MPO-PRO: TRAJECTORY SHAPING THROUGH TEMPORARY MATRIX FACTORIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Post-training adapts pretrained language models to tasks such as mathematical reasoning and code generation, but local deployment often limits model size and inference cost. One way to improve post-training is to use overparameterization during training, allowing a richer parameterization than the final deployed model. We propose MPO-PRO, an overparameterization-based approach that enriches post-training by using matrix product operators (MPOs) as temporary training coordinates for dense updates and LoRA factors, and contracts them before deployment. This keeps the original model architecture or adapter rank unchanged while providing a richer training parameterization. For the fixed-rank branch, we use the first-order LoRA-Pro update as a reference and approximately match it in the temporary MPO representation.On GSM8K and HumanEval, the dense and fixed-rank branches outperform full fine-tuning and LoRA-Pro, respectively. When trained on SciInstruct, MPO-PRO also outperforms LoRA-Pro on SciBench Physics. We further analyze the update mechanism through decomposition and find that the performance gain is mainly associated with the first-order implementation deviation introduced by the temporary representation relative to LoRA-Pro.

1 INTRODUCTION

As AI agents are increasingly deployed in specialized domains and on local or resource-constrained devices, post-training has become a central stage in adapting pretrained language models for reasoning, code generation, and domain-specific tasks. Unlike inference, which is performed repeatedly throughout deployment, post-training is typically performed only once. This distinction matters especially under fixed deployment budgets: only what remains in the final model affects inference cost, while temporary training-time structure can be discarded before deployment. This is especially relevant for local and on-device models, where inference resources are constrained. This separation suggests that training can use more capacity than deployment, as long as the extra structure is removed before inference.

Temporary overparameterization separates the training representation from the deployed model. Overparameterized fine-tuning (OPF) factorizes selected pretrained weights into temporary MPO cores (Gao et al., 2023). LoRA-Over expands selected adapter factors after preliminary LoRA training and importance selection (Gao et al., 2026). Both methods use removable training structures to improve adaptation. We study how these temporary factors affect optimization. With the final matrix shape and rank bound fixed, how does a richer training representation change the update trajectory and the final performance?

Exactly factorizing a matrix into an overparameterized set of factors does not change the set of matrices that can ultimately be deployed. The key difference from direct optimization lies in the optimization path. The factorization changes how each optimizer step is realized in matrix space. An update to one factor is mediated by the current values of the others, and the factor-to-matrix mapping changes as the factors evolve. As a result, two training procedures can end with matrices of exactly the same shape, parameter count, and rank bound, yet reach them along different optimization trajectories. Temporary overparameterization is useful only if these different trajectories lead to better deployable solutions.

We propose MPO-PRO, a post-training framework that uses matrix product operators (MPOs) in a lift–train–contract procedure (Pirvu et al., 2010). We train temporary factors and contract them before

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.

export. For compact models, OPF factorizes pretrained weights directly. Our dense branch instead freezes these weights and learns a scaled additive update through temporary MPO cores. For larger models, we retain the standard LoRA form (Hu et al., 2022) and factorize one designated LoRA factor from the start. This removes the preliminary LoRA training and importance selection used by LoRA-Over. The dense branch exports the original architecture, and the LoRA branch exports an adapter of the original rank. Section 3 describes the training procedure.

Our experiments show that MPO-PRO improves performance while preserving the deployment format. On Qwen3.5-0.8B, the dense branch outperforms full fine-tuning on GSM8K and HumanEval. On Llama-2-7B (Touvron et al., 2023), the fixed-rank branch outperforms reproduced LoRA-Pro at ranks 8 and 16 with the same adapter size; it also saves one preliminary LoRA training epoch and achieves higher mean scores than LoRA-Over on GSM8K, HumanEval, and MT-Bench (Table 2; Appendix B.1). On SciBench Physics, rank-8 adapters trained on SciInstruct-70K improve over matched LoRA-Pro from $0.88 \pm 0.76\%$ to $2.20 \pm 0.76\%$.

LoRA-Pro corrects low-rank factor gradients to better approximate the full fine-tuning update (Wang et al., 2025). We use its first-order direction as a reference and approximately match it within the temporary MPO representation. We then study how the evolving core-to-matrix map changes the training trajectory. Each contracted step is decomposed into the target P, a first-order implementation deviation E, and a higher-order remainder H. Across three seeds, retaining $P + E$ preserves the full update's GSM8K gain, whereas retaining $P + H$ loses this advantage, despite nearly identical validation losses. This indicates that the gain is primarily associated with the first-order implementation deviation E, rather than the higher-order remainder H. In other words, the temporary representation changes how an otherwise nearly matched first-order update is realized and accumulated over finite optimization steps.

Our contributions are:

- We introduce a deployment-equivalent post-training framework that uses richer temporary training representations while preserving the final dense architecture or LoRA rank.
- We show that temporary overparameterization improves mathematical reasoning, code generation, and scientific reasoning without changing the deployment format, including a 150% relative gain over matched LoRA-Pro on SciBench Physics.
- We identify a trajectory-level mechanism: even with nearly matched first-order updates, the temporary representation changes the training path, and a three-seed decomposition shows that this first-order deviation, rather than higher-order terms, preserves the GSM8K gain.

2 RELATED WORK

Low-rank adaptation. LoRA learns mergeable low-rank updates (Hu et al., 2022), while QLoRA reduces training memory with quantized backbones (Dettmers et al., 2023). Extensions improve scaling and learning rates (Kalajdzievski, 2023; Hayou et al., 2024), share bases or allocate rank (Kopiczko et al., 2024; Zhang et al., 2023), and refine initialization or separate weight direction from magnitude (Meng et al., 2024; Wang et al., 2024; Liu et al., 2024). Riemannian preconditioning and LoRA-RITE address sensitivity to factor transformations (Zhang and Pilanci, 2024; Yen et al., 2025). We study temporary training factors while preserving the exported adapter shape and rank bound.

Factorized optimization and MPO applications. Deep linear networks change the effective gradient through their factors (Arora et al., 2018), and deep matrix factorization affects implicit regularization (Arora et al., 2019; Yaras et al., 2024; Castin et al., 2026). OPF uses temporary MPO cores to fine-tune pretrained weights (Gao et al., 2023). LoRA-Over applies removable factorizations to low-rank adapters (Gao et al., 2026). OP-LoRA generates adapter parameters with a temporary network (Teterwak et al., 2024). MPOs also support neural-network compression (Gao et al., 2020) and weight-decomposed initialization in DoTA (Hu et al., 2025). We retain full bond dimensions and contract the cores before deployment, using the MPO to study how first-order matching and state-dependent core-to-matrix maps shape matrix updates.

3 METHOD

MPO-PRO represents a dense weight update or one LoRA factor with temporary cores, trains these cores, and combines them into an ordinary matrix for deployment (Figure 1). We introduce the representation, derive its local update, and explain how small update differences can accumulate along the training trajectory.

