 0.19}$
Average	–	69.68	**69.79**	69.58	30.54	**30.93**	29.67

All models and adapters are fine-tuned by us under this train/dev/val evaluation protocol to ensure a consistent comparison across methods. Full training details are provided in Table 7.

We compare against LP, FF, BitFit, LoRA, VeRA, and Uni-LoRA. GPart and Uni-LoRA are matched exactly by subspace dimension d; LoRA and VeRA are reported at the closest achievable budget according to their original hyper-parameterizations. GPart performs strongly at very small parameter budgets. On RoBERTa-base, GPart achieves an average score of 84.1 with only 23K trainable parameters, outperforming the matched-budget Uni-LoRA baseline and the larger LoRA and VeRA configurations on average. On RoBERTa-large, GPart also improves over Uni-LoRA at the matched budget, although LoRA attains a slightly higher average while using substantially more trainable parameters. Overall, the GLUE results show that GPart remains competitive under an extremely small parameter budget, with its clearest gains on RoBERTa-base.

5.2 Mathematical reasoning

We evaluate GPart on mathematical reasoning using a diverse set of pretrained decoder-only, non-reasoning models spanning multiple scales and architectures: Qwen-2.5-0.5B, Qwen2.5-3B, and Qwen-2.5-7B (Yang et al., 2024), Gemma-7B (Gemma Team et al., 2024), and Llama-3.1-8B (Grattafiori et al., 2024). Following the MetaMath setup (Yu et al., 2024b), we fine-tune on MetaMathQA and evaluate on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), reporting exact-match accuracy on the final answer. Our primary comparison is against Uni-LoRA under exactly matched trainable-parameter budgets; we additionally compare with rank-1 LoRA at comparable parameter budgets.

Table 2 shows that GPart, Uni-LoRA, and LoRA perform similarly across the decoder-only models. Averaged over all five architectures, GPart obtains the highest accuracy on both GSM8K and MATH, with 69.79 and 30.93, compared with 69.68 and 30.54 for Uni-LoRA and 69.58 and 29.67 for LoRA,

Table 3: Comparison with baseline approaches across seven computer vision datasets using ViT-Base and ViT-Large backbones. Bold indicates the best result among PEFT methods, and highlighted entries mark cases where GPart outperforms Uni-LoRA.

Model	Method	# Params	OxfordPets	CIFAR10	DTD	EuroSAT	FGVC	RESISC45	CIFAR100	Avg.
ViT-Base	LP	0	$90.28_{\pm 0.43}$	$96.41_{\pm 0.02}$	$69.77_{\pm 0.67}$	$88.72_{\pm 0.13}$	$17.44_{\pm 0.43}$	$74.22_{\pm 0.10}$	$84.28_{\pm 0.11}$	74.45
	FF	85.8M	$93.14_{\pm 0.40}$	$98.92_{\pm 0.05}$	$77.68_{\pm 1.21}$	$99.05_{\pm 0.09}$	$54.84_{\pm 1.23}$	$96.13_{\pm 0.13}$	$92.38_{\pm 0.13}$	87.45
	FourierFT	72K	$93.21_{\pm 0.26}$	$98.58_{\pm 0.07}$	$75.09_{\pm 0.37}$	$98.29_{\pm 0.04}$	$27.51_{\pm 0.64}$	$91.97_{\pm 0.31}$	$91.20_{\pm 0.14}$	82.26
	FourierFT	239K	$93.05_{\pm 0.34}$	$98.69_{\pm 0.06}$	$77.30_{\pm 0.61}$	$98.78_{\pm 0.11}$	$32.44_{\pm 0.99}$	$94.26_{\pm 0.20}$	$91.45_{\pm 0.18}$	83.71
	LoRA (r=4)	184K	$93.48_{\pm 0.26}$	$98.76_{\pm 0.11}$	$75.81_{\pm 1.42}$	$98.67_{\pm 0.09}$	$37.95_{\pm 0.46}$	$93.26_{\pm 0.10}$	**92.19**$_{\pm 0.14}$	84.30
	Uni-LoRA	72K	**93.91**$_{\pm 0.25}$	$98.68_{\pm 0.07}$	$76.60_{\pm 1.92}$	**99.00**$_{\pm 0.13}$	$51.42_{\pm 0.56}$	$94.13_{\pm 0.27}$	$91.81_{\pm 0.13}$	86.51
	GPart *(Ours)*	72K	$93.85_{\pm 0.14}$	**98.77**$_{\pm 0.05}$	**77.52**$_{\pm 1.59}$	**99.00**$_{\pm 0.10}$	**56.84**$_{\pm 0.17}$	**94.29**$_{\pm 0.22}$	$92.11_{\pm 0.11}$	**87.48**
ViT-Large	LP	0	$91.11_{\pm 0.30}$	$97.78_{\pm 0.04}$	$73.33_{\pm 0.26}$	$92.64_{\pm 0.08}$	$24.62_{\pm 0.24}$	$82.02_{\pm 0.11}$	$84.28_{\pm 0.11}$	77.97
	FF	303.3M	$94.43_{\pm 0.56}$	$99.15_{\pm 0.04}$	$81.79_{\pm 1.01}$	$99.04_{\pm 0.08}$	$68.25_{\pm 1.63}$	$96.43_{\pm 0.07}$	$93.58_{\pm 0.19}$	90.38
	FourierFT	144K	$94.46_{\pm 0.28}$	$99.10_{\pm 0.04}$	$80.83_{\pm 0.43}$	$98.65_{\pm 0.09}$	$39.92_{\pm 0.68}$	$93.86_{\pm 0.14}$	$93.31_{\pm 0.09}$	85.73
	FourierFT	480K	**94.84**$_{\pm 0.05}$	$99.08_{\pm 0.05}$	**81.88**$_{\pm 0.50}$	$98.66_{\pm 0.03}$	$51.28_{\pm 0.66}$	$95.20_{\pm 0.07}$	**93.37**$_{\pm 0.11}$	87.76
	Uni-LoRA	144K	$94.69_{\pm 0.31}$	$98.90_{\pm 0.15}$	$78.89_{\pm 0.86}$	$98.74_{\pm 0.52}$	**61.72**$_{\pm 0.29}$	**95.34**$_{\pm 0.09}$	$92.74_{\pm 0.13}$	**88.72**
	GPart *(Ours)*	144K	$93.86_{\pm 0.36}$	**99.11**$_{\pm 0.03}$	$78.74_{\pm 1.08}$	**99.08**$_{\pm 0.02}$	$61.48_{\pm 0.41}$	$95.09_{\pm 0.19}$	$92.59_{\pm 0.15}$	88.56

Table 4: Design analyses on RoBERTa-base GLUE at $d = 23{,}040$. The Fastfood comparison uses the same intrinsic dimension and training protocol. For partition sensitivity, the training seed is fixed while the projection seed is varied across three runs.

Analysis	Variant	CoLA	SST-2	MRPC	STS-B	QNLI	RTE	Avg.
Projection	Fastfood	$58.0_{\pm 1.1}$	$93.6_{\pm 0.2}$	$87.2_{\pm 0.2}$	$90.4_{\pm 0.1}$	$89.4_{\pm 0.1}$	**80.5**$_{\pm 1.6}$	83.2
	GPart	**60.6**$_{\pm 1.9}$	**94.3**$_{\pm 0.5}$	**88.5**$_{\pm 0.6}$	**90.4**$_{\pm 0.1}$	**91.1**$_{\pm 0.1}$	$79.8_{\pm 1.4}$	**84.1**
Sharing	Per-layer	$57.7_{\pm 1.0}$	$93.6_{\pm 0.2}$	$87.2_{\pm 1.0}$	**90.7**$_{\pm 0.2}$	$90.8_{\pm 0.1}$	$77.6_{\pm 1.1}$	82.9
	Global	**60.6**$_{\pm 1.9}$	**94.3**$_{\pm 0.5}$	**88.5**$_{\pm 0.6}$	$90.4_{\pm 0.1}$	**91.1**$_{\pm 0.1}$	**79.8**$_{\pm 1.4}$	**84.1**
Partition sensitivity	Fixed training seed	$60.6_{\pm 0.9}$	$93.2_{\pm 0.2}$	$87.7_{\pm 0.7}$	$90.3_{\pm 0.2}$	$90.7_{\pm 0.4}$	$79.5_{\pm 3.0}$	83.7

respectively. The differences are small relative to the observed run-to-run variation, so we interpret these results as evidence of competitiveness rather than superiority. At the model level, each method achieves the best result in some settings, with no method consistently dominating across architectures and benchmarks.

