REGRA: RESIDUAL-GRADIENT ADAPTATION FOR COMPLEMENTARY LOW-RANK FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adaptation (LoRA) enables parameter-efficient fine-tuning by learning low-rank weight updates. Increasing rank expands adaptation capacity, but does not guarantee that all learned directions improve downstream performance. In a diagnostic experiment with rank-16 LoRA, we show that retaining only the top singular components achieves the best performance, whereas adding further components degrades it. Motivated by this, we propose Residual-Gradient Adaptation (ReGRA), a two-stage method for making more effective use of a fixed rank budget. After the first stage, ReGRA estimates task gradients on a calibration set and removes their projections onto the two-sided subspace induced by the first-stage update. It then aggregates the residual-gradient directions to initialize a second-stage adapter without changing the first-stage update. An orthogonality regularizer encourages the two updates to capture complementary directions. Both updates can be merged into the pretrained weights, incurring no additional inference overhead. Across diverse benchmark sets, ReGRA achieves stronger performance than alternatives with the same total rank and even matches LoRA adapters with twice the rank. Ablation studies and further analysis support the contribution of its core design choices. Code is available at `https://anonymous.4open.science/r/ReGRA-8D6C/`.

1 INTRODUCTION

Large pretrained models have achieved strong performance across a wide range of downstream tasks, alongside scaling parameters and training corpus (Brown et al., 2020; Chen et al., 2021). In deployment, however, full-parameter fine-tuning for particular tasks or domains typically necessitates updating and storing full set of weights, motivating parameter-efficient fine-tuning (PEFT) (Houlsby et al., 2019; He et al., 2021). By optimizing only a small set of task-specific parameters, it significantly reduces the adaptation cost. Among PEFT methods, low-rank adaptation (LoRA) has become a widely used and effective choice, learning compact task-specific updates that can be merged into pretrained weights for inference (Hu et al., 2021).

The principle behind LoRA is to leverage low-rank trainable factors to approximate dense weight updates (Aghajanyan et al., 2021; Hu et al., 2021). Specifically, for a pretrained weight $\boldsymbol{W}_0$, LoRA parameterizes the task-specific update as $\Delta\boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}$, which satisfies $r = \text{rank}(\boldsymbol{B}\boldsymbol{A}) \ll d$, where d denotes the dimension of $\boldsymbol{W}$. $\boldsymbol{W}_0$ remains frozen during the fine-tuning phase. Therefore, it substantially reduces the number of trainable parameters, and the learned update can be merged into W_0 without additional inference overhead. This balance of parameter efficiency and effective task specialization has kept LoRA widely used in resource-constrained settings. Recent advances continue to refine its low-rank principle through adaptive rank allocation (Zhang et al., 2023b), initialization informed by pretrained weights or task gradients (Meng et al., 2024; Wang et al., 2024; Zhang et al., 2025) , optimization strategies tailored to its factorized parameterization (Hayou et al., 2024; Wang et al., 2025b) and adaptive adapter selection and placement(Hayou et al., 2026; Veprikov et al., 2026) Despite different methods to improve LoRA, they leave a fundamental question unanswered: *does the learned update make effective use of its allocated rank?*

To this end, we train a LoRA model with rank $r = 16$ for diagnosis and decompose each learned update $\Delta\boldsymbol{W}$ using singular value decomposition (SVD). Without retraining, we evaluate rank-k approximations formed by retaining the top k singular components. As shown in Figure 1, we find that

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.

$k = 3$ yields the best downstream performance, whereas retaining additional components degrades it. This suggests that the full learned update is *less effective* than a lower-rank approximation of itself. A larger rank budget does not necessarily produce a more useful task-specific update.

Motivated by this observation, we propose Residual-Gradient Adaptation (**ReGRA**), which helps the learned LoRA update make more effective use of its available rank. ReGRA first trains a standard LoRA adapter and then estimates task gradients with a small subset of the training data. To exhaust the allocated rank, we initialize a second-stage adapter with task gradients projected onto the two-sided complement of the first-stage update. With this first-stage estimation, we are able to leave the previously learned components unchanged, yet continually fulfil the update within the allocated rank. During second-stage training, we apply orthogonal regularization to maintain decorrelation between the two stages. After training, both updates can be merged into the base model, introducing no additional inference cost. We evaluate ReGRA using standard LoRA evaluation following (Hu et al., 2023) and expand it to natural language understanding, mathematical reasoning, and code generation. Under the same total-rank budget, ReGRA consistently achieves stronger overall performance across the evaluated benchmarks and even reaches the performance of the double-rank model. Ablation studies and further analysis support the effectiveness of residual-gradient initialization and update decorrelation.

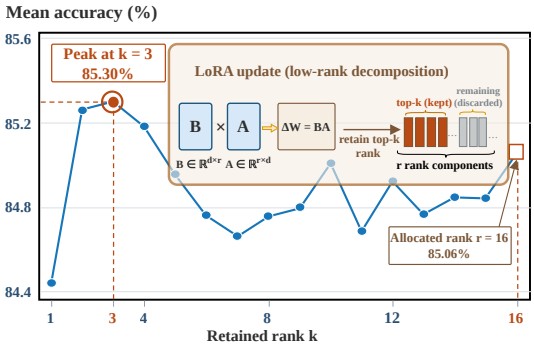


Figure 1: **Effect of retained rank on accuracy.** Model performance does not improve monotonically with increasing retained rank.

Our contributions are summarized as follows:

- We reveal that standard LoRA training does not always make effective use of its allocated rank. We approach this problem from a task-gradient perspective, targeting directions not captured by the initial update.
- We propose ReGRA to exhaust the allocated rank. ReGRA is a two-stage adaptation method that estimates task gradients after standard LoRA training, projects out the components captured by the first-stage update, and uses the residual gradients to initialize a second-stage adapter. An orthogonal regularization maintains the decorrelation between the two updates.
- We evaluate ReGRA on diverse benchmark sets, showing that it outperforms representative baselines and, in some settings, matches the performance of a model with twice the rank. Ablation studies support the efficacy and rationale of our design.

2 Related Work

Low-rank parameter-efficient fine-tuning. Low-rank adaptation (LoRA) represents task-specific weight updates using two trainable low-rank factors, reducing the number of trainable parameters while retaining mergeability at inference (Hu et al., 2021). Subsequent variants improve its memory and parameter efficiency through backbone quantization, factor freezing, or shared random projections (Dettmers et al., 2023; Zhang et al., 2023a; Kopiczko et al., 2024). Other methods explore adaptive rank allocation and rank-dependent scaling (Zhang et al., 2023b; Valipour et al., 2023; Kalajdzievski, 2023). LoRA+, DoRA, and LoRA-Pro instead improve the optimization or parameterization of low-rank updates (Hayou et al., 2024; Liu et al., 2024; Wang et al., 2025b). Recent works also explore adaptive adapter selection and placement to determine where low-rank adaptation is most effective (Hayou et al., 2026; Veprikov et al., 2026). Although these methods improve LoRA from different perspectives, they do not explicitly investigate redundancy among rank components within a learned update matrix.

Spectral and gradient-informed initialization. Several methods initialize LoRA using information from pretrained weights, activations, or downstream data. PiSSA initializes adapters from the

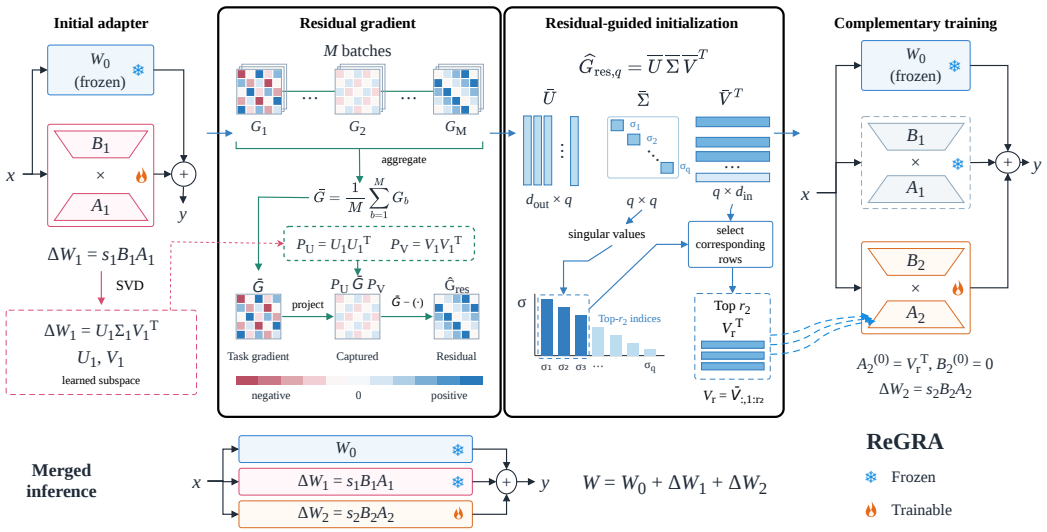


Figure 2: **Overall framework of Residual-Gradient Adaptation (ReGRA).** ReGRA first trains Adapter 1 to learn an initial task-specific update. It then projects the task gradient away from the subspace already captured by Adapter 1 and extracts the dominant residual directions via SVD. These residual directions are used to initialize and guide Adapter 2 toward complementary task-relevant information. The two adapters are finally combined with the frozen backbone to form the final model.

principal singular components of pretrained weights, whereas MiLoRA adapts their minor singular components (Meng et al., 2024; Wang et al., 2025a). CorDA constructs context-aware adaptation subspaces from weight and activation statistics, while EVA selects initialization directions according to activation variance (Yang et al., 2024; Paischer et al., 2026). Gradient-based methods use the task gradient at the start of fine-tuning to guide adapter initialization (Wang et al., 2024; Zhang et al., 2025; Ponkshe et al., 2024). These approaches determine the adaptation subspace before fine-tuning, whereas ReGRA estimates residual task gradients after a first-stage update has been learned.