3.1 LORA AND MPO PRELIMINARIES

Let $W_0 \in \mathbb{R}^{m\times n}$ be a frozen pretrained weight, where m and n are the output and input widths. LoRA (Hu et al., 2022) forms the effective weight W as

$$W = W_0 + sBA, \qquad B \in \mathbb{R}^{m\times r}, \quad A \in \mathbb{R}^{r\times n}. \tag{1}$$

Here A and B are trainable, r bounds the update rank, and $s > 0$ is fixed. An MPO represents M as a contraction of p multidimensional cores:

$$M = \mathcal{D}(C_1, \ldots, C_p) = \mathcal{D}(\theta). \tag{2}$$

Each core has local indices and shared indices with its neighbors; contraction sums products over shared indices and is multilinear in each core. We retain full bond dimensions and initialize by untruncated SVD, so the MPO exactly represents the chosen matrix shape at initialization. The temporary cores may use more parameters than the matrix, while r still bounds the exported LoRA rank. Appendix A.2 gives the index formulas and SVD construction (Pirvu et al., 2010; Oseledets, 2011). Superscripts 1 and 2 distinguish dense and LoRA cores.

3.2 DENSE ADAPTATION THROUGH FIVE TEMPORARY CORES

For compact models, we learn a dense additive update U through five cores and a trainable scalar γ:

$$W = W_0 + U, \qquad U = \gamma\mathcal{D}(\theta), \qquad \mathcal{D}(\theta_0) = W_0, \quad \gamma_0 = 10^{-8}. \tag{3}$$

Here θ contains the entries of $C_1^1, \ldots, C_5^1$, and the subscript 0 denotes initialization. The cores start from the pretrained weight, while the small scale keeps the added update close to zero. Adam updates both the cores and γ through the reconstructed weight.

To see how a core step changes the matrix, let vec stack matrix entries into a vector in a fixed order. Define the vectorized contraction a, vectorized update u, and Jacobian J by

$$a = \mathrm{vec}(\mathcal{D}(\theta)), \qquad u = \mathrm{vec}(U) = \gamma a, \qquad J = \frac{\partial a}{\partial \theta}.$$

The Jacobian has one row per matrix entry and one column per core parameter. It maps a small core increment $\delta\theta$ to the first-order matrix change $J\delta\theta$. If $\delta\gamma$ is the simultaneous scalar increment, the first-order change in u, denoted by Δu_1, is

$$\Delta u_1 = a\,\delta\gamma + \gamma J\delta\theta. \tag{4}$$

The first term changes the update's scale; the second changes its matrix entries through the cores. The subscript 1 denotes terms linear in the increments; the exact finite-step change also includes their products.

For intuition, let $\mathcal{L}$ be the training loss, $d = \nabla_u\mathcal{L}$ its gradient with respect to u, and $\eta > 0$ a learning rate. The chain rule gives scalar and core gradients $a^\top d$ and $\gamma J^\top d$, where $\top$ denotes transpose. Substituting their gradient-descent steps into Equation 4 yields $\Delta u_1 = -\eta(aa^\top + \gamma^2 JJ^\top)d$. Thus factorization transforms the matrix gradient by a map that changes with the current cores and scale. Adam uses its own parameter increments in the same first-order map. Appendix A.3 gives the expanded derivation. After the final training step T, export merges $\gamma_T\mathcal{D}(\theta_T)$ into W_0 to produce an ordinary dense weight.

3.3 MATCHING THE LORA-PRO DIRECTION THROUGH CORE UPDATES

For larger models, we retain Equation 1 and factorize only A into three cores:

$$A = \mathcal{D}(C_1^2, C_2^2, C_3^2) = \mathcal{D}(\theta_A). \tag{5}$$

Here θ_A contains the three cores' entries. Their initial contraction equals the baseline's initial A, while B remains an ordinary matrix. We use LoRA-Pro (Wang et al., 2025) as the reference optimizer. It corrects the two factor directions so that their combined first-order weight change better approximates a full-matrix direction. Denote its current update directions by $D_A \in \mathbb{R}^{r\times n}$ and $D_B \in \mathbb{R}^{m\times r}$; its factor increments are $-\eta D_A$ and $-\eta D_B$.

We keep the reference B step and seek core increments that reproduce the reference A step to first order. Let $J_A = \partial \operatorname{vec}(A)/\partial\theta_A$ be the Jacobian of this contraction, and let $d_A = \operatorname{vec}(D_A)$ be the vectorized target direction. Simply mapping d_A to the cores with $J_A^\top$ and back with J_A would give $J_A J_A^\top d_A$, changing the intended direction. We instead choose a core direction h, with one entry per core parameter, to minimize this mismatch. The resulting core increment is $\delta\theta_A$:

$$h_\lambda = \arg\min_h \left\{ \frac{1}{2}\|J_A h - d_A\|_2^2 + \frac{\lambda}{2}\|h\|_2^2 \right\}, \qquad \delta\theta_A = -\eta h_\lambda. \tag{6}$$

Here $\|\cdot\|_2$ is the Euclidean vector norm, $\lambda > 0$ controls damping, and h_λ is the minimizer. The two terms respectively match the matrix direction and limit large core steps.

We solve this regularized least-squares problem in the space of A entries. Define the Gram matrix $K_A = J_A J_A^\top \in \mathbb{R}^{rn\times rn}$, an auxiliary solution vector $z \in \mathbb{R}^{rn}$, and the identity matrix $I \in \mathbb{R}^{rn\times rn}$. The equivalent solve is

$$(K_A + \lambda I)z = d_A, \qquad h_\lambda = J_A^\top z. \tag{7}$$

The realized first-order direction is then $J_A h_\lambda = K_A z$. We call this procedure *Gram-CG*: conjugate gradients solves the system using products with K_A, each evaluated by mapping a vector to the cores and back through contractions. Neither J_A nor K_A needs to be stored explicitly. Appendix A.4 derives the equivalence and contraction operations; Appendix A.6 gives the numerical settings.

Damping and an inexact solve make the match approximate. If J_A has full row rank, meaning every infinitesimal change in A can be produced by a core change, an exact solve recovers the target as damping tends to zero. We measure the realized first-order agreement in Section 4.3.

3.4 From Local Matching to the Training Trajectory

Close local matching still leaves discrepancies from damping and numerical errors. Updating several cores together also introduces products of their increments. We separate these effects at a single training step.

To separate these effects, let ΔA be the exact change after the applied core increment $\delta\theta_A$, and define its first-order part L by $\operatorname{vec}(L) = J_A \delta\theta_A$. Let P be the reference LoRA-Pro increment, E its first-order implementation deviation, and H the second- and higher-order remainder:

$$\begin{aligned} \Delta A &= \mathcal{D}(\theta_A + \delta\theta_A) - \mathcal{D}(\theta_A) = P + E + H, \\ P &= -\eta D_A, \qquad E = L - P, \qquad H = \Delta A - L. \end{aligned} \tag{8}$$

Each of $\Delta A, L, P, E, H$ is an $r \times n$ matrix. Appendix A.5 expands L and H using multilinearity.