5.3 Computer vision tasks

We evaluate GPart on seven computer vision benchmarks to test whether the method transfers beyond language tasks. We follow the protocol used in FourierFT (Gao et al., 2024) and later adopted in Uni-LoRA (Li et al., 2025), using ViT-Base and ViT-Large (Dosovitskiy et al., 2021) pretrained backbones. We exclude Stanford Cars, which was included in prior work, due to the dataset provenance concerns discussed in the Ethics Statement (Section 6). To align with prior work, we use matched adapter budgets of 72K trainable parameters for ViT-Base and 144K for ViT-Large. All controlled results are averaged over five random seeds. Additional implementation details can be found in Table 9.

Table 3 shows that GPart achieves the strongest average among the matched-budget PEFT methods on ViT-Base and remains competitive with Uni-LoRA on ViT-Large. On ViT-Base, GPart improves over Uni-LoRA by 0.97 points on average. On ViT-Large, GPart performs within 0.16 points of Uni-LoRA. Across both backbones, GPart remains substantially more parameter-efficient than full fine-tuning while approaching its average accuracy.

5.4 Additional experiments

We examine GPart's main design choices: subspace dimension and partition sensitivity, global versus per-layer sharing, comparison with Fastfood projection, and optimization geometry.

Effect of subspace dimension d. We fine-tune RoBERTa-Large on SST-2 while varying d and keeping all other hyperparameters fixed. Figure 3 shows that performance rises rapidly at small d, plateaus in the mid-range, and slightly declines at large d. Thus, d provides a direct trade-off between parameter efficiency and model capacity.

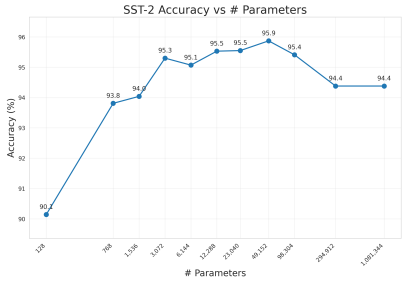

Figure 3: Accuracy on SST-2 with RoBERTa-Large as a function of the subspace dimension d.

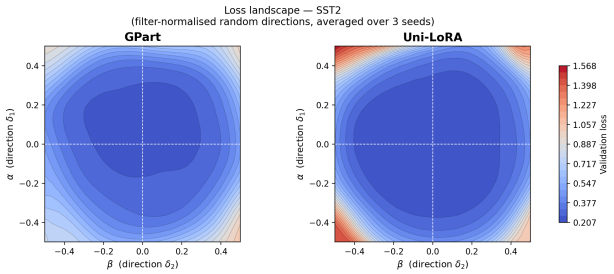

Figure 4: Loss landscape around the converged solution for GPart (left) and Uni-LoRA (right) on SST-2 with RoBERTa-Large and 23K trainable parameters, averaged over three random direction seeds (Li et al., 2018).

Sensitivity to the random partition. We fix the training seed and vary the projection seed across three runs on RoBERTa-base. As shown in Table 4, the standard deviation is at most 0.9 points on five of six GLUE tasks, with higher variability on RTE (3.0), suggesting that GPart is generally robust to the sampled partition.

Global versus per-layer sharing. At the same total dimension $d = 23{,}040$, global GPart achieves 84.1 average GLUE compared with 82.9 for a per-layer variant and performs better on five of six tasks (Table 4), supporting global sharing in this setting.

Comparison with Fastfood. At the same intrinsic dimension and training protocol, GPart achieves 84.1 average GLUE compared with 83.2 for Fastfood (Aghajanyan et al., 2021), while Fastfood requires approximately 60% more training time (Table 4). GPart is therefore both more accurate and more efficient.

Loss landscape geometry. We visualize the optimization geometry of GPart and Uni-LoRA using the filter-normalized random-direction method of Li et al. (2018), evaluating validation loss on a 30×30 grid around the converged solution with perturbations confined to the adapter subspace and averaged over three direction seeds. Figure 4 shows a smooth, well-centered basin for GPart, while Uni-LoRA exhibits sharper and more directionally asymmetric high-loss regions, consistent with the fixed geometry of GPart and the parameter-dependent geometry induced by the bilinear map.

6 Conclusion

We introduced GPart, a parameter-efficient fine-tuning method that maps a low-dimensional trainable vector directly into the full model weight space through a random partition matrix. By removing the intermediate low-rank parameterization used by LoRA-based methods, GPart yields an end-to-end isometric parameterization of the optimized subspace while retaining a single hyperparameter, d, to control the trainable budget. Across natural language understanding, mathematical reasoning, and computer vision benchmarks, GPart is competitive with existing PEFT methods at ultra-low parameter budgets, with its clearest gains on encoder-only and vision tasks. These results suggest that effective PEFT do not need to rely on a low-rank bottleneck, and that our direct random-subspace parameterization provides a simple and promising alternative.

Limitations. Our evaluation spans encoder-only language models, vision encoders, and decoder-only LLMs, but the decoder-only setting remains limited to a relatively narrow set of models and mathematical reasoning benchmarks. Future work should evaluate GPart on larger language and multimodal models, as well as broader instruction-following and long-context settings. The current stateless implementation reduces adapter-state and peak memory relative to LoRA and achieves comparable or faster runtime on smaller models. However, on the larger evaluated 7B/8B models, it is currently $2.2\times$–$3.5\times$ slower and does not yet support efficient batched multi-adapter serving. Detailed runtime and memory profiling is provided in Appendix F.

AI USE STATEMENT

In this work, we used generative AI tools for limited assistance with code development and manuscript preparation. Specifically, generative AI was used to assist with small code snippets, debugging, experimental scripting, and deterministic scripts for aggregating results across models and random seeds. It was also used for minor language and presentation improvements, including improving English, refining tables, and checking mathematical expressions.

Generative AI was not used to develop the scientific ideas, hypotheses, methodology, theoretical contributions, or conclusions of this work. The literature review, identification and analysis of related work, data selection and processing, and the substantive scientific content of the manuscript were carried out by the authors without generative AI assistance. The main research implementations, including the proposed adapter, mathematical functions, and evaluation procedures, were developed by the authors or, where AI assistance was used during implementation, were fully supervised and manually inspected by the authors. Figures, diagrams, and other scientific visualizations were created manually.

All AI-assisted text and code were reviewed by the authors. AI-assisted code was manually inspected and tested as part of the experimental pipeline, and scripts used for results collection are deterministic. Citations, literature analysis, and scientific ideas were produced and verified by the authors without generative AI involvement. We take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects or the collection of personal or sensitive data. All datasets used in our experiments are publicly available and are cited and described in Section 5 and Appendix I. The proposed method does not introduce capabilities intended to facilitate harmful use beyond those already present in the underlying pretrained models. We did not identify additional privacy, security, fairness, or discrimination concerns arising specifically from the proposed method.