Sequential and subspace-aware adaptation. ReLoRA and Chain of LoRA (COLA) construct higher-rank effective updates by repeatedly training, merging, and reinitializing low-rank adapters (Lialin et al., 2024; Xia et al., 2024). LoRA-based adaptation has also been studied in continual learning. CoLoR applies LoRA to domain-incremental learning(Wistuba et al., 2023), while O-LoRA assigns sequential tasks to orthogonal low-rank subspaces and InfLoRA constructs adaptation subspaces that reduce interference with prior tasks (Wang et al., 2023; Liang & Li, 2024). KeepLoRA further uses residual-gradient information to balance knowledge retention and plasticity across tasks (Luo et al., 2026). Unlike these methods, ReGRA considers two-stage adaptation to the same downstream objective and combines residual-gradient initialization with update decorrelation to encourage complementary stage-wise updates.

3 Preliminaries

Low-rank adaptation (LoRA). LoRA adapts a frozen pretrained weight $\boldsymbol{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, as shown in Figure 3, by adding a trainable low-rank update (Hu et al., 2021):

$$\boldsymbol{W} = \boldsymbol{W}_0 + \Delta \boldsymbol{W}, \quad \Delta \boldsymbol{W} = s\boldsymbol{B}\boldsymbol{A}. \tag{1}$$

Here, $\boldsymbol{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\boldsymbol{B} \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable factors, r is the adapter rank, and s is the LoRA scaling coefficient. The pretrained weight $\boldsymbol{W}_0$ remains frozen during adaptation. Several extensions modify how LoRA updates are trained or initialized. QLoRA reduces memory use by quantizing the frozen backbone(Dettmers et al., 2023), while AdaLoRA and DyLoRA explore flexible rank allocation (Zhang et al.,

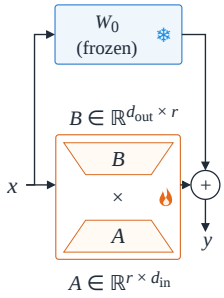


Figure 3: **Illustration of the standard LoRA.**

2023b; Valipour et al., 2023). LoRA-GA uses the initial task gradient to guide adapter initialization (Wang et al., 2024). Our method focuses on learning a second-stage subspace that complements the subspace learned in the first stage.

Singular value decomposition (SVD). For a learned low-rank update $\Delta \boldsymbol{W}$ with $k = \mathrm{rank}(\Delta \boldsymbol{W}) \leq r$, its compact singular value decomposition is

$$\Delta \boldsymbol{W} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\top. \tag{2}$$

The matrices $\boldsymbol{U} \in \mathbb{R}^{d_{\text{out}} \times k}$ and $\boldsymbol{V} \in \mathbb{R}^{d_{\text{in}} \times k}$ contain the left and right singular vectors, respectively, while $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_k)$ contains the positive singular values in descending order. The columns of $\boldsymbol{U}$ span the output-side directions represented by the update, whereas the columns of $\boldsymbol{V}$ span its input-side directions. These singular subspaces are used to identify which components of a task gradient are already represented by a learned low-rank update.

Spectral information has also been used to initialize LoRA adapters: PiSSA selects the principal singular components of the pretrained weights(Meng et al., 2024), while MiLoRA selects the minor components (Wang et al., 2025a). Here, we apply SVD to the first-stage update $\Delta \boldsymbol{W}$ to identify the components of the task gradient already captured by the update.

4 Residual-Gradient Adaptation

We briefly introduce ReGRA to provide an overall understanding. ReGRA decomposes adaptation into two sequential low-rank updates:

$$\boldsymbol{W} = \boldsymbol{W}_0 + \Delta \boldsymbol{W}_1 + \Delta \boldsymbol{W}_2, \qquad \Delta \boldsymbol{W}_i = s_i \boldsymbol{B}_i \boldsymbol{A}_i, \quad i \in \{1, 2\}. \tag{3}$$

We describe ReGRA for a single adapted weight and omit the layer index. The same procedure is applied independently to all target weights. As demonstrated in Figure 2, the first stage follows standard LoRA training, optimizing a rank-r_1 adapter with the objective $\mathcal{L}_{\text{stage1}} = \mathcal{L}_{\text{task}}\left(\boldsymbol{W}_0 + \Delta \boldsymbol{W}_1; \mathcal{D}_{\text{train}}\right)$. ReGRA estimates task gradients and removes the component associated with the first stage, as characterized by the left and right singular subspaces of $\Delta \boldsymbol{W}_1$. The residual gradient is used to initialize a rank-r_2 second stage adapter while preserving the function defined by $\boldsymbol{W}_0 + \Delta \boldsymbol{W}_1$. During second-stage training, we apply an update-level regularizer to promote orthogonality between the first- and second-stage updates. After training, both updates can be merged into the pretrained weight without introducing an additional inference-time branch. The full procedure is listed in Appendix B.5.

4.1 Residual-Guided Initialization

To help the second stage learn complementary components, we first examine the first-stage update. Applying the compact SVD to the fixed first-stage update gives $\Delta \boldsymbol{W}_1 = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^\top$. We define the projectors onto its output- and input-side subspaces as $\boldsymbol{P}_U = \boldsymbol{U}_1 \boldsymbol{U}_1^\top$ and $\boldsymbol{P}_V = \boldsymbol{V}_1 \boldsymbol{V}_1^\top$, respectively. Let $\{\mathcal{C}_b\}_{b=1}^M$ be calibration batches taken from the training data. At the first-stage solution, we compute the task gradient with respect to the adapted weight: $\boldsymbol{G} = \nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}(\boldsymbol{W}_0 + \Delta \boldsymbol{W}_1; \mathcal{C}_b)$. We remove its component in the two-sided subspace represented by the first-stage update, as derived in Appendix A.1:

$$\boldsymbol{G}_{\text{res}} = \boldsymbol{G} - \boldsymbol{P}_U \boldsymbol{G}_b \boldsymbol{P}_V. \tag{4}$$

Let $\widehat{\boldsymbol{G}}_{\text{res},q} = \overline{\boldsymbol{U}} \overline{\boldsymbol{\Sigma}} \overline{\boldsymbol{V}}^\top$ denote the rank-q truncated SVD approximation of $\widehat{\boldsymbol{G}}_{\text{res}}$, , obtained by batch-wise low-rank aggregation as detailed in Appendix A.2. The calibration procedure is described in Appendix B.2. For a rank-r_2 second-stage adapter with $r_2 \leq q$, define $\boldsymbol{V}_r = \overline{\boldsymbol{V}}_{:,1:r_2}$, $\boldsymbol{U}_r = \overline{\boldsymbol{U}}_{:,1:r_2}$, and $\boldsymbol{\Sigma}_r = \overline{\boldsymbol{\Sigma}}_{1:r_2,1:r_2}$. ReGRA initializes the adapter as

$$\boldsymbol{A}_2^{(0)} = \boldsymbol{V}_r^\top, \qquad \boldsymbol{B}_2^{(0)} = \boldsymbol{0}. \tag{5}$$

This initialization specifies the adapter's input-side subspace using the leading right singular vectors of the residual-gradient estimate. The following theorem characterizes its initial effect on the model and its first task-gradient update.

Proposition 1 (Residual-guided first-step update)**.** *Consider $\Delta \boldsymbol{W}_2 = s_2 \boldsymbol{B}_2 \boldsymbol{A}_2$ initialized as in Equation equation 5. The initialization preserves the Stage-1 model function. After one simultaneous gradient-descent step on the task loss, the effective update lies in the input-side subspace*

spanned by $\boldsymbol{V}_r$. If the initial task gradient equals $\widehat{\boldsymbol{G}}_{\mathrm{res},q}$, this update is proportional to its negative leading rank-r_2 component. The proof is given in Appendix A.3.