These deviations change the next contracted state, at which both the loss gradient and J_A are evaluated again. Close local agreement can therefore coexist with different training trajectories. Section 4.3 tests which deviations preserve the downstream gain by retaining selected terms in this decomposition. After step T, the final contraction A_T and final matrix B_T form the exported adapter, with $r(m+n)$ entries and update rank at most r. No MPO core or Gram-CG operation remains in the inference graph.

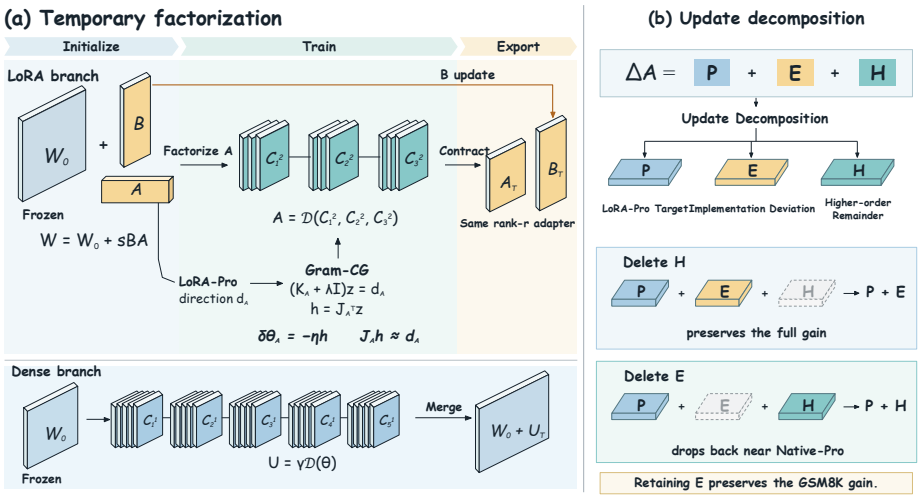


Figure 1: **Temporary factorization changes the update while preserving the deployment format.** (a) Gram-CG updates three cores representing LoRA's A, which contracts back into a standard rank-r adapter. The dense variant trains five cores and merges their scaled contraction into W_0. (b) Update decomposition $\Delta A = P + E + H$, where P is the LoRA-Pro target, E is the first-order implementation deviation, and H contains higher-order terms. Dashed components indicate deletion: retaining $P + E$ preserves the GSM8K gain, whereas retaining $P + H$ returns performance near the reference branch(Native pro) (Section 4.3).

4 Experiments

We evaluate dense and fixed-rank adaptation under matched deployment formats, then analyze local update agreement, the components that shape the trajectory, and the spectra of the exported matrices.

Training details. We train separate models for mathematical reasoning and code generation on MetaMathQA-100K and CodeFeedback-100K, respectively, with held-out validation splits. We evaluate reasoning on GSM8K (Cobbe et al., 2021) and code generation on HumanEval (Chen et al., 2021). HumanEval pass@k estimates the probability that at least one of k sampled programs passes the tests. We also report Llama-2-7B first-turn dialogue scores on MT-Bench (Zheng et al., 2023).

In the dense study, all methods train Qwen3.5-0.8B for one epoch with Adam and zero weight decay. Every method exports the same dense architecture. Learning rates, training parameter counts, and recorded costs are listed in Table 1. In the fixed-rank study, MPO-Pro and reproduced LoRA-Pro use the same rank, target modules, initialization, forward scaling, one-epoch SGD schedule, and effective batch size. At rank 8, both export 19.99M adapter parameters, while MPO-Pro trains 20.45M parameters before contraction. We evaluate both ranks 8 and 16. We additionally train rank-8 LoRA, LoRA-Over, LoRA-Pro, and MPO-Pro on SciInstruct-70K (Zhang et al., 2024) from the official SciGLM repository, with three seeds each and evaluate the resulting models on SciBench Physics.

Unless stated otherwise, main comparisons report means and sample standard deviations over three runs. GSM8K uses greedy decoding on all 1,319 questions, and HumanEval uses 200 samples per task on all 164 tasks, with shared decoding settings across methods. Appendix B and Appendix F provide the seeds and full training and evaluation settings.

4.1 Dense Factorization Improves Math and Code Performance

We compare five-core MPO-Pro with full fine-tuning and a Scrambled MPO control. Scrambled MPO changes the index grouping through fixed row and column permutations while retaining initialization from W_0 and the same trainable parameter count as MPO-Pro.

Table 1: **Dense adaptation on Qwen3.5-0.8B.** Values are three-seed means ± sample standard deviations. Every row exports the same dense model. HE p@1 denotes HumanEval pass@1, and LR is the learning rate. Full-FT uses 10^{-5}, the best tested learning rate; Appendix F reports the learning-rate comparison. Cost sums GPU-hours across devices over three math and three code runs. Full fine-tuning at 10^{-5} and MPO-PRO use two H100 GPUs per run; Scrambled MPO uses one H100 GPU. Bold and underline mark the best and second-best benchmark means.

Training method	GSM8K ↑	HE p@1 ↑	LR	Trainable params	Cost (GPU-h)
Full-FT	61.031±1.070	22.860±0.618	10^{-5}	0.8B	27.79
Scrambled MPO	**63.937±1.292**	<u>25.167±0.472</u>	10^{-4}	2.016B	67.77
MPO-PRO	<u>63.811±0.662</u>	**25.484±0.313**	10^{-4}	2.016B	33.30

Table 2: Llama-2-7B fixed-rank adaptation. The upper block gives published results from Wang et al. (2025) under their original protocol. Lower blocks report our three-run means ± sample standard deviations. Difference is MPO-PRO minus reproduced LoRA-Pro; MT-Bench differences are paired by seed. HE p@k denotes HumanEval pass@k. GSM8K, HE, and SciBench scores are percentages; SciBench uses rank-8 models trained on SciInstruct-70K. Bold marks the highest mean within each rank; — denotes an unreported value.