We also considered the provenance and legal status of datasets when selecting experimental benchmarks. In particular, although Stanford Cars has been used in prior work on this topic, we refrain from using it because of ongoing legal concerns surrounding the collection and distribution of the dataset. Code required to reproduce our experiments is included with the submission and will also be released publicly. We are not aware of additional conflicts of interest or research-ethics concerns specific to this work.

REPRODUCIBILITY STATEMENT

We provide the source code required to reproduce our experiments as supplementary material with the submission. Our experiments follow standard evaluation protocols adopted in prior work, with training and evaluation procedures described in Section 5 and complete implementation details provided in Appendix E. Pseudocode for the proposed algorithm is provided in Appendix C. All datasets and pretrained models used in the experiments are publicly available.

For all reported tables, we report the standard deviation across at least three independent runs with different random seeds, capturing variability due to random initialization and data sampling. We document the computational setup, including the number and type of GPUs, training memory usage, training time, and other relevant implementation details in Section 5 and Appendix E; additional runtime and memory analyses are provided in Appendix F.

The assumptions and notation underlying our theoretical analysis are stated in Section 3. The main theoretical results are presented in Section 4, including the isometry of the GPart projection (Proposition 1), the non-isometry of LoRA's bilinear map (Proposition 11), and the connection to intrinsic dimensionality. Complete proofs and additional theoretical results, including the partition matrix construction, the gradient identity and norm bound, the exact weight-decay correspondence, and the non-isometric ablation theory, are provided in Appendices A and B.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *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)*, pp. 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568/.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1/.

Martin Juan José Bucher and Marco Martini. Fine-tuned'small'llms (still) significantly outperform zero-shot generative ai models in text classification. *arXiv preprint arXiv:2406.08660*, 2024.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, 2015. arXiv:1504.04788.

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

MohammadReza Davari and Eugene Belilovsky. Model breadcrumbs: Scaling multi-task model merging with sparse masks. In *European Conference on Computer Vision*, pp. 270–287. Springer, 2024.

Pala Tej Deep, Rishabh Bhardwaj, and Soujanya Poria. Della-merging: Reducing interference in model merging through magnitude-based sampling, 2024. URL https://arxiv.org/abs/2406.11617.

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

William Fleshman and Benjamin Van Durme. Spectr: Dynamically composing LM experts with spectral routing. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=tK8GHR62EX.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Thomas Mesnard Gemma Team, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, and et al. Gemma. 2024. doi: 10.34740/KAGGLE/M/3301. URL https://www.kaggle.com/m/3301.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ziyi Han, Huanyu Wang, Zeyu Zhang, Xiangxiang Dai, Xutong Liu, and John C.S. Lui. HiLoRA: Adaptive hierarchical LoRA routing for training-free domain generalization. *arXiv preprint arXiv:2510.12266*, 2025.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17783–17806. PMLR, 2024.

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

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with LoRA. *arXiv preprint arXiv:2312.03732*, 2023.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Quoc Le, Tamás Sarlós, and Alex Smola. Fastfood — approximating kernel expansions in loglinear time. In *International Conference on Machine Learning (ICML)*, 2013.

Yu-Ang Lee, Ching-Yun Ko, Pin-Yu Chen, and Mi-Yen Yeh. Learning rate matters: Vanilla lora may suffice for llm fine-tuning, 2026. URL https://arxiv.org/abs/2602.04998.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Neural Information Processing Systems*, 2018.

Kaiyang Li, Shaobo Han, Qing Su, Wei Li, Zhipeng Cai, and Shihao Ji. Uni-LoRA: One vector is all you need. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=hzBqQZK2iV.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. LoftQ: LoRA-fine-tuning-aware quantization for large language models. *arXiv preprint arXiv:2310.08659*, 2023.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach, 2019. URL https://arxiv.org/abs/1907.11692.

Daniel Marczak, Bartłomiej Twardowski, Tomasz Trzciński, and Sebastian Cygert. Magmax: Leveraging model merging for seamless continual learning. 2024.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2404.02948.

Oleksiy Ostapenko, Zhan Su, Edoardo Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards modular LLMs by building and reusing a library of loRAs. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=0ZFWfeVsaD.

Joel Walsh, Siddarth Mamidanna, Benjamin Nye, Mark Core, and Daniel Auerbach. Fine-tuning for better few shot prompting: An empirical comparison for short answer grading. *arXiv preprint arXiv:2508.04063*, 2025.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJ4km2R5t7.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 7093–7115. Curran Associates, Inc., 2023. doi: 10.52202/075280-0310. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1644c9af28ab7916874f6fd6228a9bcf-Paper-Conference.pdf.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Linyan Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Icml*, volume 2, pp. 21, 2024a.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=N8N0hgNDRt.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Ziyu Zhao, Tao Shen, Didi Zhu, Zexi Li, Jing Su, Xuwu Wang, Kun Kuang, and Fei Wu. Merging LoRAs like playing LEGO: Pushing the modularity of LoRA to extremes through rank-wise clustering. In *International Conference on Learning Representations*, 2025.

A Formal Analysis of GPart's Isometric Structure

A.1 Partition matrix construction

We provide a self-contained construction of the partition matrix and proof of its isometry property.

Definition 1 (Partition matrix)**.** *Given N parameters, d groups, and a random assignment function $g : \{1, \dots, N\} \to \{1, \dots, d\}$, the partition matrix $P \in \mathbb{R}^{N \times d}$ is defined by:*

$$P_{ij} = \begin{cases} 1/\sqrt{n_j} & \text{if } g(i) = j, \\ 0 & \text{otherwise}, \end{cases} \tag{12}$$

where $n_j = |\{i : g(i) = j\}|$.

Proposition 2. *$P^\top P = I_d$.*

Proof. $(P^\top P)_{jk} = \sum_{i=1}^{N} P_{ij} P_{ik}$. If $j \neq k$, the supports of columns j and k are disjoint (each row has a single non-zero entry), so the sum is 0. If $j = k$, the sum is $\sum_{i:g(i)=j} (1/\sqrt{n_j})^2 = n_j \cdot (1/n_j) = 1$. □

A.2 Gradient identity

Proposition 3. *For any loss function $\mathcal{L}(w)$ with $w = w_0 + P\boldsymbol{\theta}_d$:*

$$\nabla_{\boldsymbol{\theta}_d} \mathcal{L} = P^\top \nabla_w \mathcal{L}. \tag{13}$$

Moreover, $\|\nabla_{\boldsymbol{\theta}_d} \mathcal{L}\|_2 \leq \|\nabla_w \mathcal{L}\|_2$, with equality when $\nabla_w \mathcal{L}$ lies entirely in the image of P.

Proof. The first identity follows from the chain rule. For the norm bound: $\|\nabla_{\boldsymbol{\theta}_d} \mathcal{L}\|^2 = \|P^\top \nabla_w \mathcal{L}\|^2 = (\nabla_w \mathcal{L})^\top P P^\top (\nabla_w \mathcal{L})$. Since $PP^\top$ is an orthogonal projection onto $\mathrm{image}(P)$, its eigenvalues are 0 and 1, giving the inequality. □

A.3 Weight decay and regularization

End-to-end isometry also gives GPart a simple interpretation under L2 regularization. If weight decay is applied to $\boldsymbol{\theta}_d$, then

$$\lambda\|\boldsymbol{\theta}_d\|_2^2 = \lambda\|P\boldsymbol{\theta}_d\|_2^2 = \lambda\|\Delta w\|_2^2, \tag{14}$$

where $\Delta w = P\boldsymbol{\theta}_d = w - w_0$. Thus, weight decay on the trainable parameters is exactly weight decay on the induced perturbation in full weight space.