Remark. Since $\boldsymbol{B}_2^{(0)} = \mathbf{0}$, we have $\Delta\boldsymbol{W}_2^{(0)} = \mathbf{0}$. Let $\boldsymbol{G}^{(0)}$ be the task gradient at $\boldsymbol{W}_0 + \Delta\boldsymbol{W}_1$. One simultaneous gradient-descent step with learning rate η gives $\Delta\boldsymbol{W}_2^{(1)} = -\eta\boldsymbol{G}^{(0)}\boldsymbol{V}_r\boldsymbol{V}_r^\top$. In the special case $\boldsymbol{G}^{(0)} = \widehat{\boldsymbol{G}}_{\mathrm{res},q}$, this reduces to $\Delta\boldsymbol{W}_2^{(1)} = -\eta\boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^\top$. These expressions describe the first plain gradient-descent step, not the entire Stage-2 trajectory. Both factors are subsequently optimized, and the residual-gradient projection is not applied to later task gradients.

4.2 Stage-Wise Orthogonality Regularization

Residual-gradient initialization guides the second-stage adapter toward directions associated with the remaining task gradient. As training proceeds, the second-stage update can move away from these initial directions. ReGRA therefore applies an update-level regularizer to promote orthogonality between the two stage-wise updates. For each adapted weight, let $\boldsymbol{D}_1 = \boldsymbol{B}_1\boldsymbol{A}_1$ and $\boldsymbol{D}_2 = \boldsymbol{B}_2\boldsymbol{A}_2$ denote the unscaled first- and second-stage updates. We define the orthogonality penalty as

$$\Omega = \frac{\langle \boldsymbol{D}_1, \boldsymbol{D}_2\rangle_F^2}{\|\boldsymbol{D}_1\|_F^2\|\boldsymbol{D}_2\|_F^2 + \varepsilon}, \tag{6}$$

where $\varepsilon > 0$ ensures numerical stability. Squaring the inner product penalizes deviations from orthogonality in either direction. The penalty promotes orthogonal updates. For L adapted weights, we average the penalties: $\mathbb{R}_{\mathrm{decorr}} = \frac{1}{L}\sum_{\ell=1}^{L}\Omega_\ell$. The inner products and norms can be computed from the LoRA factors without forming dense update matrices. The derivation is given in Appendix A.4.

During second-stage training, $\Delta\boldsymbol{W}_1$ remains fixed, while $\boldsymbol{A}_2$ and $\boldsymbol{B}_2$ are optimized using

$$\mathcal{L}_{\mathrm{stage2}} = \mathcal{L}_{\mathrm{task}}\left(\boldsymbol{W}_0 + \Delta\boldsymbol{W}_1 + \Delta\boldsymbol{W}_2; \mathcal{D}_{\mathrm{train}}\right) + \lambda_t\mathbb{R}_{\mathrm{decorr}}. \tag{7}$$

We set $\lambda_t = 0.1$ throughout the second stage in all experiments.

Computation and inference. Residual-gradient calibration introduces a one-time cost between the two training stages. Each dense task gradient is formed only transiently, projected, compressed to rank q, and then discarded, while cross-batch aggregation and update decorrelation operate on low-rank factors. After training, the two learned updates are merged into the pretrained weight as $\boldsymbol{W}_{\mathrm{merged}} = \boldsymbol{W}_0 + s_1\boldsymbol{B}_1\boldsymbol{A}_1 + s_2\boldsymbol{B}_2\boldsymbol{A}_2$. The Stage-2 procedure is detailed in Appendix B.4. ReGRA therefore introduces neither an additional adapter branch nor extra computation during inference, storage and computation bounds are given in Appendix B.6.

5 Experiments

To validate the efficacy and rationale of our design, we conduct experiments to address the following research questions (RQs):

- **RQ1:** How does ReGRA compare with baselines across diverse benchmark sets, and how does it compare with LoRA with larger rank capacity (*i.e.*, $r = 8$ *v.s.* $r = 16$)?
- **RQ2:** What are the separate and combined effects of residual-gradient initialization and update decorrelation?
- **RQ3:** Which singular components of a trained LoRA update affect performance when removed, and what do isolated components of ReGRA's second-stage update contribute?
- **RQ4:** How does performance change as rank-2 stages are progressively added?

5.1 Experimental Setup

Tasks and datasets. We evaluate ReGRA on natural language understanding (Wang et al., 2018), mathematical reasoning (Hu et al., 2023; Cobbe et al., 2021),and code generation (Chen et al., 2021; Austin et al., 2021). For natural language understanding, we fine-tune RoBERTa-base and RoBERTa-large (Liu et al., 2019) separately on the eight GLUE tasks: CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, and RTE. We evaluate all tasks on their labeled validation sets, with the

Table 1: **Performance (%) on eight GLUE natural language understanding tasks** using RoBERTa-base and RoBERTa-large. Avg. is the unweighted mean across tasks. We calculate the performance gain of ReGRA against the rank-8 LoRA on the same backbone. For ReGRA, $r = 8$ denotes the rank of each stage. The best average for each backbone is marked in bold.

	Method	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	Avg.
RoBERTa-base	LoRA ($r = 16$)	60.24	85.75	87.25	91.45	87.94	75.09	93.69	90.31	83.97
	LoRA ($r = 8$)	58.04	85.11	87.50	90.99	87.16	72.92	93.92	90.07	83.21
	DoRA ($r = 8$)	56.53	86.20	88.23	91.21	85.43	74.36	93.92	90.05	83.24
	PiSSA ($r = 8$)	56.01	84.16	89.12	90.49	87.43	72.56	93.23	89.56	82.82
	LoRA-One ($r = 8$)	58.39	85.28	89.12	91.15	85.51	75.81	94.38	90.54	83.77
	KeepLoRA ($r = 8$)	58.55	84.76	90.35	90.93	84.98	67.87	93.69	90.42	82.69
	ReGRA ($r = 8$)	$61.60_{+3.56}$	$85.96_{+0.85}$	$89.43_{+1.93}$	$91.95_{+0.96}$	$88.15_{+0.99}$	$75.81_{+2.89}$	$94.38_{+0.46}$	$90.66_{+0.59}$	**84.74**$_{+1.53}$
RoBERTa-large	LoRA ($r = 16$)	64.55	90.59	90.33	94.58	87.57	85.92	96.22	92.20	87.75
	LoRA ($r = 8$)	66.02	90.39	89.70	94.60	88.93	84.12	96.22	91.83	87.73
	DoRA ($r = 8$)	65.53	90.52	91.04	94.34	87.53	84.11	96.10	91.90	87.63
	PiSSA ($r = 8$)	60.59	89.89	90.91	94.08	86.88	83.75	96.33	91.72	86.77
	LoRA-One ($r = 8$)	65.30	90.35	91.34	94.63	87.41	85.19	95.98	92.30	87.81
	KeepLoRA ($r = 8$)	60.72	89.35	91.71	92.62	86.04	85.19	94.61	91.67	86.49
	ReGRA ($r = 8$)	$68.82_{+2.80}$	$90.45_{+0.06}$	$91.76_{+2.06}$	$94.60_{0.00}$	$87.77_{-1.16}$	$85.92_{+1.80}$	$96.22_{0.00}$	$91.85_{+0.02}$	**88.42**$_{+0.69}$

matched validation split used for MNLI. For mathematical reasoning (Hu et al., 2023), we fine-tune Llama-3-8B (Grattafiori et al., 2024) and Qwen3-8B-Base (Yang et al., 2025) on Math-7K training set released with LLM-Adapters (Hu et al., 2023). We evaluate on AddSub, MultiArith, SingleEq, GSM8K, AQuA, and SVAMP. For code generation, we fine-tune Llama-3-8B on CodeFeedback-Filtered-Instruction (Zheng et al., 2024) and evaluate on MBPP (Austin et al., 2021) and its enhanced MBPP+ test suite using EvalPlus (Liu et al., 2023).

Implementation details. Adapters are applied to the query, key, and value projections while the pretrained backbones remain frozen, details are provided in Appendix B.1. Each ReGRA stage uses rank 8, $\alpha = 16$, and dropout 0.05; the GLUE classification head is trained in the first stage and frozen in the second stage. Generative losses are computed only on response tokens. We use zero weight decay, and $\lambda_t = 0.1$, whose contribution is warmed up over the first 100 second-stage steps. The training schedules for GLUE, mathematics, and code generation comprise 20, 2, and 1 epochs, respectively, full settings are tabulated in Appendix B.3.

Baselines and evaluation. We compare ReGRA with LoRA (Hu et al., 2021), DoRA (Liu et al., 2024), PiSSA (Meng et al., 2024), LoRA-One (Zhang et al., 2025), and KeepLoRA (Luo et al., 2026). LoRA is evaluated at ranks 8 and 16, while the other baselines use rank 8. For ReGRA, $r = 8$ refers to the rank of each stage. GLUE uses the standard task-specific metrics: Matthews correlation for CoLA, the mean of Pearson and Spearman correlations for STS-B, the mean of accuracy and F_1 for MRPC and QQP, and accuracy otherwise. Mathematical reasoning uses answer accuracy, and code generation uses pass@1. All scores are reported as percentages.