Method	GSM8K ↑	HE p@1 ↑	HE p@10 ↑	HE p@100 ↑	MT-Bench ↑	SciBench Physics (%) ↑
Full FT	59.36±0.85	35.31±2.13	—	—	5.30±0.11	—
LoRA	42.08±0.04	14.76±0.17	—	—	5.61±0.10	—
PiSSA	44.54±0.27	16.02±0.78	—	—	5.30±0.02	—
DoRA	53.07±0.75	19.75±0.41	—	—	5.97±0.02	—
LoRA-GA	53.60±0.30	19.81±1.46	—	—	5.95±0.16	—
LoRA-Pro	57.57±0.50	22.97±0.35	—	—	5.72±0.03	—
Matched rank $r = 8$						
LoRA (reproduced)	45.818±0.662	14.809±0.086	35.811±0.294	60.498±1.434	—	1.32±0.00
LoRA-Pro (reproduced)	60.804±0.602	20.242±0.230	47.443±0.317	72.071±1.030	**6.1833±0.0686**	0.88±0.76
LoRA-Over (reproduced)	57.70±1.37	19.31±0.46	45.80±0.99	72.41±1.89	5.92±0.07	1.03±0.51
MPO-PRO (ours)	**61.612±0.232**	**21.588±0.504**	**49.518±0.225**	**73.371±0.501**	6.0521±0.1092	**2.20±0.76**
Difference	**+0.809**	**+1.347**	**+2.075**	**+1.300**	−0.1313±0.0410	**+1.32**
Matched rank $r = 16$						
LoRA (reproduced)	49.052±0.303	15.173±0.302	37.137±0.656	61.509±0.482	—	—
LoRA-Pro (reproduced)	61.789±0.152	21.590±0.173	49.811±0.646	73.955±2.661	**6.2287±0.0731**	—
LoRA-Over (reproduced)	58.00±1.44	19.88±0.10	47.62±0.85	72.88±1.01	5.92±0.07	—
MPO-PRO (ours)	**63.002±0.661**	**21.920±0.701**	**51.021±0.107**	**74.116±0.330**	6.1449±0.0968	—
Difference	**+1.213**	**+0.329**	**+1.210**	**+0.161**	−0.0838±0.0396	—

Table 1 shows gains from both MPO parameterizations over full fine-tuning on both benchmarks. Relative to full fine-tuning, MPO-PRO raises GSM8K accuracy from 61.03% to 63.81% and HumanEval pass@1 from 22.86% to 25.48%. Scrambled MPO leads on GSM8K, while MPO-PRO leads on HumanEval. The gain under scrambling suggests that the observed improvement extends beyond the default MPO index grouping.

These gains use additional training resources. The recorded cost is 33.30 cumulative H100 GPU-hours across the six math and code runs for MPO-PRO, compared with 27.79 for full fine-tuning at 10^{-5}. Both use two H100 GPUs per run, giving a 1.20× GPU-hour ratio. Wall times for the different execution protocols are reported in Appendix F; the exported architecture remains unchanged.

4.2 FIXED-RANK ADAPTERS IMPROVE ON MATH AND CODE BENCHMARKS

We next compare MPO-PRO with reproduced LoRA-Pro at the same adapter rank. The lower blocks of Table 2 give these matched comparisons. The upper block provides published context and is not directly comparable, because those methods were evaluated under their original protocols. On SciBench Physics, MPO-PRO reaches 2.20 ± 0.76%, compared with 0.88 ± 0.76% for matched LoRA-Pro, a 1.32-point absolute improvement.

At rank 8, MPO-PRO improves GSM8K accuracy by 0.809 percentage points and HumanEval pass@1 by 1.347 points over reproduced LoRA-Pro, with gains also at pass@10 and pass@100. At rank 16, MPO-PRO remains higher than reproduced LoRA-Pro on GSM8K and all three HumanEval

Table 3: Recorded training wall time for the matched Llama-2-7B comparisons. Values are three-run means ± sample standard deviations; every run uses two H100 GPUs.

Rank	Method	Math wall time (h)	Code wall time (h)
8	LoRA-Pro	1.249±0.004	2.202±0.029
8	MPO-PRO	2.330±0.157	3.241±0.246
16	LoRA-Pro	1.908±0.371	3.085±0.546
16	MPO-PRO	3.940±1.031	4.600±0.330

Table 4: **Three-seed update-decomposition experiment on Llama-2-7B (rank 8).** Each branch retains the terms shown in Equation 8; all other training and evaluation settings are matched.

Branch (retained terms)	Final validation loss ↓	GSM8K (%) ↑
Native-Pro (P)	0.1185354 ± 0.0001840	60.854 ± 0.191
Full ($P + E + H$)	0.1185443 ± 0.0002630	61.638 ± 0.227
No-H ($P + E$)	0.1185119 ± 0.0003270	$\mathbf{61.815 \pm 0.191}$
No-E ($P + H$)	$\mathbf{0.1183806 \pm 0.0001520}$	60.677 ± 0.191

metrics, although the code gains are smaller. Compared with reproduced LoRA-Over, MPO-PRO leads on all four metrics at both ranks, with GSM8K gains of 3.91 and 5.00 points, respectively. Held-out loss is also lower than LoRA-Pro on both tasks at rank 8 (Appendix C).

On dialogue generation, MT-Bench favors LoRA-Pro at both ranks, with paired differences of -0.1313 ± 0.0410 at rank 8 and -0.0838 ± 0.0396 at rank 16. The matched gains in this study are therefore concentrated in math and code; a five-task language-understanding comparison on T5-Base is reported in Appendix E.

Recorded training wall times are shown in Table 3.

4.3 TRAJECTORY-SENSITIVE DIAGNOSTICS UNDER A MATCHED BF16 PROTOCOL

The reconstructed first-order updates closely match the LoRA-Pro targets. Across the recorded rank-8 math and code runs, mean cosine similarity exceeds 0.9999997, measuring direction agreement. The realized-to-target norm ratio is 0.99988, measuring magnitude agreement. Mean relative pullback error, the norm of the first-order matrix mismatch divided by the target norm, is below 7.7×10^{-4}.

We retain P, $P + E + H$, $P + E$, or $P + H$ at every step in four otherwise matched branches. Here P is the LoRA-Pro reference increment, E is the first-order implementation deviation, and H is the higher-order remainder defined in Equation 8.

All four branches use bfloat16 (BF16), a 16-bit floating-point format, for core and base-weight storage. They share Llama-2-7B, rank 8, 3,125 optimizer steps, data order, and greedy evaluation on all 1,319 GSM8K questions. Table 4 reports three-seed means and sample standard deviations.

Retaining $P + E$ preserves the full-update advantage (61.815% versus 61.638%), whereas retaining $P + H$ returns performance near Native-Pro (60.677% versus 60.854%). The mean validation losses differ by just 1.64×10^{-4}, with No-E attaining the lowest loss. Thus retaining E preserves the GSM8K gain even without H in this matched rank-8 experiment, while endpoint validation loss does not track the performance ordering.

4.4 ABLATIONS OF DAMPING AND DIRECTION CHOICE

Table 5 varies the damping coefficient μ, which scales the regularization strength λ in Equation 7 (Appendix A.4). It also replaces the filtered target with an equal-norm isotropic direction to probe direction choice at fixed magnitude. These seed-0 runs use the same temporary representation and training schedule.