For LoRA, regularization is typically applied to the factors A and B, yielding the penalty

$$\lambda\big(\|A\|_F^2 + \|B\|_F^2\big).$$

This does not directly equal the squared norm of the induced weight perturbation $\Delta W = BA$. Instead, by submultiplicativity of the Frobenius norm and the arithmetic–geometric mean inequality,

$$\|\Delta W\|_F = \|BA\|_F \le \|B\|_F\|A\|_F \le \frac{1}{2}\big(\|A\|_F^2 + \|B\|_F^2\big). \tag{15}$$

Thus, regularization in LoRA controls only an upper bound on the norm of the resulting weight update, rather than the norm itself.

Remark 1. *For Uni-LoRA, applying weight decay to the trainable vector yields an exact norm interpretation in the intermediate LoRA parameter space, since the projection into that space is isometric. However, after the subsequent bilinear map into weight space, the same mismatch as in LoRA remains.*

A.4 Relation to structured random projections

GPart differs from structured random projections such as Fastfood (Le et al., 2013) in the strength of its geometric guarantee. For every realized partition P, GPart satisfies

$$P^\top P = I_d,$$

and therefore preserves norms and inner products exactly over the entire trainable space.

By contrast, an appropriately normalized Fastfood projection is isotropic in the second-moment sense,

$$\mathbb{E}[P^\top P] = I_d,$$

but a particular sampled projection does not generally satisfy $P^\top P = I_d$. Thus, GPart provides an exact per-realization isometry rather than norm preservation only in expectation over the random projection.

We compare the two parameterizations empirically at matched intrinsic dimension and training protocol in Table 4.

B The Cost of Breaking Isometry

We study the role of the $1/\sqrt{n_j}$ normalization in GPart by comparing the standard isometric construction with a non-isometric variant in which the normalization is removed. This comparison isolates whether end-to-end isometry is merely a geometric convenience or whether it has a measurable effect on optimization and downstream performance.

B.1 Setup

We compare two variants of GPart that differ only in the column normalization of P:

- **GPart (isometric):** $P_{ij} = 1/\sqrt{n_j}$ if $g(i) = j$, so $P^\top P = I_d$.
- **GPart (non-isometric):** $P_{ij} = 1$ if $g(i) = j$, so $P^\top P = \mathrm{diag}(n_1, \ldots, n_d)$.

All other aspects of the method are identical.

B.2 WEIGHT DECAY MISCALIBRATION

The performance gap between the two variants can be understood as a direct consequence of Equation 14. In the isometric case, we have

$$\|\Delta w\|_2^2 = \|P_{\text{iso}}\boldsymbol{\theta}_d\|_2^2 = \|\boldsymbol{\theta}_d\|_2^2,$$

so AdamW weight decay on $\boldsymbol{\theta}_d$ with coefficient λ corresponds exactly to L_2 regularization on the induced weight perturbation.

In the non-isometric case, the perturbation associated with group j has magnitude $\sqrt{n_j}\,\theta_j$, so

$$\|\Delta w\|_2^2 = \sum_j n_j \theta_j^2.$$

AdamW still applies decay directly to θ_j, so the decay acts at scale $\|\boldsymbol{\theta}_d\|_2^2$, while the actual weight perturbation lives at scale $\sum_j n_j\theta_j^2$. Since $n_j \approx N/d$ under a roughly balanced random partition, the regularization is miscalibrated by a factor of approximately N/d. For typical settings such as $N \approx 10^8$ and $d \approx 10^4$, this factor is on the order of 10^4.

This mismatch is not corrected by Adam's gradient normalization. Equivalently, the non-isometric parameterization behaves like the isometric variant with the induced weight perturbation scaled up by $\sqrt{n_j}$ while the corresponding regularization strength is scaled down by n_j. In practice, this yields severe under-regularization in weight space.

B.3 EMPIRICAL EFFECT OF NON-ISOMETRIC P

To test whether this theoretical mismatch has practical consequences, we compare GPart against the non-isometric variant on GLUE using RoBERTa-base and RoBERTa-large at a matched budget of 23K trainable parameters. For fairness, we independently tuned the optimization hyperparameters of the non-isometric variant so that any performance difference in performance cannot be attributed to a mismatched optimization scale. The results are reported in Table 5.

Table 5: Isometric vs. non-isometric GPart on GLUE.

Model	Method	# Params	CoLA	SST-2	MRPC	STS-B	QNLI	RTE	Avg
Base	GPart Non-Iso	23K	$44.7_{\pm 3.4}$	$93.5_{\pm 0.5}$	$87.5_{\pm 0.9}$	$90.3_{\pm 0.2}$	$\mathbf{91.2}_{\pm 0.1}$	$77.3_{\pm 2.8}$	80.7
	GPart	23K	$\mathbf{60.6}_{\pm 1.9}$	$\mathbf{94.3}_{\pm 0.5}$	$\mathbf{88.5}_{\pm 0.6}$	$\mathbf{90.4}_{\pm 0.1}$	$91.1_{\pm 0.1}$	$\mathbf{79.8}_{\pm 1.4}$	**84.1**
Large	GPart Non-Iso	23K	$60.1_{\pm 0.6}$	$95.4_{\pm 0.3}$	$\mathbf{89.0}_{\pm 1.6}$	$89.9_{\pm 2.3}$	$93.8_{\pm 0.1}$	$80.5_{\pm 2.2}$	84.8
	GPart	23K	$\mathbf{64.2}_{\pm 0.1}$	$\mathbf{95.4}_{\pm 0.2}$	$87.2_{\pm 0.8}$	$\mathbf{91.7}_{\pm 0.2}$	$\mathbf{94.0}_{\pm 0.2}$	$\mathbf{85.2}_{\pm 0.9}$	**86.3**

Across both RoBERTa-base and RoBERTa-large, the isometric variant achieves a higher average score than the non-isometric variant. The gap is especially pronounced on CoLA for the base model (60.6 vs. 44.7) and on RTE for the large model (85.2 vs. 80.5). Since the partition assignments, parameter-sharing pattern, and subspace dimension are held fixed, this result isolates the practical effect of the isometric normalization.

These results are consistent with the analysis above. Removing the $1/\sqrt{n_j}$ factor does not merely alter the parameterization algebraically; it changes the relationship between parameter-space regularization and the actual magnitude of the induced weight update. The empirical degradation in Table 5 therefore supports the view that the isometric normalization is a practically important part of GPart rather than a cosmetic design choice.

C ALGORITHM

Algorithm 1 summarizes the GPart training procedure. Given a fixed random seed, GPart constructs a deterministic partition of the pretrained parameters into d groups and optimizes a single scalar per group. The partition and group sizes implicitly define the projection, avoiding the need to materialize a dense projection matrix. During training, the resulting update is added to the frozen pretrained parameters, while optimization is performed only over the d-dimensional parameter vector $\boldsymbol{\theta}_d$. The learned parameters and partition seed are sufficient to reconstruct the adapted model.