5.2 Main Results (RQ1)

As shown in Table 1, ReGRA achieves the highest average score on both backbones. Increasing the rank of standard LoRA from 8 to 16 also improves its average score, confirming that additional low-rank capacity is useful. ReGRA obtains a further improvement at the same maximum update rank by guiding how that capacity is used. After learning the first-stage update, ReGRA initializes the second adapter from task-gradient directions outside the subspace associated with that update. Update-level decorrelation then discourages the two stage-wise updates from aligning during training. Standard rank-16 LoRA optimizes all rank components jointly without this explicit guidance toward complementary updates. The comparison supports the benefit of directing added capacity toward optimization directions left insufficiently covered by the first stage. Gains on CoLA, MRPC, and RTE further support this pattern.

Table 2: **Accuracy (%) on six mathematical reasoning benchmarks** using Llama-3-8B and Qwen3-8B-Base. Avg. is the unweighted mean across tasks. Subscript values beside ReGRA scores show percentage-point differences from rank-8 LoRA on the same backbone. For ReGRA, $r = 8$ denotes the rank of each stage. The best average for each backbone is marked in bold.

	Method	AddSub	MultiArith	SingleEq	SVAMP	GSM8K	AQuA	Avg.
Llama-3-8B	LoRA ($r = 16$)	79.74	92.83	87.59	69.80	60.50	39.37	71.64
	LoRA ($r = 8$)	76.20	92.66	87.99	69.30	61.41	37.40	70.83
	DoRA ($r = 8$)	77.72	94.16	86.41	72.10	60.34	39.76	71.75
	PiSSA ($r = 8$)	74.93	89.00	85.23	67.80	58.07	43.30	69.72
	LoRA-One ($r = 8$)	79.24	91.33	87.99	70.30	60.65	40.15	71.61
	KeepLoRA ($r = 8$)	80.50	87.33	88.18	70.50	60.80	37.79	70.85
	ReGRA ($r = 8$)	79.24$_{+3.04}$	89.50$_{-3.16}$	88.19$_{+0.20}$	72.40$_{+3.10}$	62.55$_{+1.14}$	40.94$_{+3.54}$	**72.14**$_{+1.31}$
Qwen3-8B-Base	LoRA ($r = 16$)	91.65	98.50	95.08	85.90	77.41	61.42	84.99
	LoRA ($r = 8$)	91.39	98.50	94.09	86.00	75.28	59.84	84.18
	DoRA ($r = 8$)	91.65	98.83	94.29	84.80	75.97	62.99	84.76
	PiSSA ($r = 8$)	89.37	98.33	94.29	85.40	76.19	61.02	84.10
	LoRA-One ($r = 8$)	92.15	99.00	94.88	84.90	76.12	60.63	84.61
	KeepLoRA ($r = 8$)	86.84	97.50	94.09	81.90	77.48	62.99	83.47
	ReGRA ($r = 8$)	92.91$_{+1.52}$	98.67$_{+0.17}$	95.28$_{+1.19}$	85.70$_{-0.30}$	75.51$_{+0.23}$	63.30$_{+3.46}$	**85.23**$_{+1.05}$

Table 3: **Pass@1 (%) on the MBPP and enhanced MBPP+ tests** using Llama-3-8B. For ReGRA, $r = 8$ denotes the rank of each stage. The best result in each column is marked in bold.

Backbone	Method	MBPP	MBPP+
Llama-3-8B	LoRA ($r = 16$)	67.98	57.14
	LoRA ($r = 8$)	65.87	55.02
	DoRA ($r = 8$)	66.40	55.55
	PiSSA ($r = 8$)	66.93	54.49
	LoRA-One ($r = 8$)	66.13	54.76
	KeepLoRA ($r = 8$)	67.46	56.35
	ReGRA ($r = 8$)	**68.78**$_{+2.91}$	**57.41**$_{+2.39}$

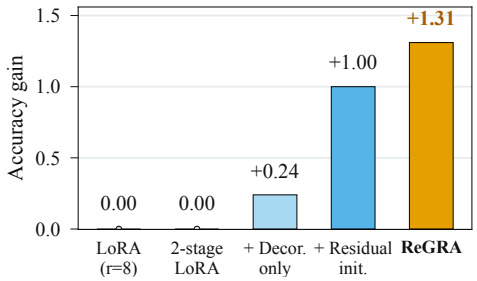


Figure 4: **Component ablation of ReGRA.** Llama-3-8B gains in six-task average accuracy, in percentage points, relative to LoRA ($r = 8$). Each configuration used one seed.

5.3 Results on Specialized Capabilities

Mathematical reasoning (RQ1). Table 2 shows that ReGRA achieves the highest six-task average on both Llama-3-8B and Qwen3-8B-Base. The task-level pattern differs between the backbones. On Llama-3-8B, the gains over rank-8 LoRA are concentrated on AddSub, SVAMP, and AQuA, while MultiArith declines. On Qwen3-8B-Base, AddSub, SingleEq, and AQuA benefit most, whereas ReGRA does not lead on every task. Its average also exceeds rank-16 LoRA on both backbones, though the margin is small on Qwen3-8B-Base. These results show that the overall mathematical-reasoning benefit carries across two generative backbones, with different task-level patterns.

Code generation (RQ1). Table 3 shows that ReGRA obtains the highest pass@1 on both the base and enhanced MBPP+ tests. All methods score lower on the enhanced tests, but ReGRA retains its lead under the stricter evaluation. Its advantage over rank-16 LoRA is modest on both test variants, so the code result supports a consistent improvement within this benchmark rather than a large gain in code generation. The commonsense comparison is reported in Appendix C.1.

5.4 Ablation Analysis (RQ2)

To disentangle the effects of the two key design components of ReGRA, we conduct a controlled 2×2 ablation study on the mathematical reasoning benchmarks. All variants share the same rank-8 first-stage adapter and train an additional rank-8 Stage-2 adapter under identical optimization settings. When residual-gradient initialization is removed, the second-stage adapter adopts the standard

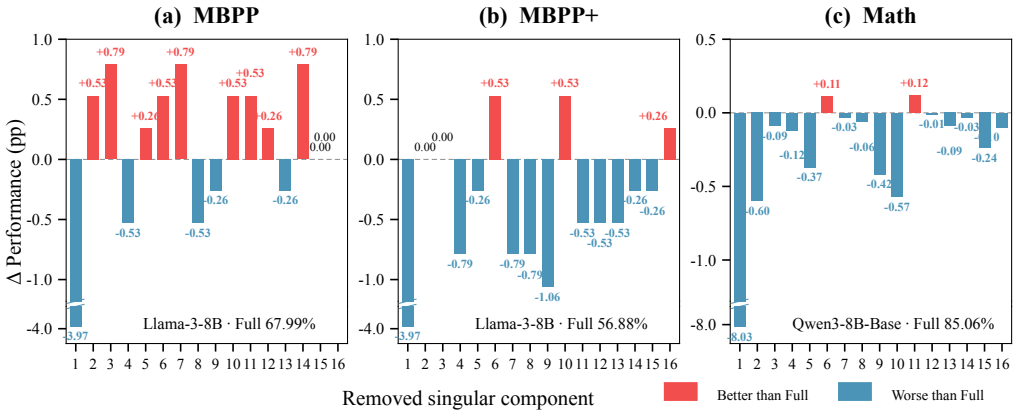


Figure 5: **Functional importance of LoRA singular components.** For rank-16 LoRA, we apply SVD to each learned update ΔW and remove one component at a time. Bars show performance changes relative to the full update (dashed zero line) for Llama-3-8B on MBPP (a) and MBPP+ (b), and Qwen3-8B-Base on Math (c). Removing the leading component reduces performance by 3.97 percentage points on both code tasks and 8.03 percentage points on Math. Red and blue indicate gains and losses, respectively; broken axes display the larger leading-component effect.

LoRA initialization, with a randomly initialized input-side factor and a zero-initialized output-side factor. When update-level decorrelation is removed, the corresponding regularization term is omitted from the Stage-2 objective. Performance is measured by the unweighted average accuracy across the six mathematical reasoning datasets.

As shown in Figure 4, residual-gradient initialization provides the dominant improvement, supporting the motivation of directing second-stage adaptation toward gradient information insufficiently captured by the first-stage. Update-level decorrelation provides a smaller additional benefit by discouraging redundancy between the two updates. The full model achieves the strongest overall performance, suggesting that the two components play complementary roles: the former determines informative adaptation directions, while the latter encourages the resulting update to remain distinct from the first-stage.

5.5 More Analysis

Unequal Functional Contributions of LoRA Singular Components (RQ3). We further examine how individual directions in a trained rank-16 LoRA update affect downstream performance. We decompose each learned update using SVD, remove one singular component at a time, and evaluate the resulting model. As shown in Figure 5, removing the leading singular component produces the largest performance drop in all three evaluations: 3.97 percentage points on both MBPP and MBPP+, and 8.03 percentage points on Math. Removing most of the remaining components has a much smaller effect, and removing some even improves performance slightly.