Zero damping gives the highest GSM8K score in this seed-0 comparison (61.9409%). The default $\mu = 10^{-4}$ attains the lowest validation loss but scores 1.3647 percentage points lower on GSM8K, whose dependence on μ is non-monotonic. The equal-norm isotropic direction raises accuracy from

Table 5: **Damping and direction ablations on Llama-2-7B (rank 8, math, seed 0).** All rows use 3,125 steps and greedy GSM8K evaluation on 1,319 questions.

Method	μ	Final validation loss ↓	GSM8K (%) ↑
LoRA-Pro (reference)	–	0.122373313	60.3487
MPO-PRO ($\mu = 0$)	0	0.118229359	61.9409
MPO-PRO ($\mu = 10^{-5}$)	10^{-5}	0.118211716	60.6520
MPO-PRO (default)	10^{-4}	**0.118171923**	60.5762
MPO-PRO ($\mu = 10^{-3}$)	10^{-3}	0.118272133	61.0311
MPO-PRO (isotropic, norm-matched)	10^{-4}	0.118223630	61.7134

60.5762% to 61.7134%, or 15 additional correct answers. Both changes alter task performance despite similar validation losses. The main comparisons use the fixed default configuration.

4.5 THE FINAL UPDATES SPREAD ACROSS MORE MATRIX DIRECTIONS

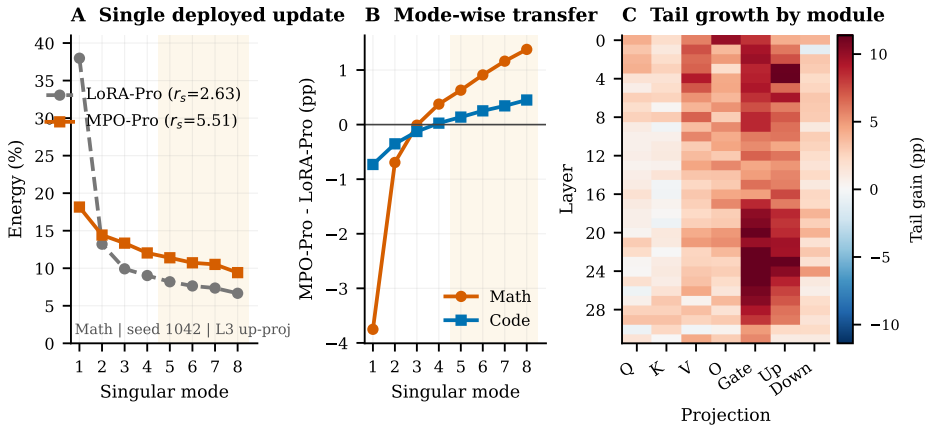


Figure 2: **Different update spectra.** (A) A representative math MLP up-projection. (B) Three-run spectral summaries on math and code. (C) Change in the energy share of directions 5–8 across projection types. Statistics use each method's final rank-8 weight update, with MPO cores contracted before analysis.

Adapters with the same rank can use their available matrix directions differently. To examine this, we decompose each final update into singular directions. We order directions by decreasing singular value. A direction's energy is its squared singular value; its share is the fraction of total update energy it carries.

Stable rank, the squared Frobenius norm divided by the squared largest singular value, summarizes this concentration. A larger value indicates a more evenly spread update without increasing the LoRA rank. We use energy shares for individual directions and stable rank for the whole matrix.

The rank-8 comparison shows a redistribution within the eight available directions (Figure 2). In the representative layer, stable rank rises from 2.63 for LoRA-Pro to 5.51 for MPO-PRO. The leading direction carries less energy, while the weaker directions carry more.

The pattern extends beyond one layer: directions 5–8 gain energy share in more than 90% of the 224 adapted modules on both math and code, and mean stable rank increases on both tasks. Appendix D gives the exact module-level summary values.

Uniformly enlarging an update would leave its energy shares and stable rank unchanged. The observed changes therefore reflect redistribution across the available directions, and the difference remains after the temporary cores are contracted into the rank-8 adapter.

For dense adaptation, MPO-PRO achieves the lowest validation loss among the compared methods (Appendix G, Figure 4). In the illustrated layer, the leading 32 singular directions carry 14.1% of its final update energy. Together, the dense and fixed-rank analyses show that temporary factorizations can change how update energy is distributed in the exported matrix.

5 DISCUSSION AND LIMITATIONS

Like MPO, SVD, CP, and Tucker provide factorized representations, but their factors connect differently. SVD writes a matrix as a product of two factors, with the singular values absorbed into either factor (Henry and Hofrichter, 1992). CP expresses a tensor as a sum of rank-one components (Hitchcock, 1927). It is also a special case of Tucker decomposition with a superdiagonal core. General Tucker decomposition connects factor matrices through a central core (Tucker, 1966), whose size grows as the product of the mode ranks. MPO instead connects local cores in a chain, with bond dimensions controlling its capacity. These structures determine how factor updates map to matrix updates.

MPO suits our initialization, update matching, and deployment procedure. Full bond dimensions allow successive SVDs to reconstruct the initial matrix exactly (Oseledets, 2011); chain contractions support Jacobian-vector products for matching and for analyzing P, E, and H. The same construction covers dense updates and LoRA factors, then contracts to the original deployment format. Other factorizations could also be used, but their parameter structures and update maps may produce different trajectories.

The experiments cover Qwen3.5-0.8B, Llama-2-7B, and T5-Base, with three-seed main comparisons. The link between the first-order implementation deviation E and the GSM8K gain comes from the matched rank-8 experiments. Gains are concentrated in math and code: LoRA-Pro scores higher on MT-Bench, while matched GLUE averages are nearly unchanged. The dense branch adds training memory and computation. Zero damping shows promise in the current ablation, but its performance across tasks and seeds needs further evaluation. Quantized checkpoints and complete local-agent tasks remain to be evaluated.

6 CONCLUSION

Temporary factorization changes how a matrix is learned while preserving how it is deployed. MPO-PRO uses this principle for both dense and fixed-rank adaptation. It improves math and code performance over full fine-tuning on Qwen3.5-0.8B and over matched LoRA-Pro on Llama-2-7B, while retaining the corresponding deployment formats.

The fixed-rank analysis shows that a temporary MPO coordinate can closely match a reference local direction while producing a different sequence of contracted states. The contracted adapters redistribute energy across their available singular directions. Across three seeds, retaining the first-order implementation deviation E preserves the high GSM8K endpoint, whereas retaining only $P + H$ returns performance to the Native-Pro level despite nearly identical validation losses. Temporary factorization acts as a trajectory-level optimization resource within a fixed deployment family. This perspective also applies beyond LoRA post-training: the same mathematical formulation and framework can be used for full fine-tuning and even pretraining.