Algorithm 1 GPart: Global Partition Fine-Tuning

Require: Pretrained parameters $w_0 \in \mathbb{R}^N$, training set $\mathcal{D}$, subspace dimension d, seed s, optimizer Opt

Generate a fixed partition map $g : \{1, \dots, N\} \to \{1, \dots, d\}$ from seed s
Compute group sizes $n_j = |\{i : g(i) = j\}|$ for $j = 1, \dots, d$
Initialize $\boldsymbol{\theta}_d = 0$
Implement P implicitly using only $g(\cdot)$ and $\{n_j\}_{j=1}^d$
for each minibatch $\mathcal{B} \subset \mathcal{D}$ **do**
    Form Δw implicitly using $(\Delta w)_i \leftarrow (\boldsymbol{\theta}_d)_{g(i)} / \sqrt{n_{g(i)}}$ for all i
    Set adapted parameters $w \leftarrow w_0 + \Delta w$
    Compute loss $\mathcal{L}_{\mathcal{B}}(w)$
    Compute $\nabla_{\boldsymbol{\theta}_d} \mathcal{L}_{\mathcal{B}}$ from $\nabla_w \mathcal{L}_{\mathcal{B}}$
    Update $\boldsymbol{\theta}_d \leftarrow \mathrm{Opt}(\boldsymbol{\theta}_d, \nabla_{\boldsymbol{\theta}_d} \mathcal{L}_{\mathcal{B}})$
end for
return $s, \boldsymbol{\theta}_d$

D Comparison of Parameter Counts

Table 6 summarizes the number of trainable parameters introduced by each method, both per adapted layer and globally. Full fine-tuning scales with the total model size N, while LoRA introduces $r(m + n)$ parameters per layer, giving a global count $D = \sum_\ell r(m_\ell + n_\ell)$ that grows with depth and layer width. BitFit and VeRA are more frugal, tuning only bias vectors or diagonal scaling factors, but their global counts still accumulate over layers. FourierFT decouples the budget from layer dimensions by fixing d frequencies per layer, yielding a global count of $L \times d$. GPart and Uni-LoRA go one step further: both parameterize the entire adaptation through a *single* global vector $\boldsymbol{\theta}_d \in \mathbb{R}^d$, shared across all L adapted layers, so the parameter count is independent of model depth and layer dimensions. The key distinction between the two methods is therefore not in parameter count — which is identical by construction in our experiments — but in how $\boldsymbol{\theta}_d$ is mapped back to weight space.

Table 6: Trainable parameter counts per adapted layer and globally across a model with L adapted layers of dimensions $W_\ell \in \mathbb{R}^{m_\ell \times n_\ell}$, with N denoting total model parameters.

Method	Per layer	Global
Full FT	mn	N
LoRA (rank r)	$r(m + n)$	$\sum_\ell r(m_\ell + n_\ell) = D$
BitFit	m (bias only)	$\sum_\ell m_\ell$
VeRA	$m + n$ (diag. scaling)	$\sum_\ell (m_\ell + n_\ell)$
FourierFT	d	$L \times d$
Uni-LoRA	d (global, shared across all layers)	
GPart	d (global, shared across all layers)	

E Implementation Details

All experiments are run on a single NVIDIA H100 80GB GPU. We use AdamW throughout, with task-specific learning rates for the classification head and the subspace parameters $\boldsymbol{\theta}_d$ tuned independently. The partition matrix P is fixed at initialization and applied to the `target_modules` matrices in every transformer block, excluding the task head. Full per-task hyperparameters for natural language understanding, mathematical reasoning, and computer vision are provided in Tables 7, 8, and 9, respectively.

GPart never explicitly materializes the partition matrix P. Instead, the broadcast update is applied implicitly during the forward and backward passes. Each adapted layer gathers the relevant entries of $\boldsymbol{\theta}_d$, rescales and reshapes them into weight updates, and uses a custom autograd function to accumulate gradients directly into the global trainable vector.

Our stateless implementation can additionally derive parameter-to-group assignments directly from the seed and global parameter positions, avoiding the $O(N)$ materialized assignment-index buffer. Detailed runtime and memory comparisons with LoRA, including the resulting memory savings and large-model computational overhead, are provided in Appendix F.

Table 7: Hyperparameters for GLUE experiments.

Model	Hyperparameter	SST-2	MRPC	CoLA	QNLI	RTE	STS-B
	Optimizer	AdamW					
	Weight Decay	0.1					
	Warmup Ratio	0.06					
	LR Schedule	Linear					
	Init. of $\boldsymbol{\theta}_d$	0					
	LR ($\boldsymbol{\theta}_d$)	5E-3					
	$\lvert\boldsymbol{\theta}_d\rvert$	23,040					
	Batch Size Per GPU	32					
	Target Modules	`["query", "value"]`					
BASE	Epochs	60	30	80	25	160	80
BASE	LR (Head)	5E-4	1E-3	2E-4	1E-3	1E-2	2E-4
BASE	Max Seq. Len.	512					
BASE	Training Time (Total)	154 minutes					
BASE	GPU Memory	19GiB					
LARGE	Epochs	30	30	60	25	120	80
LARGE	LR (Head)	2E-4	1E-3	2E-3	1E-3	1E-2	2E-4
LARGE	Max Seq. Len.	128					
LARGE	Training Time (Total)	66 minutes					
LARGE	GPU Memory	15GiB					

Table 8: Hyperparameters for mathematical reasoning experiments.

Hyperparameter	Qwen-2.5-0.5B	Qwen-2.5-3B	Qwen-2.5-7B	Gemma-7B	Llama-3.1-8B
Optimizer	AdamW				
LR Schedule	Cosine				
Batch Size	2				
Accumulation Steps	8				
Warmup Ratio	0.05				
Weight Decay	0.05				
Epochs	2				
Warmup Ratio	0.02				
Target Modules	`["q_proj", "k_proj", "v_proj", "o_proj"]`				
Init. of $\boldsymbol{\theta}_d$	0				
Max Seq. Len.	2048				
LR ($\boldsymbol{\theta}_d$)	2E-4				
$\lvert\boldsymbol{\theta}_d\rvert$	131,072	524,288	524,288	524,288	524,288
Training Time	4h	8h	15h	21h	17h
GPU Memory	19GiB	31GiB	45GiB	66GiB	53GiB

F RUNTIME AND MEMORY ANALYSIS

The minimal GPart checkpoint contains only the trainable vector $\boldsymbol{\theta}_d$ and the partition seed. At runtime, however, a materialized implementation additionally requires parameter-to-group indices and normalization scales. With `int32` indices, the assignment buffer alone requires $4N$ bytes. Our stateless implementation eliminates this buffer by deriving assignments directly from the seed and global parameter positions.

For Qwen-2.5-7B, stateless assignment reduces the fixed runtime buffers to 2 MiB and initialization time from 70.79 s to 8.57 s.

Table 9: Hyperparameters for computer vision experiments.

Model	Hyperparameter	OxfordPets	CIFAR10	DTD	EuroSAT	FGVC	RESISC45	CIFAR100
	Optimizer				AdamW			
	Weight Decay				0.01			
	LR Schedule				Linear			
	Warmup Ratio				0.05			
	Init. of $\boldsymbol{\theta}_d$				0			
	Epochs				20			
	Target Modules				`["query", "value"]`			
ViT-Base	LR (Head)	2E-2	5E-3	1E-2	1E-2	1E-2	5E-2	5E-3
	LR ($\boldsymbol{\theta}_d$)	1E-2	1E-2	1E-2	5E-2	5E-2	8E-2	5E-2
	$\lvert\boldsymbol{\theta}_d\rvert$				72,000			
	Batch Size Per GPU				64			
	Training Time	4m	26m	4m	11m	15m	15m	27m
	GPU Memory				5GiB			
ViT-Large	LR (Head)	2E-2	5E-3	1E-2	1E-2	1E-2	5E-2	5E-3
	LR ($\boldsymbol{\theta}_d$)	1E-2	1E-2	1E-2	5E-2	5E-2	8E-2	5E-2
	$\lvert\boldsymbol{\theta}_d\rvert$				144,000			
	Batch Size Per GPU				32			
	Training Time	6m	56m	8m	24m	17m	27m	51m
	GPU Memory				8GiB			

Table 10: Memory required by materialized assignment indices and the reduction obtained with stateless assignment. Peak memory reports the decrease in peak allocated CUDA memory relative to the materialized GPart implementation.

Metric	Qwen-2.5-0.5B	Qwen-2.5-3B	Qwen-2.5-7B	Gemma-7B	Llama-3.1-8B
Materialized index buffer	0.17 GiB	1.27 GiB	3.06 GiB	5.25 GiB	5.01 GiB
Stateless index buffer	0 GiB	0 GiB	0 GiB	0 GiB	0 GiB
Peak memory reduction	0.29 GiB	2.12 GiB	5.32 GiB	9.02 GiB	8.59 GiB

We further compare stateless GPart with rank-4 LoRA under identical models, data, precision, batching, and implementation conditions.