These leave-one-component-out results measure the functional effect of each removal, beyond the magnitude of its singular value. Together with the top-k analysis in Figure 1, they indicate that the learned LoRA directions do not contribute uniformly to task performance. This observation motivates ReGRA to seek complementary directions from the residual gradient after the first-stage update has been learned.

Functional diversity of Stage-2 components (RQ3). To examine the functional contribution of the learned Stage-2 update, we decompose each trained Stage-2 update matrix by SVD and evaluate eight independent variants. Variant C_i retains only the ith rank-one component in each adapted matrix while keeping Stage 1 fixed; the variants are not cumulative. As shown in Figure 6, their effects differ across task families. On the six mathematical reasoning benchmarks, only C_3, C_4, and C_6 improve mean accuracy over Stage 1-only, by 2.91, 1.27, and 0.56 percentage points, respectively; the other five components produce small decreases. The full Stage-2 update gains 1.27 points, below C_3 alone. On MBPP+, seven components improve pass@1 by 0.79–1.85 points, while C_6 leaves it unchanged; the full update gains 2.65 points. These results show heterogeneous and task-dependent

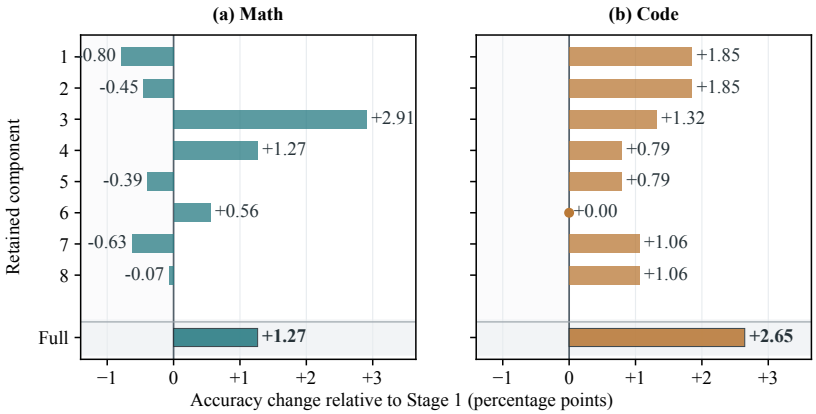


Figure 6: **Effects of isolated Stage-2 SVD components on Llama-3-8B.** Each C_i retains only the ith rank-one component of every Stage-2 update matrix on a fixed Stage-1 model; components are evaluated independently, not cumulatively. *Full* retains all eight components. Bars show percentage-point changes from Stage 1-only in (a) mean accuracy across six math benchmarks and (b) MBPP+ pass1 on 378 problems. Positive and negative bars indicate gains and losses, respectively. Results are from one deterministic evaluation per condition.

component effects. The full update's benchmark performance is not a sum of its isolated component effects, although these independent evaluations do not identify each component's marginal contribution within the full update, example-level error analyses are reported in Appendix C.2.

Progressive Rank Expansion (RQ4). We further evaluate our method on LLaMA-3-8B under a multi-stage training setting. Starting from the base model, we iteratively train a new rank-2 adapter at each stage while freezing all previously learned adapters. Therefore, the effective adaptation rank increases progressively as more adapters are accumulated. As shown in Fig. 7, the overall performance consistently improves with the increasing cumulative rank, rising from 69.20% at rank 2 to 71.81% at rank 16. Although minor fluctuations are observed at intermediate ranks, the overall trend demonstrates that newly introduced adapters continue to provide complementary adaptation capacity rather than merely duplicating previously learned directions. These results support the effectiveness of our additive multi-stage training strategy for progressively expanding the model's adaptation capacity.

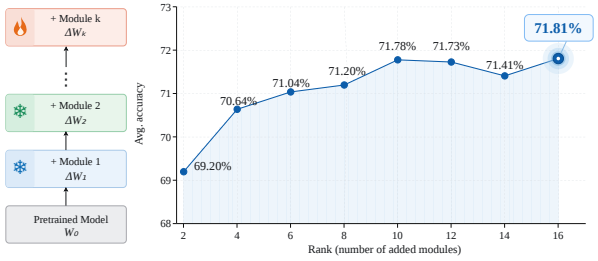


Figure 7: **Progressive rank expansion** on LLaMA-3-8B. At each stage, previously learned adapters are frozen and a new rank-2 adapter is trained.

6 Conclusion

In this work, we introduced ReGRA, a two-stage adaptation method that makes more effective use of a fixed total-rank budget. Motivated by the observation that additional LoRA singular components do not always improve performance, ReGRA uses residual task gradients to initialize a second adapter while preserving the first-stage solution. An orthogonality regularizer helps maintain separation between the two updates, which can be merged without additional inference overhead. Experiments on natural language understanding, mathematical reasoning, and code generation demonstrate improvements over rank-matched LoRA, while ablations support both design choices. These findings suggest that effective low-rank adaptation depends not only on increasing rank, but also on learning useful additional components relative to an existing update. Future work may investigate more principled ways to identify and allocate useful residual directions under a fixed rank budget.

AI USE STATEMENT

AI tools were used to assist with writing and language polishing, and to support literature retrieval and the discovery of related work. Their use was limited to these purposes. The authors take full responsibility for the scientific content, the accuracy of the references, and the final manuscript.

ETHICS STATEMENT

This work studies parameter-efficient adaptation of pretrained language models. Our experiments use existing benchmark datasets and pretrained models for natural language understanding, mathematical reasoning, code generation, and commonsense reasoning. The use and redistribution of these resources are subject to their respective licenses and terms of use. The proposed adaptation method does not itself address potential biases or harmful outputs inherited from the underlying models and training data.

REPRODUCIBILITY STATEMENT

We describe the ReGRA method in Section 4 and the experimental setup in Section 5.1. Appendix B provides training hyperparameters, calibration settings, and implementation details, while Algorithm 1 summarizes the complete procedure. The source code is available at `https://anonymous.4open.science/r/ReGRA-8D6C/`.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing (volume 1: long papers)*, pp. 7319–7328, 2021.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

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.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Plop: Precise lora placement for efficient finetuning of large models. In *International Conference on Learning Representations*, volume 2026, pp. 65166–65191, 2026.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*, 2021.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pp. 2790–2799. PMLR, 2019.

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. *arXiv preprint arXiv:2106.09685*, 2021.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 5254–5276, 2023.

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

Dawid Kopiczko, Tijmen Blankevoort, and Yuki Asano. Vera: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, volume 2024, pp. 6815–6835, 2024.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. Relora: High-rank training through low-rank updates. In *International Conference on Learning Representations*, volume 2024, pp. 49405–49421, 2024.

Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23638–23647. IEEE, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 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. *arXiv preprint arXiv:2402.09353*, 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. *arXiv preprint arXiv:1907.11692*, 2019.

Mao-Lin Luo, Zi-Hao Zhou, Yi-Lin Zhang, Yuanyu Wan, Min-Ling Zhang, and Tong Wei. Keeplora: Continual learning with residual gradient adaptation. In *International Conference on Learning Representations*, volume 2026, pp. 136017–136040, 2026.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37:121038–121072, 2024.

Fabian Paischer, Lukas Hauzenberger, Thomas Schmied, Benedikt Alkin, Marc Deisenroth, and Sepp Hochreiter. Parameter efficient fine-tuning via explained variance adaptation. *Advances in Neural Information Processing Systems*, 38:46215–46267, 2026.

Kaustubh Ponkshe, Raghav Singhal, Eduard Gorbunov, Alexey Tumanov, Samuel Horvath, and Praneeth Vepakomma. Initialization using update approximation is a silver bullet for extremely efficient low-rank fine-tuning. *arXiv preprint arXiv:2411.19557*, 2024.

Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 3274–3287, 2023.

Andrey Veprikov, Vladimir Solodkin, Zyl Alexander, Andrey Savchenko, and Aleksandr Beznosikov. Weightlora: Keep only necessary adapters. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12419–12437, 2026.

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 *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, pp. 353–355, 2018.

Hanqing Wang, Yixia Li, Shuo Wang, Guanhua Chen, and Yun Chen. Milora: Harnessing minor singular components for parameter-efficient llm finetuning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4823–4836, 2025a.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, 2023.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Lora-pro: Are low-rank adapters properly optimized? In *International Conference on Learning Representations*, volume 2025, pp. 93787–93808, 2025b.

Martin Wistuba, Prabhu Teja Sivaprasad, Lukas Balles, and Giovanni Zappella. Continual learning with low rank adaptation. *arXiv preprint arXiv:2311.17601*, 2023.

Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of LoRA: Efficient fine-tuning of language models via residual learning. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yibo Yang, Xiaojie Li, Zhongzhu Zhou, Shuaiwen L Song, Jianlong Wu, Liqiang Nie, and Bernard Ghanem. Corda: Context-oriented decomposition adaptation of large language models for task-aware parameter-efficient fine-tuning. *Advances in neural information processing systems*, 37: 71768–71791, 2024.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv e-prints*, pp. arXiv–2308, 2023a.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023b.