AI USE STATEMENT

Generative AI tools assisted with manuscript organization and language editing, literature search and summarization, and the formulation and presentation of parts of the mathematical analysis. The authors checked the cited sources, verified the derivations against the implementation, validated all numerical results against experiment records, reviewed all AI-assisted content, and take responsibility for the final manuscript.

Reproducibility Statement. We provide detailed implementation, training, and evaluation settings in the appendices. We plan to publicly release the source code and scripts for reproducing the main experiments upon acceptance of the paper.

REFERENCES

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 244–253. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/arora18a.html.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c0c783b5fc0d7d808f1d14a6e9c8280d-Paper.pdf.

Valérie Castin, Kimia Nadjahi, Pierre Ablin, and Gabriel Peyré. Balanced loRA: Removing parameter invariance to accelerate convergence. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=kHInw3cjCP.

Mark Chen et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc., 2023. doi: 10.52202/075280-0441. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf.

Jing Gao, Zhong-Yi Lu, Pan Zhang, and Ze-Feng Gao. Strategic over-parameterization for generalizable low-rank adaptation, 2026. URL https://arxiv.org/abs/2605.16470.

Ze-Feng Gao, Song Cheng, Rong-Qiang He, Z. Y. Xie, Hui-Hai Zhao, Zhong-Yi Lu, and Tao Xiang. Compressing deep neural networks by matrix product operators. *Phys. Rev. Res.*, 2:023300, Jun 2020. doi: 10.1103/PhysRevResearch.2.023300. URL https://link.aps.org/doi/10.1103/PhysRevResearch.2.023300.

Ze-Feng Gao, Kun Zhou, Peiyu Liu, Wayne Xin Zhao, and Ji-Rong Wen. Small pre-trained language models can be fine-tuned as large models via over-parameterization. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3819–3834, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.212. URL https://aclanthology.org/2023.acl-long.212/.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 17783–17806. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/hayou24a.html.

E.R. Henry and J. Hofrichter. [8] singular value decomposition: Application to analysis of experimental data. In *Numerical Computer Methods*, volume 210 of *Methods in Enzymology*, pages 129–192. Academic Press, 1992. doi: https://doi.org/10.1016/0076-6879(92)10010-B. URL https://www.sciencedirect.com/science/article/pii/007668799210010B.

Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927. doi: https://doi.org/10.1002/sapm192761164. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm192761164.

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.

Xiaolin Hu, Xiang Cheng, Peiyu Liu, Wei Liu, Jian Luan, Bin Wang, and Yong Liu. Dota: Weight-decomposed tensor adaptation for large language models. In Xintao Wu, Myra Spiliopoulou, Can Wang, Vipin Kumar, Longbing Cao, Yanqiu Wu, Yu Yao, and Zhangkai Wu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 16–27, Singapore, 2025. Springer Nature Singapore. ISBN 978-981-96-8186-0.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora, 2023. URL https://arxiv.org/abs/2312.03732.

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.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32100–32121. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 121038–121072. Curran Associates, Inc., 2024. doi: 10.52202/079017-3846. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/db36f4d603cc9e3a2a5e10b93e6428f2-Paper-Conference.pdf.

I. V. Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011. doi: 10.1137/090752286. URL https://doi.org/10.1137/090752286.

B Pirvu, V Murg, J I Cirac, and F Verstraete. Matrix product operator representations. *New Journal of Physics*, 12(2):025012, feb 2010. doi: 10.1088/1367-2630/12/2/025012. URL https://doi.org/10.1088/1367-2630/12/2/025012.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Piotr Teterwak, Kate Saenko, Bryan A. Plummer, and Ser-Nam Lim. Op-lora: The blessing of dimensionality. *CoRR*, abs/2412.10362, 2024. URL https://doi.org/10.48550/arXiv.2412.10362.

Hugo Touvron et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3): 279–311, 1966. doi: 10.1007/BF02289464.

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.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 54905–54931. Curran Associates, Inc., 2024. doi: 10.52202/

079017-1741. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/62c4718cc334f6a0a62fb81c4a2095a1-Paper-Conference.pdf.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-pro: Are low-rank adapters properly optimized? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gTwRMU3lJ5.

Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 56946–56965. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/yaras24a.html.

Jui-Nan Yen, Si Si, Zhao Meng, Felix Yu, Sai Surya Duvvuri, Inderjit S Dhillon, Cho-Jui Hsieh, and Sanjiv Kumar. LoRA done RITE: Robust invariant transformation equilibration for loRA optimization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VpWki1v2P8.

Dan Zhang, Ziniu Hu, Sining Zhoubian, Zhengxiao Du, Kaiyu Yang, Zihan Wang, Yisong Yue, Yuxiao Dong, and Jie Tang. Sciinstruct: a self-reflective instruction annotated dataset for training scientific language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 1443–1473. Curran Associates, Inc., 2024. doi: 10.52202/079017-0046. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/02ee6b7295f720407b56c457b34c54d5-Paper-Datasets_and_Benchmarks_Track.pdf.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned lora for fine-tuning foundation models. In *ICML*, 2024. URL https://openreview.net/forum?id=IwqE4QqBew.

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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

A Matrix Shapes and Optimization Details

A.1 LoRA Differential

Let δA and δB be one-step increments of the LoRA factors. Expanding $s(B+\delta B)(A+\delta A) - sBA$ gives the exact weight change ΔW:

$$\Delta W = s(B\delta A + \delta B A) + s\delta B\delta A. \tag{9}$$

The first two terms are linear in the factor increments; the last is second order. For a weight gradient $D = \nabla_W \mathcal{L}$, the chain rule gives $\nabla_A \mathcal{L} = sB^\top D$ and $\nabla_B \mathcal{L} = sDA^\top$. These are the standard LoRA derivatives (Hu et al., 2022); LoRA-Pro uses corrected factor directions as described by Wang et al. (2025).

A.2 MPO Indices and Full Bond Dimensions

For $M \in \mathbb{R}^{I \times J}$, choose $I = \prod_{k=1}^{p} i_k$ and $J = \prod_{k=1}^{p} j_k$. Here I and J denote matrix dimensions within this subsection, p is the number of cores, and i_k and j_k are the local row and column sizes at core k. This rewrites a row index as $(a_1, \ldots, a_p)$ and a column index as $(b_1, \ldots, b_p)$. The kth *core* stores a matrix $C_k(a_k, b_k) \in \mathbb{R}^{d_{k-1} \times d_k}$ for each local index pair. The shared sizes d_k are the *bond dimensions*. Setting the two boundary sizes to $d_0 = d_p = 1$ makes the following product a scalar for each matrix entry:

$$M_{(a_1,\ldots,a_p),(b_1,\ldots,b_p)} = C_1(a_1, b_1) \cdots C_p(a_p, b_p), \qquad M = \mathcal{D}(\theta), \tag{10}$$

where $1 \le a_k \le i_k$, $1 \le b_k \le j_k$, and θ collects all core entries. The slice $C_k(a_k, b_k)$ fixes the two local indices of core C_k. Evaluating the products for all row and column index tuples reconstructs M; this operation is called contraction (Pirvu et al., 2010; Oseledets, 2011).