Table 11: Runtime and memory comparison between stateless GPart and rank-4 LoRA. Speed is GPart throughput relative to LoRA; peak Δ is the reduction in peak allocated CUDA memory for GPart. Adapter state excludes optimizer state.

Model	GPart speed vs. LoRA	Peak Δ (LoRA$-$GPart)	Adapter state (LoRA/GPart)
Qwen-2.5-0.5B	1.10×	+0.73 GiB	2.06 / 1.00 MiB
Qwen-2.5-3B	0.93×	+2.40 GiB	7.03 / 4.00 MiB
Qwen-2.5-7B	0.45×	+3.47 GiB	9.62 / 4.00 MiB
Gemma-7B	0.28×	+3.08 GiB	12.25 / 4.00 MiB
Llama-3.1-8B	0.33×	+4.27 GiB	13.00 / 4.00 MiB

Profiles use one NVIDIA H100 80GB GPU, BF16 precision, sequence length 2,048, batch size 2, gradient accumulation of 8, no gradient checkpointing, 10 warmup steps, and 50 measured steps. Peak memory denotes peak allocated CUDA memory.

Stateless GPart uses 43–69% less fixed adapter-state memory and 0.73–4.27 GiB less peak memory than rank-4 LoRA. Training speed is comparable on the smaller models, but the current implementation is $2.2\times$–$3.5\times$ slower on the evaluated 7B/8B models. Efficient large-model execution and batched multi-adapter serving therefore remain important implementation directions.

G Why the BA Factorization Is Problematic

The BA parametrization introduced by Hu et al. (2022) is a pragmatic choice: it expresses a rank-r update $\Delta W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ using only $r(d_{\text{in}} + d_{\text{out}})$ parameters, a substantial saving when $r \ll \min(d_{\text{in}}, d_{\text{out}})$. Despite its empirical success, this factorization introduces a collection of

theoretical pathologies that have spawned a large body of follow-up work. We identify the two root causes—a representational non-uniqueness and the asymmetric coupling of the bilinear factors—and show that GPart eliminates both by construction.

G.1 Representational non-uniqueness of LoRA

The bilinear factorization $\Delta W = BA$ is not unique. For any invertible matrix $G \in \mathbb{R}^{r\times r}$, the substitution

$$B \longmapsto BG^{-1}, \qquad A \longmapsto GA$$

leaves ΔW unchanged. The set of all pairs that produce a given ΔW therefore forms an equivalence class

$$[(B, A)] = \big\{(BG^{-1}, GA) : G \in GL(r)\big\},$$

and the true object of interest—the linear map ΔW—is an element of the quotient. Any quantity computed from (B, A) that is not invariant under this family of substitutions is a property of the *representation*, not of the underlying adapter.

This non-uniqueness has concrete consequences throughout the LoRA literature:

Routing and merging. Methods that operate directly on the raw (B, A) factorization—such as the rank-one routing of Han et al. (2025) and the clustering-based merging of Zhao et al. (2025)—perform computations on representation-dependent objects. Two adapters that implement identical linear maps ΔW but were trained with different random seeds will in general have different (B, A) pairs, and will therefore be treated as dissimilar by any metric defined on those pairs. The permutation invariance claimed by Zhao et al. (2025) accounts only for the discrete subgroup of signed permutation matrices inside $GL(r)$; the remaining continuous non-uniqueness goes unaddressed.

Scale non-invariance. A special case of this non-uniqueness is the *scale symmetry*: $(B, A) \mapsto (\lambda B, \lambda^{-1}A)$ for any $\lambda \neq 0$. Under this transformation ΔW is unchanged, but any method that computes norms or distances on the columns of B or the rows of A individually will produce different results. This affects, for instance, the token-level routing scores of Han et al. (2025), which score hidden states against columns of B and are therefore arbitrarily sensitive to the optimizer-induced scaling of the factorization.

The SVD as a canonical representative. Ostapenko et al. (2024) and Fleshman & Durme (2025) avoid this problem by working with the singular value decomposition $\Delta W = U\Sigma V^\top$ rather than the raw (B, A) pair. When all singular values are distinct—which holds generically for fully trained adapters, since the set of matrices with repeated singular values has measure zero in $\mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$—this picks a unique representative from each equivalence class, up to a finite group of sign flips on singular vector pairs. Since both methods operate post-hoc on trained adapters, this is a principled and largely complete fix within the LoRA framework. GPart eliminates the non-uniqueness entirely by construction: the map $\theta_d \mapsto P\theta_d$ is injective, so every adapter has a unique representation without requiring a post-hoc canonicalization step.

G.2 Pathologies of the bilinear factorization

The factorization $\Delta W = BA$ introduces two coupled matrices whose asymmetric roles generate a cluster of optimization pathologies. Unlike the representational non-uniqueness discussed in Section G.1, these pathologies are not about which ΔW can be expressed, but about how the optimization landscape over (B, A) behaves relative to the underlying objective over ΔW. GPart eliminates both by replacing the bilinear factorization with a single linear map $\theta_d \mapsto P\theta_d$, removing the coupling entirely.

Asymmetric optimization dynamics. The gradient of the loss with respect to B depends on A and vice versa, creating coupled dynamics that are a structural consequence of the bilinear factorization and are not present in full fine-tuning. Hayou et al. (2024) showed that using the same learning rate for A and B is provably suboptimal for large-width networks, and proposed LoRA+ which assigns different learning rates to the two matrices. Kalajdzievski (2023) showed that the standard α/r scaling causes gradient collapse as rank increases, and derived the corrected $\alpha/\sqrt{r}$ scaling.

Both issues are absent in GPart by construction: since there is no bilinear factorization, there are no coupled factors A and B to which these pathologies can apply.

Initialization pathology. LoRA initializes $B = 0$ so that $\Delta W = BA = 0$ at the start of training, ensuring the fine-tuned model begins from the pretrained weights. This convention is structurally forced by the bilinear factorization: there is no symmetric way to initialize (B, A) to zero output while keeping both matrices in a generic position. The asymmetry has spawned a line of work proposing alternative initializations, including PiSSA (Meng et al., 2024), which initializes from the top-r singular components of W_0, and LoftQ (Li et al., 2023), which handles the quantized setting. Recent work suggests that these gains may be largely attributable to the learning rate regimes they induce rather than the initializations themselves (Lee et al., 2026), which is consistent with the view that the initialization literature is patching a symptom rather than the root cause. In GPart, any initialization of $\boldsymbol{\theta}_d$ produces a well-defined $\Delta W = P\boldsymbol{\theta}_d$ from step one, and the question of how to initialize the adapter does not interact with the structure of the map itself.

G.3 Summary

Table 12 summarizes the pathologies of the BA factorization discussed above, the fixes proposed in the literature, and whether each is absent in GPart by construction.

Table 12: Pathologies of the LoRA BA factorization and their status in GPart. ✓ = present; × = absent by construction. The final column indicates structural absence, not a fix.

Pathology	LoRA	Proposed fix	GPart
$GL(r)$ non-uniqueness	✓	ARROW, SpectR	×
Scale non-invariance	✓	ARROW, SpectR	×
Coupled gradient dynamics	✓	LoRA+, rsLoRA	×
Initialization pathology	✓	PiSSA, LoftQ	×

The pattern is consistent: every identified pathology traces to either the representational non-uniqueness of the BA factorization or the asymmetric coupling of its two factors, and GPart eliminates both simultaneously by replacing the bilinear map with a single linear projection. These differences follow structurally from replacing the bilinear BA factorization with a single linear parameterization, reconnecting the adapter coordinates more directly with the random-subspace perspective of Aghajanyan et al. (2021).