Yuanhe Zhang, Fanghui Liu, and Yudong Chen. Lora-one: One-step full gradient could suffice for fine-tuning large language models, provably and efficiently. *arXiv preprint arXiv:2502.01235*, 2025.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 12834–12859, 2024.

Appendix

CONTENTS

A Theoretical Analysis **13**

A.1 Residual-Gradient Projection and Orthogonality 13

A.2 Low-Rank Calibration and Gradient Aggregation 14

A.3 Proof of the Residual-Guided First-Step Update 14

A.4 Update Orthogonality Regularization 15

B Supplementary Implementation Details **15**

B.1 Model and Training Setup 15

B.2 Residual-Gradient Calibration 16

B.3 Hyperparameter Settings 16

B.4 Stage-2 Optimization and Merging 16

B.5 Algorithm Flow 17

B.6 Computational Cost 17

C Additional Results **18**

C.1 Commonsense Reasoning 18

C.2 Error-Level Analysis of Stage-2 Components 18

A THEORETICAL ANALYSIS

In this section, we analyze the residual-gradient projection (Appendix A.1), its low-rank aggregation (Appendix A.2), the initial Stage-2 update (Appendix A.3), and the factorized decorrelation penalty (Appendix A.4).

A.1 RESIDUAL-GRADIENT PROJECTION AND ORTHOGONALITY

Consider an adapted weight $\boldsymbol{W}_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. For the fixed first-stage update, define

$$\begin{aligned} \Delta \boldsymbol{W}_1 &= s_1 \boldsymbol{B}_1 \boldsymbol{A}_1 = \boldsymbol{U}_1 \boldsymbol{\Sigma}_1 \boldsymbol{V}_1^\top, \\ \boldsymbol{P}_U &= \boldsymbol{U}_1 \boldsymbol{U}_1^\top, \qquad \boldsymbol{P}_V = \boldsymbol{V}_1 \boldsymbol{V}_1^\top. \end{aligned} \tag{8}$$

For a calibration batch $\mathcal{C}_b$, the gradient and its residual are

$$\begin{aligned} \boldsymbol{G}_b &= \nabla_{\boldsymbol{W}} \mathcal{L}_{\text{task}}(\boldsymbol{W}_0 + \Delta \boldsymbol{W}_1; \mathcal{C}_b), \\ \boldsymbol{G}_{\text{res},b} &= \boldsymbol{G}_b - \boldsymbol{P}_U \boldsymbol{G}_b \boldsymbol{P}_V \\ &= \boldsymbol{G}_b - \boldsymbol{U}_1 \boldsymbol{U}_1^\top \boldsymbol{G}_b \boldsymbol{V}_1 \boldsymbol{V}_1^\top. \end{aligned} \tag{9}$$

The residual is orthogonal to every matrix in the two-sided subspace represented by the first-stage update. Indeed, for any compatible $\boldsymbol{Z}$,

$$\begin{aligned} \left\langle \boldsymbol{G}_{\text{res},b}, \boldsymbol{U}_1 \boldsymbol{Z} \boldsymbol{V}_1^\top \right\rangle_F &= \left\langle \boldsymbol{U}_1^\top \boldsymbol{G}_{\text{res},b} \boldsymbol{V}_1, \boldsymbol{Z} \right\rangle_F \\ &= \left\langle \boldsymbol{U}_1^\top \left(\boldsymbol{G}_b - \boldsymbol{P}_U \boldsymbol{G}_b \boldsymbol{P}_V \right) \boldsymbol{V}_1, \boldsymbol{Z} \right\rangle_F \\ &= \left\langle \boldsymbol{U}_1^\top \boldsymbol{G}_b \boldsymbol{V}_1 - \boldsymbol{U}_1^\top \boldsymbol{G}_b \boldsymbol{V}_1, \boldsymbol{Z} \right\rangle_F \\ &= 0. \end{aligned} \tag{10}$$

Thus, the projection removes components jointly aligned with the left and right first-stage subspaces. Unlike $(\boldsymbol{I} - \boldsymbol{P}_U)\boldsymbol{G}_b(\boldsymbol{I} - \boldsymbol{P}_V)$, it retains components aligned with only one of these subspaces.

A.2 Low-Rank Calibration and Gradient Aggregation

For each calibration batch, we truncate the residual gradient immediately after projection. Let $\mathcal{T}_q(\cdot)$ denote the rank-at-most-q matrix returned by the low-rank SVD routine. Then

$$\begin{aligned} \boldsymbol{G}_{\mathrm{res},b} &= \boldsymbol{G}_b - \boldsymbol{P}_U \boldsymbol{G}_b \boldsymbol{P}_V, \\ \widetilde{\boldsymbol{G}}_{\mathrm{res},b} &:= \mathcal{T}_q\big(\boldsymbol{G}_{\mathrm{res},b}\big) \\ &= \boldsymbol{U}_b^{(g)} \boldsymbol{\Sigma}_b^{(g)} \big(\boldsymbol{V}_b^{(g)}\big)^\top, \\ \widehat{\boldsymbol{G}}_{\mathrm{res}} &:= \frac{1}{M} \sum_{b=1}^{M} \widetilde{\boldsymbol{G}}_{\mathrm{res},b} \\ &= \frac{1}{M} \sum_{b=1}^{M} \boldsymbol{U}_b^{(g)} \boldsymbol{\Sigma}_b^{(g)} \big(\boldsymbol{V}_b^{(g)}\big)^\top. \end{aligned} \tag{11}$$

The superscript (g) distinguishes the batch-gradient singular vectors from those of the first-stage update. The mean can be aggregated without constructing a dense matrix. Define the concatenated factors

$$\begin{aligned} \mathbf{L} &= \left[\frac{\boldsymbol{U}_1^{(g)} \boldsymbol{\Sigma}_1^{(g)}}{M}, \ldots, \frac{\boldsymbol{U}_M^{(g)} \boldsymbol{\Sigma}_M^{(g)}}{M} \right], \\ \mathbf{R} &= \left[\boldsymbol{V}_1^{(g)}, \ldots, \boldsymbol{V}_M^{(g)} \right]. \end{aligned} \tag{12}$$

Using thin QR factorizations $\mathbf{L} = \mathbf{Q}_L \mathbf{R}_L$ and $\mathbf{R} = \mathbf{Q}_R \mathbf{R}_R$, we obtain

$$\begin{aligned} \widehat{\boldsymbol{G}}_{\mathrm{res}} &= \mathbf{L}\mathbf{R}^\top \\ &= (\mathbf{Q}_L \mathbf{R}_L)(\mathbf{Q}_R \mathbf{R}_R)^\top \\ &= \mathbf{Q}_L (\mathbf{R}_L \mathbf{R}_R^\top) \mathbf{Q}_R^\top. \end{aligned} \tag{13}$$

Thus, the SVD can be performed on the smaller core matrix $\mathbf{K} = \mathbf{R}_L \mathbf{R}_R^\top$. Let $\mathbf{K}_q$ be its rank-q truncated SVD. The corresponding truncated aggregate and Stage-2 initialization are

$$\begin{aligned} \widehat{\boldsymbol{G}}_{\mathrm{res},q} &:= \mathbf{Q}_L \mathbf{K}_q \mathbf{Q}_R^\top \\ &= \overline{\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}}^\top, \\ \boldsymbol{A}_2^{(0)} &= \big(\overline{\boldsymbol{V}}_{:,1:r_2}\big)^\top, \qquad \boldsymbol{B}_2^{(0)} = \mathbf{0}, \end{aligned} \tag{14}$$

where $r_2 \le q$. This procedure obtains the initialization directions without explicitly forming the dense aggregate $\widehat{\boldsymbol{G}}_{\mathrm{res}}$. The orthogonality in Equation equation 10 applies to each residual gradient before truncation.

A.3 Proof of the Residual-Guided First-Step Update

We prove Theorem 1 for one adapted weight. Write

$$\begin{aligned} \Delta \boldsymbol{W}_2^{(0)} &= s_2 \boldsymbol{B}_2^{(0)} \boldsymbol{A}_2^{(0)} \\ &= s_2 \mathbf{0} \boldsymbol{V}_r^\top \\ &= \mathbf{0}, \end{aligned} \tag{15}$$

so the initial Stage-2 model represents exactly $\boldsymbol{W}_0 + \Delta \boldsymbol{W}_1$.