We retain the full bond dimensions

$$d_k = \min\left\{ \prod_{\ell=1}^{k} i_\ell j_\ell, \prod_{\ell=k+1}^{p} i_\ell j_\ell \right\}, \qquad N_\theta = \sum_{k=1}^{p} d_{k-1} i_k j_k d_k. \tag{11}$$

Here N_θ counts the core parameters, and ℓ indexes the local row–column pairs in each product. To initialize the cores, reshape M into an array with paired indices $(a_1, b_1), \ldots, (a_p, b_p)$ and apply successive SVDs. Each SVD separates the next local pair from the remaining array, producing one core and a residual array for the next step. Retaining all singular values, including zeros, gives the full bond dimensions above and reconstructs M to floating-point precision (Oseledets, 2011). These dimensions can represent any $I \times J$ matrix; our layouts use $N_\theta > IJ$ training parameters. Contraction returns an ordinary matrix with IJ entries. Bond dimensions therefore control the training representation, while the exported LoRA update still has rank at most r.

The key property of $\mathcal{D}$ is *multilinearity*: changing one core changes the reconstructed matrix linearly when the other cores are fixed. This gives the local Jacobian used to analyze the trajectory.

A.3 Dense Update Derivation

Our five-core layout places all local indices with size greater than one at the two ends: $i_2 = i_3 = i_4 = j_2 = j_3 = j_4 = 1$. Let $\mathcal{R}$ regroup entries so that (a_1, b_1) indexes rows and (a_5, b_5) indexes columns. Reshaping the cores into matrices then gives

$$\mathcal{R}\big(\mathcal{D}(\theta)\big) = C_1^1 C_2^1 C_3^1 C_4^1 C_5^1. \tag{12}$$

In this equation each C_k^1 denotes the matrix view of the same four-index core, with singleton local dimensions suppressed and the end indices grouped. $\mathcal{R}$ only rearranges entries and preserves the Frobenius inner product, the sum of entrywise products of two matrices. The inverse rearrangement $\mathcal{R}^{-1}$ restores the original weight layout.

To derive the matrix update, write $a = \mathrm{vec}(\mathcal{D}(\theta))$, $u = \mathrm{vec}(U)$, and $d = \nabla_u \mathcal{L}$. The Jacobian $J = \partial a / \partial \theta$ maps small core changes to matrix changes. Since $u = \gamma a$, the chain rule yields

$$\nabla_\gamma \mathcal{L} = a^\top d, \qquad \nabla_\theta \mathcal{L} = \gamma J^\top d. \tag{13}$$

For simultaneous gradient-descent increments $\delta\gamma = -\eta a^\top d$ and $\delta\theta = -\eta\gamma J^\top d$, substitution into Equation 4 gives

$$\Delta u_1 = -\eta\big(aa^\top + \gamma^2 JJ^\top\big)d. \tag{14}$$

This is the first-order change. The exact change is $(\gamma + \delta\gamma)\,\mathrm{vec}(\mathcal{D}(\theta + \delta\theta)) - \gamma a$, which also includes products of parameter increments.

This transformation has an explicit matrix form. Set $L_k = C_1^1 \cdots C_{k-1}^1$ and $R_k = C_{k+1}^1 \cdots C_5^1$, with empty products equal to identity matrices. The matrices L_k and R_k are the products to the left and right of core k. For a matrix direction Z in the regrouped layout, define

$$\mathcal{M}_C[Z] = \sum_{k=1}^{5} L_k L_k^\top Z R_k^\top R_k. \tag{15}$$

Here $\mathcal{M}_C$ is the linear map induced by the five core gradients. For a scalar loss whose gradient with respect to the regrouped product is Z, the gradient with respect to C_k^1 is $L_k^\top Z R_k^\top$, whose contribution to the product change is $L_k L_k^\top Z R_k^\top R_k$. Consequently, $JJ^\top \operatorname{vec}(Z_0) = \operatorname{vec}(\mathcal{R}^{-1}(\mathcal{M}_C[\mathcal{R}(Z_0)]))$ for any matrix direction Z_0 in the original weight layout. Thus $\mathcal{M}_C$ expresses the Jacobian product $JJ^\top$ using ordinary matrix multiplications.

The implementation uses Adam on the cores and γ. Its parameter increments give a first-order matrix change through Equation 4; Equation 14 is the gradient-descent case. Export merges the final update into the pretrained weight: $W_{\text{final}} = W_0 + \gamma_T \mathcal{D}(\theta_T)$, where T is the final optimizer step.

A.4 Regularized Matching and the Gram-CG Solve

Let $d_A = \operatorname{vec}(D_A)$ be the vectorized LoRA-Pro direction, J_A the core-to-matrix Jacobian, and K_A its matrix-space Gram matrix:

$$J_A = \frac{\partial \operatorname{vec}(A)}{\partial \theta_A}, \qquad K_A = J_A J_A^\top. \tag{16}$$

Directly passing d_A through the chain rule gives the core direction $J_A^\top d_A$. Mapping this direction back to the vectorized A gives $K_A d_A$, rather than d_A. Equation 6 chooses a core direction to reduce this mismatch. Differentiating the objective with respect to h gives $J_A^\top(J_A h - d_A) + \lambda h = 0$. Thus its normal equation is $(J_A^\top J_A + \lambda I)h_\lambda = J_A^\top d_A$. Because $\lambda > 0$, the objective has a unique minimizer. The identity $(J_A^\top J_A + \lambda I)J_A^\top = J_A^\top(K_A + \lambda I)$ gives an equivalent system for an auxiliary matrix-space vector z, followed by a mapping to the core direction h_λ. Each I is the identity matrix of the size required by its equation:

$$(K_A + \lambda I)z = d_A, \tag{17}$$

$$h_\lambda = J_A^\top z, \qquad \delta\theta_A = -\eta h_\lambda, \tag{18}$$

$$\operatorname{vec}(\Delta A_1) = -\eta K_A (K_A + \lambda I)^{-1} d_A. \tag{19}$$

Here ΔA_1 is the first-order reconstructed change for an exact linear solve. Along an eigenvector of K_A with eigenvalue σ, the target direction is multiplied by $\sigma/(\sigma + \lambda)$. If J_A has full row rank, the zero-damping limit recovers the target exactly and gives the minimum-norm core direction satisfying $J_A h = d_A$.