H Partition Structure and Adapter Merging

GPart and intrinsic-dimension fine-tuning with Fastfood share the same broad template: both learn $\theta \in \mathbb{R}^d$ and map it directly into an N-dimensional weight update, $\Delta w = P\theta$. The distinction is therefore not the use of a random subspace itself, but the structure of its embedding. Fastfood implements a structured dense transform and is isometric in expectation over its random factors. In contrast, the normalized GPart partition satisfies $P^\top P = I_d$ for every draw and has exactly one nonzero element per row. Thus, if weight i belongs to group $g(i)$ of size $n_{g(i)}$,

$$(P\theta)_i = \frac{\theta_{g(i)}}{\sqrt{n_{g(i)}}}. \tag{16}$$

This coordinate separability is immaterial for a linear merge, but becomes relevant for nonlinear, coordinatewise merge rules. This is not a niche case: several widely used methods sparsify or select task-vector coordinates before aggregation. TIES removes small-magnitude entries and resolves sign conflicts (Yadav et al., 2023); DARE randomly drops and rescales delta parameters (Yu et al., 2024a); Model Breadcrumbs removes both small changes and magnitude outliers (Davari & Belilovsky, 2024); DELLA assigns drop probabilities using magnitude ranks (Deep et al., 2024); and MagMax selects, at each coordinate, the task-vector entry with the largest absolute magnitude (Marczak et al., 2024). A dense projection changes the coordinates on which all of these masks or decisions are defined.

Table 13: Pairwise TIES merging on GLUE. Geometry is averaged over 15 task pairs at seed 0; merged score is averaged over 15 pairs and three seeds. Lower is better for the two error columns and higher is better for score.

κ	Projection	Comm. residual	Pullback error	Merged score
0.4	GPart partition	0.0090	0.0008	0.7410
0.4	Fastfood	0.6902	0.4918	0.7369
1.0	GPart partition	0.0000	0.0000	0.7426
1.0	Fastfood	0.7050	0.5020	0.7309

Commutation with TIES. Suppose adapters $\theta_1, \ldots, \theta_m$ share the same projection P, let μ denote a merge in adapter coordinates, and let M be the corresponding merge after expansion into weight space. The two routes commute when

$$P\mu(\theta_1, \ldots, \theta_m) = M(P\theta_1, \ldots, P\theta_m). \tag{17}$$

Weighted averaging satisfies Equation equation 17 for every linear P, including both GPart and Fastfood, and hence does not distinguish the two parameterizations. TIES, however, trims entries by magnitude, elects a sign coordinatewise, and averages only sign-consistent entries. These operations are not invariant to a dense change of basis.

For the partition, all expanded entries in a group have the same sign and magnitude within each adapter. The positive factor in Equation equation 16 therefore preserves sign election, and the disjoint mean factors through P. Consequently, without trimming, TIES commutes exactly for arbitrary group sizes. With top-κ trimming, only groups intersecting the trimming boundary can disagree; balanced groups make this effect small. Fastfood mixes many entries of θ into each weight coordinate, so even sign election alone can produce different decisions before and after expansion.

Text experiment. We test this distinction on the six GLUE tasks used in the main paper with RoBERTa-base. Both arms adapt query and value matrices, with $N = 14{,}155{,}776$ and the matched budget $d = 23{,}040$. The GPart arm uses balanced random groups. We merge all $\binom{6}{2} = 15$ task pairs, pairing only adapters with the same projection seed. Table 13 reports geometry for seed 0 and downstream scores over three seeds (45 pair–seed units). The commutation residual is $\|P\mu(\theta) - M(P\theta)\|_2 / \|M(P\theta)\|_2$. The reconstruction error instead merges in weight space and maps back with $P^\top$; for GPart this is the exact least-squares pullback because $P^\top P = I_d$.

The geometric separation is large. At $\kappa = 0.4$, the partition is approximately $77\times$ closer to commuting than Fastfood and loses less than 10^{-3} of the weight-space merge after pullback. At $\kappa = 1$, the partition commutes up to floating-point error, whereas Fastfood retains a relative residual of 0.705; this isolates dense basis mixing and sign election from the trimming step.

TIES is the measured case study rather than the boundary of the argument. Deterministic coordinate-wise rules that depend only on magnitude and sign act groupwise on Equation equation 16; this covers the masking logic behind Model Breadcrumbs and the magnitude-ranking component of DELLA, up to threshold ties. It also makes maximum-magnitude selection, as used by MagMax, commute exactly: the common positive group scale cannot change which adapter has the largest absolute value. Stochastic rules require one extra condition: DARE or DELLA remains groupwise only when the random mask is coupled within each partition group. Independent weight-level masks can split a group and therefore need not commute. Thus, GPart supplies a natural compressed coordinate system for this broader class of sparse or selective merging methods, while the table directly validates only TIES.

The downstream result is encouraging for the text setting of this paper but should be interpreted cautiously. The partition is ahead by 0.41 points at $\kappa = 0.4$ and by 1.17 points without trimming, winning 34 of 45 comparisons in the latter setting. However, the $\kappa = 0.4$ advantage did not pass the pre-specified interaction test ($p = 0.0621$), the density sweep was exploratory, and a separate vision replication did not show an accuracy advantage for the partition. We therefore view merge commutation and compact recoverability as structural benefits of GPart, not as a guarantee of higher merged accuracy. Together with exact per-draw isometry and a linear-time gather/scatter implementation, this is a concrete difference between the partition and a dimension-matched Fastfood projection that is specific to modular adapters.

Finally, merging in native coordinates requires adapters generated with the same projection, including the same seed. Adapters trained with different random partitions inhabit different coordinate systems and must first be expanded to weight space.

Merging in weight space vs. adapter space. The preceding analysis in Appendix G identifies several pathologies arising from the bilinear LoRA factorization $\Delta W = BA$. We now examine how these pathologies affect downstream adapter merging. In particular, we compare the spaces in which different parameterizations perform merging and show that GPart avoids the resulting mismatch by construction.

For clarity, we first consider additive linear merging; the same argument applies to weighted linear combinations up to the corresponding scalar coefficients. Let

$$\Delta W_1 = B_1 A_1, \qquad \Delta W_2 = B_2 A_2$$

denote two trained LoRA updates for the same model weight W_0. Merging the induced weight updates gives

$$\Delta W_1 + \Delta W_2 = B_1 A_1 + B_2 A_2.$$

In contrast, merging the LoRA factors before multiplication gives

$$(B_1 + B_2)(A_1 + A_2) = B_1 A_1 + B_2 A_2 + B_1 A_2 + B_2 A_1.$$

Thus, factor-space merging introduces the cross terms $B_1 A_2$ and $B_2 A_1$, which are not present in either trained update. Consequently, merging the factors is generally not equivalent to merging the corresponding updates in weight space. Moreover, because the factorization BA is non-unique, factor-space operations can depend on the particular representation of an update rather than only on the update itself.

UniLoRA removes the need to directly average independently parameterized LoRA factors. It represents the complete collection of LoRA parameters as a vector $\Theta_D \in \mathbb{R}^D$ and parameterizes this vector using a lower-dimensional vector $\Theta_d \in \mathbb{R}^d$:

$$\Theta_D = P_{\mathrm{U}} \Theta_d, \qquad P_{\mathrm{U}} \in \mathbb{R}^{D \times d}.$$

After reconstructing Θ_D, UniLoRA reshapes its entries into the individual matrices $\{B_\ell, A_\ell\}_{\ell=1}^{L}$ and forms the model updates through

$$\Delta W_\ell = B_\ell A_\ell.$$

The reconstruction of Θ_D from Θ_d is linear and exact within the UniLoRA parameterization. However, the subsequent map from the reconstructed LoRA parameters to model-weight updates remains bilinear. Therefore, although compact-vector merging is well-defined, it does not generally coincide with additive merging of the resulting weight updates: after merging two vectors, the reconstructed factors are added and their product again contains cross terms. UniLoRA thus resolves the ambiguity of directly manipulating separately parameterized LoRA factors, but its compact coordinates still represent LoRA factor space rather than model weight space.