Let $\boldsymbol{G}^{(0)}$ be the task-loss gradient with respect to the adapted weight at this initial model. The chain rule gives

$$\begin{aligned} \nabla_{\boldsymbol{B}_2} \mathcal{L}_{\mathrm{task}} &= s_2 \boldsymbol{G} \boldsymbol{A}_2^\top, \\ \nabla_{\boldsymbol{A}_2} \mathcal{L}_{\mathrm{task}} &= s_2 \boldsymbol{B}_2^\top \boldsymbol{G}. \end{aligned} \tag{16}$$

Because $\boldsymbol{B}_2^{(0)} = \mathbf{0}$, one simultaneous gradient-descent step with learning rate η gives

$$\begin{aligned} \boldsymbol{A}_2^{(1)} &= \boldsymbol{A}_2^{(0)} - \eta s_2 \big(\boldsymbol{B}_2^{(0)}\big)^\top \boldsymbol{G}^{(0)} = \boldsymbol{V}_r^\top, \\ \boldsymbol{B}_2^{(1)} &= \boldsymbol{B}_2^{(0)} - \eta s_2 \boldsymbol{G}^{(0)} \big(\boldsymbol{A}_2^{(0)}\big)^\top \\ &= -\eta s_2 \boldsymbol{G}^{(0)} \boldsymbol{V}_r. \end{aligned} \tag{17}$$

Therefore,

$$\begin{aligned} \Delta \boldsymbol{W}_2^{(1)} &= s_2 \boldsymbol{B}_2^{(1)} \boldsymbol{A}_2^{(1)} \\ &= s_2 \big(-\eta s_2 \boldsymbol{G}^{(0)} \boldsymbol{V}_r\big) \boldsymbol{V}_r^\top \\ &= -\eta s_2^2 \boldsymbol{G}^{(0)} \boldsymbol{V}_r \boldsymbol{V}_r^\top. \end{aligned} \tag{18}$$

In the special case $\boldsymbol{G}^{(0)} = \widehat{\boldsymbol{G}}_{\mathrm{res},q}$, the SVD definition of $\boldsymbol{V}_r$ further yields

$$\begin{aligned} \Delta \boldsymbol{W}_2^{(1)} &= -\eta s_2^2 \widehat{\boldsymbol{G}}_{\mathrm{res},q} \boldsymbol{V}_r \boldsymbol{V}_r^\top \\ &= -\eta s_2^2 \overline{\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}}^\top \boldsymbol{V}_r \boldsymbol{V}_r^\top \\ &= -\eta s_2^2 \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^\top. \end{aligned} \tag{19}$$

This result describes one plain gradient-descent step. It does not assume that later mini-batch gradients equal the calibration estimate or that subsequent updates remain in the initialized subspace.

A.4 Update Orthogonality Regularization

Let $\boldsymbol{D}_1 = \boldsymbol{B}_1 \boldsymbol{A}_1$ and $\boldsymbol{D}_2 = \boldsymbol{B}_2 \boldsymbol{A}_2$ be the unscaled stage-wise updates. Their Frobenius inner product satisfies

$$\begin{aligned} \langle \boldsymbol{D}_1, \boldsymbol{D}_2 \rangle_F &= \mathrm{tr}(\boldsymbol{D}_1^\top \boldsymbol{D}_2) \\ &= \mathrm{tr}(\boldsymbol{A}_1^\top \boldsymbol{B}_1^\top \boldsymbol{B}_2 \boldsymbol{A}_2) \\ &= \mathrm{tr}\big[(\boldsymbol{B}_1^\top \boldsymbol{B}_2)^\top (\boldsymbol{A}_1 \boldsymbol{A}_2^\top)\big] \\ &= \big\langle \boldsymbol{B}_1^\top \boldsymbol{B}_2,\ \boldsymbol{A}_1 \boldsymbol{A}_2^\top \big\rangle_F. \end{aligned} \tag{20}$$

Similarly,

$$\begin{aligned} \|\boldsymbol{D}_1\|_F^2 &= \mathrm{tr}(\boldsymbol{D}_1^\top \boldsymbol{D}_1) = \big\langle \boldsymbol{B}_1^\top \boldsymbol{B}_1,\ \boldsymbol{A}_1 \boldsymbol{A}_1^\top \big\rangle_F, \\ \|\boldsymbol{D}_2\|_F^2 &= \mathrm{tr}(\boldsymbol{D}_2^\top \boldsymbol{D}_2) = \big\langle \boldsymbol{B}_2^\top \boldsymbol{B}_2,\ \boldsymbol{A}_2 \boldsymbol{A}_2^\top \big\rangle_F. \end{aligned} \tag{21}$$

Substitution into Equation equation 6 gives a factor-only expression:

$$\Omega = \frac{\big\langle \boldsymbol{B}_1^\top \boldsymbol{B}_2,\ \boldsymbol{A}_1 \boldsymbol{A}_2^\top \big\rangle_F^2}{\big\langle \boldsymbol{B}_1^\top \boldsymbol{B}_1,\ \boldsymbol{A}_1 \boldsymbol{A}_1^\top \big\rangle_F \big\langle \boldsymbol{B}_2^\top \boldsymbol{B}_2,\ \boldsymbol{A}_2 \boldsymbol{A}_2^\top \big\rangle_F + \varepsilon}. \tag{22}$$

Thus, the penalty can be evaluated without explicitly forming either dense update matrix. It helps maintain orthogonality between $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ during Stage-2 training; it does not require their left or right singular subspaces to be separately orthogonal.

B Supplementary Implementation Details

This section specifies the model setup, residual-gradient calibration, training hyperparameters, algorithm flow, and computational cost of ReGRA.

B.1 Model and Training Setup

We apply LoRA adapters to the query, key, and value projections. The pretrained backbone remains fixed during both training stages. Each ReGRA stage uses rank 8, LoRA scaling parameter $\alpha = 16$, and dropout 0.05. Stage 1 trains a standard LoRA adapter on the task training data. Its learned update is then fixed during calibration and Stage-2 training.

For GLUE, the task-specific classification head is trained in Stage 1 and retained without further updates in Stage 2. For generative tasks, the language-modeling loss is computed only on response tokens. Experiments use seed 42, bfloat16 precision, and zero weight decay. The training runs were conducted on an NVIDIA GeForce RTX 3090.

Table 4: Training settings for ReGRA. The Stage-2 learning rate for RoBERTa-large RTE is 10^{-4} rather than 5×10^{-5}.

Setting	Epochs	Stage-1 LR	Stage-2 LR	Batch size	Max. length
RoBERTa-base, GLUE	Task-specific	2×10^{-4}	2×10^{-4}	32	128
RoBERTa-large, GLUE	Task-specific	2×10^{-4}	$5 \times 10^{-5*}$	32	128
Llama-3-8B, math	2	10^{-4}	10^{-4}	2	256
Qwen3-8B-Base, math	2	10^{-4}	10^{-5}	2	256
Llama-3-8B, code	1	2×10^{-5}	5×10^{-6}	4	1024
Llama-3-8B, commonsense	1	10^{-4}	2×10^{-5}	4	256

*Except RTE, which uses 10^{-4}.

Table 5: Residual-gradient calibration settings for ReGRA.

Setting	M	b	q
GLUE, both backbones	8	32	16
Llama-3-8B, math	1	1	8
Qwen3-8B-Base, math	4	2	8
Llama-3-8B, code	16	1	16
Llama-3-8B, commonsense	16	4	32

B.2 Residual-Gradient Calibration

Calibration is performed once after Stage 1 and before Stage 2. We load the pretrained backbone together with the fixed Stage-1 adapter and evaluate task gradients on batches drawn from the training data. The selected calibration records are processed without data-loader shuffling. For each adapted weight, a gradient with respect to the underlying weight is computed without applying an optimizer update.

Each gradient is projected using the first-stage left and right subspaces, as described in Appendix A.1. The residual is immediately compressed to rank at most q, and the dense gradient is discarded. Batchwise low-rank factors are aggregated using the procedure in Appendix A.2. The leading right singular vectors of the aggregate initialize $\boldsymbol{A}_2$, while $\boldsymbol{B}_2$ is initialized to zero. Consequently, the initialized Stage-2 adapter does not change the function represented by the Stage-1 model.

For commonsense reasoning, the calibration subset contains 64 examples, with eight selected from each of the eight tasks. This task balancing applies only to calibration; it does not change the order of subsequent Stage-2 training.

B.3 Hyperparameter Settings

Both stages use fused AdamW with a linear learning-rate schedule and 100 optimizer-step learning-rate warmup. The effective training batch sizes, maximum sequence lengths, epoch counts, and peak learning rates are listed in Table 4. Both stages use the same epoch count within a given task.

For both RoBERTa backbones, CoLA, MNLI, MRPC, QNLI, QQP, RTE, SST-2, and STS-B are trained for 10, 3, 20, 3, 3, 20, 5, and 10 epochs, respectively.

Table 5 gives the calibration settings. Here, M is the number of calibration batches, b is their batch size, and q is the gradient compression rank. The low-rank SVD routine uses four iterations in all listed configurations.