GPart instead maps its trainable vector directly into model-weight space:

$$\Delta W = P_{\mathrm{G}} \Theta_d, \qquad P_{\mathrm{G}} \in \mathbb{R}^{N \times d},$$

where P_{G} is the partition projection defined in Eq. 5. Because this map is linear, vector-space merging is exactly equivalent to weight-space merging:

$$P_{\mathrm{G}}\big(\Theta_d^{(1)} + \Theta_d^{(2)}\big) = P_{\mathrm{G}} \Theta_d^{(1)} + P_{\mathrm{G}} \Theta_d^{(2)} = \Delta W^{(1)} + \Delta W^{(2)}.$$

Unlike UniLoRA, GPart therefore does not require an intermediate reconstruction into bilinear LoRA factors. Its merge coordinates are coordinates of the actual additive model-weight update. This distinction is particularly important for nonlinear coordinate-wise merging rules such as TIES: GPart applies trimming and sign selection directly to a structured weight-space representation, whereas UniLoRA applies them to a compressed representation of the LoRA factors.

Table 14: Average relative merging degradation (%) on GLUE tasks for RoBERTa-base and RoBERTa-large. Results are averaged over 15 task pairs and reported as mean ± standard deviation across three independent seeds. Lower is better. **Bold** denotes the best overall result per merging algorithm; underline denotes the best result within adapter-space merges; * denotes adapter space merging; † denotes full weight space merging.

Model	Method	Linear	TIES ($k = 0.4$)
RoBERTa-base	LoRA*	23.15 ± 4.15	15.81 ± 1.14
	UniLoRA*	22.48 ± 3.49	24.51 ± 2.40
	GPart*	$\underline{19.22 \pm 1.41}$	$\underline{13.03 \pm 0.39}$
	LoRA-Direct-W†	$\mathbf{10.55 \pm 2.10}$	6.97 ± 0.83
	LoRA-W-SVD†	10.78 ± 2.04	$\mathbf{6.96 \pm 0.92}$
RoBERTa-large	LoRA*	26.23 ± 2.95	16.40 ± 1.88
	UniLoRA*	40.71 ± 7.30	35.59 ± 8.01
	GPart*	$\underline{16.41 \pm 1.92}$	$\underline{10.34 \pm 0.54}$
	LoRA-Direct-W†	$\mathbf{9.20 \pm 2.19}$	$\mathbf{7.18 \pm 0.91}$
	LoRA-W-SVD†	9.74 ± 2.26	7.72 ± 1.09

Merging experiments on GLUE. We evaluate the preceding analysis on the six GLUE tasks used in the main paper, using both RoBERTa-base and RoBERTa-large. We merge all $\binom{6}{2} = 15$ task pairs and, for methods with a shared random projection, pair only adapters generated with the same projection seed. LoRA adapters are merged in factor space, whereas UniLoRA and GPart adapters are merged in their respective compact vector spaces.

For comparison, we also evaluate two weight-space LoRA baselines. In LoRA-Direct-W, each LoRA adapter is first converted into its weight update $\Delta W = BA$, after which the updates are merged directly in weight space. In LoRA-W-SVD, the merged weight update is further compressed back into rank-r LoRA factors using an SVD. The difference between these two baselines isolates the effect of rank-constrained refactorization after merging.

GPart achieves the lowest degradation among all adapter-space methods for both model sizes and both merging rules. The advantage is not limited to linear merging, GPart TIES results suggest that its structured coordinates remain effective when the merge rule performs coordinate-wise trimming and sign selection.

The weight-space LoRA baselines obtain still lower degradation. In particular, LoRA-Direct-W reduces linear-merging degradation for RoBERTa-base and for RoBERTa-large. This comparison is important because it shows that the poor performance of factor-space LoRA merging should not be attributed to the learned LoRA updates themselves. Once the same updates are converted to ΔW and merged additively, their degradation decreases substantially.

The small gap between LoRA-Direct-W and LoRA-W-SVD suggests that rank-r refactorization is not the dominant source of error in this setting. For example, the two methods differ by only 0.23 and 0.54 percentage points under linear merging for RoBERTa-base and RoBERTa-large, respectively. The much larger gap between factor-space LoRA and LoRA-Direct-W instead implicates the factor-space merge itself, consistent with the cross terms introduced by $(B_1 + B_2)(A_1 + A_2)$.

GPart does not outperform direct full-weight merging. Rather, its contribution is to approach the behavior of weight-space merging while retaining a compact adapter representation and avoiding the materialization of full weight updates during merging. Its merge coordinates are directly related to the additive model-weight update, so linear merging is exactly preserved and coordinate-wise rules such as TIES operate on structurally meaningful groups.

The comparison with UniLoRA is especially revealing. UniLoRA has the largest degradation in three of the four adapter-space settings and is particularly weak for RoBERTa-large. This result is consistent with the distinction between the two parameterizations: UniLoRA reconstructs the full LoRA factor vector exactly, but the resulting model update is still formed through the bilinear map BA. Consequently, merging compact UniLoRA vectors does not generally correspond to additive

merging of the induced weight updates. GPart avoids this additional bilinear stage by mapping its compact vector directly into weight space.

I LICENSES AND ASSET USAGE

We document the external models and datasets used in this work, together with their source URLs and publicly stated licenses.

Natural language understanding. We use RoBERTa-base and RoBERTa-large, released by Facebook AI under the MIT License and available at `https://huggingface.co/roberta-base` and `https://huggingface.co/roberta-large`. We evaluate on the GLUE benchmark, available at `https://gluebenchmark.com/`; GLUE is a collection of public benchmark tasks whose component datasets are distributed under their respective licenses.

Mathematical reasoning. We use Qwen2.5-0.5B, Qwen2.5-3B, and Qwen2.5-7B from Qwen, available at `https://huggingface.co/Qwen/Qwen2.5-0.5B`, `https://huggingface.co/Qwen/Qwen2.5-3B`, and `https://huggingface.co/Qwen/Qwen2.5-7B`, released under Apache 2.0 license. We also use Gemma-7B, which is gated under Google's Gemma usage license at `https://huggingface.co/google/gemma-7b`, and Llama-3.1-8B, which is distributed under the Llama 3.1 Community License at `https://huggingface.co/meta-llama/Llama-3.1-8B`. For training and evaluation, we use MetaMathQA (`https://huggingface.co/datasets/meta-math/MetaMathQA`), GSM8K (`https://huggingface.co/datasets/openai/gsm8k`), and MATH (`https://github.com/hendrycks/math`); these assets are commonly distributed under permissive academic/open licenses, with MetaMathQA and GSM8K publicly listed under MIT in prior work.

Vision. We use the Vision Transformer models ViT-Base and ViT-Large from Google, available at `https://huggingface.co/google/vit-base-patch16-224` and `https://huggingface.co/google/vit-large-patch16-224` and listed under the Apache 2.0 license. We evaluate on Oxford-IIIT Pets (`https://www.robots.ox.ac.uk/~vgg/data/pets/`), CIFAR-10 (`https://www.cs.toronto.edu/~kriz/cifar.html`), DTD (`https://www.robots.ox.ac.uk/~vgg/data/dtd/`), EuroSAT (`https://github.com/phelber/EuroSAT`), FGVC-Aircraft (`https://www.robots.ox.ac.uk/~vgg/data/fgvc-aircraft/`), RESISC45 (`https://huggingface.co/datasets/timm/resisc45`), and CIFAR-100 (`https://www.cs.toronto.edu/~kriz/cifar.html`).

All assets were used in accordance with their publicly stated terms. We did not use proprietary or closed-access datasets.