B.4 Stage-2 Optimization and Merging

During Stage 2, the backbone and Stage-1 adapter remain fixed. Only the Stage-2 LoRA factors are optimized. Training uses ordinary task gradients; the residual-gradient projection is used for

Algorithm 1 ReGRA: two-stage residual-gradient adaptation

Require: Pretrained weights $\{\boldsymbol{W}_{0,\ell}\}_{\ell=1}^{L}$; training data $\mathcal{D}_{\text{train}}$; calibration batches $\{\mathcal{C}_b\}_{b=1}^{M}$; stage ranks r_1, r_2; compression rank $q \geq r_2$
Ensure: Adapted weights $\{\boldsymbol{W}_\ell^*\}_{\ell=1}^{L}$
Train a rank-r_1 Stage-1 LoRA adapter on $\mathcal{D}_{\text{train}}$
Fix its updates $\{\Delta\boldsymbol{W}_{1,\ell}\}_{\ell=1}^{L}$
for each adapted weight ℓ **do**
    Obtain the first-stage subspace projectors $\boldsymbol{P}_{U,\ell}$ and $\boldsymbol{P}_{V,\ell}$
    for each calibration batch $\mathcal{C}_b$ **do**
        Compute the task gradient $\boldsymbol{G}_{\ell,b}$ at $\boldsymbol{W}_0 + \Delta\boldsymbol{W}_1$
        Form $\boldsymbol{G}_{\text{res},\ell,b} \leftarrow \boldsymbol{G}_{\ell,b} - \boldsymbol{P}_{U,\ell}\boldsymbol{G}_{\ell,b}\boldsymbol{P}_{V,\ell}$
        Store its rank-q compressed factors and discard the dense gradient
    end for
    Aggregate the compressed factors and extract the leading r_2 right singular vectors $\boldsymbol{V}_{\ell,r}$
    Initialize $\boldsymbol{A}_{2,\ell} \leftarrow \boldsymbol{V}_{\ell,r}^\top$, $\boldsymbol{B}_{2,\ell} \leftarrow \boldsymbol{0}$
end for
Fix Stage 1 and optimize only the Stage-2 factors using the task loss and orthogonality regularizer
for each adapted weight ℓ **do**
    Merge $\boldsymbol{W}_\ell^* \leftarrow \boldsymbol{W}_{0,\ell} + \Delta\boldsymbol{W}_{1,\ell} + \Delta\boldsymbol{W}_{2,\ell}$
end for
return $\{\boldsymbol{W}_\ell^*\}_{\ell=1}^{L}$

initialization rather than applied to every training step. The task objective is augmented with the update-level orthogonality regularizer in Equation equation 6.

We linearly warm up the regularization coefficient over the first 100 Stage-2 optimization steps:

$$\lambda_t = 0.1 \min\left\{1, \frac{t}{100}\right\}, \tag{23}$$

where t denotes the Stage-2 optimization step. This regularizer warmup is separate from the learning-rate warmup.

After training, the effective updates from both stages are merged into each adapted pretrained weight:

$$\boldsymbol{W}_{\text{merged}} = \boldsymbol{W}_0 + \Delta\boldsymbol{W}_1 + \Delta\boldsymbol{W}_2. \tag{24}$$

The merged model requires no additional adapter branch or adapter-specific computation during inference.

B.5 Algorithm Flow

Algorithm 1 summarizes the complete procedure. Stage 1 supplies a fixed task-specific update. Calibration constructs a low-rank estimate of the residual gradient at that solution. Stage 2 is initialized from this estimate, optimized with update-orthogonality regularization, and finally merged with Stage 1.

B.6 Computational Cost

For an adapted weight of size $d_{\text{out}} \times d_{\text{in}}$, each dense calibration gradient is held only transiently. Storing rank-q factors for M batches requires $O(Mq(d_{\text{out}} + d_{\text{in}}))$ numbers per adapted weight. Factorized aggregation uses QR factorizations and an SVD of a core matrix of size at most $Mq \times Mq$; it does not explicitly form the dense aggregate gradient. Calibration is performed only once between the training stages.

The orthogonality penalty is computed from LoRA factors rather than dense update matrices. Its per-weight cost is

$$O\big((d_{\text{out}} + d_{\text{in}})(r_1^2 + r_2^2 + r_1 r_2)\big). \tag{25}$$

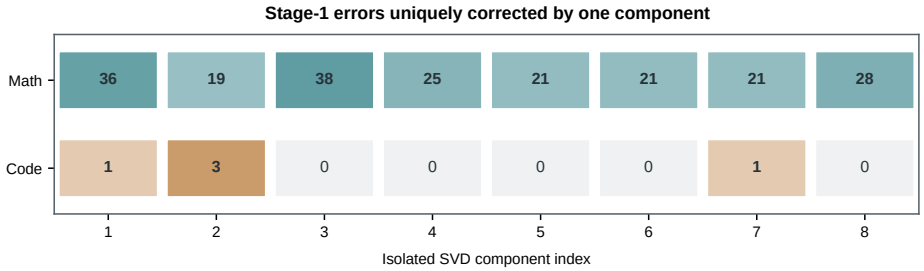


Figure 8: **Stage 1 errors uniquely corrected by isolated Stage-2 components.** Each cell counts cases answered incorrectly by Stage 1, corrected by the indicated component, and not corrected by any of the other seven isolated components. Math counts task–example pairs across six benchmarks; Code counts MBPP+ problems.

Across the two stages, the LoRA factors contain $(r_1 + r_2)(d_{\text{out}} + d_{\text{in}})$ parameters per adapted weight, excluding any task-specific output head. Once both updates are merged, inference uses the same dense weight multiplication as the pretrained model.

C Additional Results

C.1 Commonsense Reasoning

Table 6: **Accuracy (%) on eight commonsense reasoning benchmarks** using Llama-3-8B. Avg. is the unweighted mean across tasks. Subscript values beside ReGRA scores show percentage-point differences from rank-8 LoRA. The best average is marked in bold.

	Method	OBQA	ARC-C	Wino.	PIQA	SIQA	ARC-E	BoolQ	Hella.	Avg.
	LoRA ($r = 16$)	88.40	81.10	87.29	88.74	81.63	93.14	74.19	93.35	85.98
Llama-3-8B	LoRA ($r = 8$)	86.80	81.31	87.61	88.57	81.12	93.10	69.93	92.73	85.15
	DoRA ($r = 8$)	86.40	81.14	87.21	89.28	81.57	93.43	73.82	92.57	85.68
	PiSSA ($r = 8$)	81.00	77.90	86.02	88.46	80.80	92.38	73.18	91.55	83.91
	LoRA-One ($r = 8$)	88.20	81.05	87.37	88.84	81.67	93.22	74.31	92.81	85.93
	ReGRA ($r = 8$)	$88.20_{+1.40}$	$82.00_{+0.69}$	$86.90_{-0.71}$	$89.34_{+0.77}$	$81.93_{+0.81}$	$92.30_{-0.80}$	$74.22_{+4.29}$	$93.07_{+0.34}$	**86.00**$_{+0.85}$

Table 6 shows that ReGRA achieves the highest eight-task average on Llama-3-8B, improving upon rank-8 LoRA by 0.85 percentage points. The largest task-level gains occur on BoolQ and OBQA, while Wino. and ARC-E decline. Increasing standard LoRA to rank 16 also improves the average and nearly matches ReGRA (85.98% versus 86.00%). These results extend the average gain over rank-8 LoRA to commonsense reasoning.

C.2 Error-Level Analysis of Stage-2 Components

C.2.1 Unique Error Corrections

Figure 8 compares the Stage-1 errors corrected by each isolated Stage-2 component. On Math, all eight components uniquely corrected errors, with 19–38 task–example pairs per component. On MBPP+, only components 1, 2, and 7 had unique corrections, with 1, 3, and 1 problems, respectively. The isolated components therefore showed distinct correction patterns on Math, while this distinction was less evident on MBPP+.

C.2.2 Overlap with Full-Model Gains

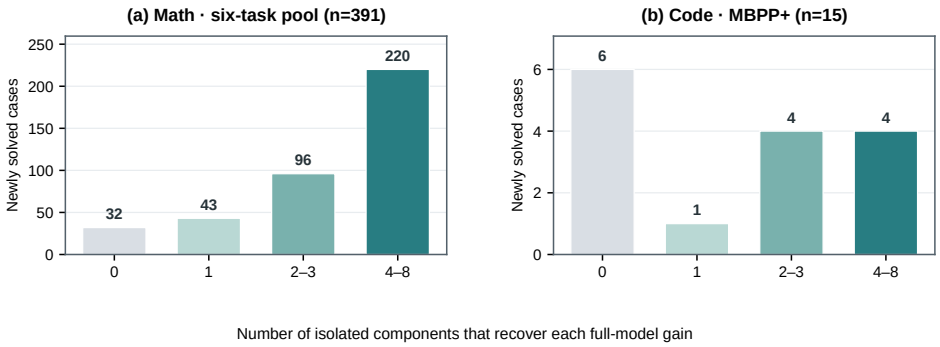


Figure 9: **Coverage of the full Stage-2 model's newly solved cases.** For each case answered incorrectly by Stage 1 and correctly by the full Stage-2 model, the bars show how many isolated components also answer it correctly. Math includes 391 task–example pairs; Code includes 15 MBPP+ problems.

Figure 9 examines cases newly solved by the full Stage-2 model. At least one isolated component also solved 359 of 391 Math cases and 9 of 15 MBPP+ problems. The remaining 32 Math cases and 6 MBPP+ problems were not solved by any isolated component. These results show that isolated components recover many full-model gains, while some gains appear only with the combined update. The analysis does not measure each component's marginal contribution within that update.