The products with K_A can be evaluated without storing the Jacobian. Define a linear map for each core:

$$\mathcal{T}_1[X] = \mathcal{D}(X, C_2^2, C_3^2),$$
$$\mathcal{T}_2[X] = \mathcal{D}(C_1^2, X, C_3^2),$$
$$\mathcal{T}_3[X] = \mathcal{D}(C_1^2, C_2^2, X).$$

Here X has the shape of the replaced core, and $\mathcal{T}_k$ maps that core's increment to a matrix increment. Let $\mathcal{T}_k^*$ be its adjoint under the Frobenius inner product, that is, $\langle \mathcal{T}_k[X], Z\rangle_F = \langle X, \mathcal{T}_k^*[Z]\rangle_F$ for a matrix Z. The subscript F denotes the Frobenius inner product. For a vector v with rn entries, let $V \in \mathbb{R}^{r\times n}$ be its matrix reshape. Then

$$K_A v = \operatorname{vec}\left(\sum_{k=1}^{3} \mathcal{T}_k[\mathcal{T}_k^*[V]]\right). \tag{20}$$

This follows by writing J_A as three column blocks, one per core. It gives the matrix-free operation used inside each conjugate-gradient iteration. Let $\widehat{z}$ be the computed solution and $\widehat{Z}$ its $r \times n$ reshape. The direction for core k is then $h_k = \mathcal{T}_k^*[\widehat{Z}]$.

The implementation solves Equation 17 by preconditioned conjugate gradients (PCG), using these contraction-based products. It sets $\lambda = \mu \max\{d_A^\top K_A d_A / \|d_A\|_2^2, \epsilon\}$ for $d_A \neq 0$, where μ is a dimensionless damping coefficient and $\epsilon > 0$ is a numerical floor. The ratio scales damping to the magnitude of K_A along the target direction. A zero target gives a zero step. For the computed solution $\widehat{z}$, let ΔA_1 now denote the first-order change from its core direction $J_A^\top \widehat{z}$, and define the residual $\rho = d_A - (K_A + \lambda I)\widehat{z}$. The matching error is the vector e_A defined by

$$e_A \equiv \operatorname{vec}(\Delta A_1) + \eta d_A = \eta(\lambda \widehat{z} + \rho). \tag{21}$$

Indeed, $K_A\widehat{z} = d_A - \lambda\widehat{z} - \rho$, and the implemented core direction is $J_A^\top\widehat{z}$. This yields $e_A = \eta(d_A - K_A\widehat{z})$ and the identity above. This identity describes the first-order step before additional rounding of the applied core increment. For $d_A \neq 0$, its relative error satisfies

$$\frac{\|e_A\|_2}{\eta\|d_A\|_2} \leq \frac{\lambda\|\widehat{z}\|_2 + \|\rho\|_2}{\|d_A\|_2}. \tag{22}$$

The two terms bound the contributions from damping and the linear-solve residual. Full bond dimensions provide global representational capacity, whereas local direction matching depends on the current Jacobian. If J_A is rank deficient, the realized direction in the exact-solve, zero-damping limit is the orthogonal projection of d_A onto the range of J_A.

A.5 Finite-Step Expansion of the Three-Core Update

Let δC_k^2 be the applied increment to LoRA core C_k^2, for $k \in \{1, 2, 3\}$. Expanding the contraction after all three increments groups the terms by the number of increments they contain. The terms with one increment give the first-order change L:

$$L = \mathcal{D}(\delta C_1^2, C_2^2, C_3^2) + \mathcal{D}(C_1^2, \delta C_2^2, C_3^2) + \mathcal{D}(C_1^2, C_2^2, \delta C_3^2),$$

which satisfies $\mathrm{vec}(L) = J_A\delta\theta_A$. The remainder H contains all terms with two or three increments:

$$\begin{aligned} H = {} & \mathcal{D}(\delta C_1^2, \delta C_2^2, C_3^2) + \mathcal{D}(\delta C_1^2, C_2^2, \delta C_3^2) \\ & + \mathcal{D}(C_1^2, \delta C_2^2, \delta C_3^2) + \mathcal{D}(\delta C_1^2, \delta C_2^2, \delta C_3^2). \end{aligned}$$

The first three terms contain two core increments and the last contains three. Together they give $\Delta A = L + H = P + E + H$ in Equation 8. For the Gram-CG increment $\delta\theta_A = -\eta J_A^\top\widehat{z}$, the vectorized deviation $\mathrm{vec}(E)$ equals the matching error e_A in Equation 21, before any additional rounding of the applied core increment. This links the damping and solve residual to the first-order component used in the intervention.

A.6 Implementation and Tensor Shapes

Exact full-bond initialization. For $I = \prod_{k=1}^{p} i_k$ and $J = \prod_{k=1}^{p} j_k$, the cores used in Equation 10 have shapes

$$\theta = \left[\mathrm{vec}(C_1)^\top, \ldots, \mathrm{vec}(C_p)^\top\right]^\top, \qquad C_k \in \mathbb{R}^{d_{k-1}\times i_k \times j_k \times d_k}, \qquad d_0 = d_p = 1. \tag{23}$$

Using full intermediate dimensions and retaining every singular value reconstructs the original matrix to floating-point precision.

Dense update map. The dense branch stores five cores and a scalar during training. At the final step T, it computes the update $U_{\text{deploy}} = \gamma_T\mathcal{D}(\theta_T)$ and merges it into W_0. The exported checkpoint stores the resulting dense weight.

Dense training configuration. The five-core chain uses the end-index layout in Equation 12, with full bonds determined by the matrix shape. The cores are initialized from the frozen base weight and the scalar gate from 10^{-8}. Both are trained with Adam in 32-bit floating point (FP32). Across 186 projections, the representation contains 2,016,215,040 core parameters and 186 gates, totaling 2,016,215,226 trainable parameters. Export merges each scaled contraction into the Qwen3.5-0.8B checkpoint.

Gram correction and preconditioning. PCG rescales its search directions using a preconditioner constructed from the two outer cores. The preconditioner stays fixed during each solve. If C_L^2 and C_R^2 are the matrices obtained by grouping their outer and shared indices, its action is

$$Q[V] = (C_L^2(C_L^2)^\top)^\dagger\, V\, ((C_R^2)^\top C_R^2)^\dagger. \tag{24}$$

Here Q is the preconditioning map, V is the $r \times n$ reshape of an input vector v, and $\dagger$ denotes the matrix pseudoinverse. The matrices C_L^2 and C_R^2 are views of C_1^2 and C_3^2, respectively. The output is vectorized again for the conjugate-gradient solve. At the final step T, the core vector $\theta_{A,T}$ is contracted for export:

$$A_{\text{deploy}} = \mathcal{D}(\theta_{A,T}), \tag{25}$$

The exported factor A_{deploy} has the standard LoRA shape $r \times n$.

Scrambled-coordinate control. For each adapted layer l, fixed row and column permutations are derived from the SHA256 digest of the recipe `scrambled-mpo-v1|SCRAMBLE_SEED=20260811|layer-name`. The same five-core SVD initialization and factor normalization are applied to

$$\widetilde{W}_{0,l} = P_{o,l} W_{0,l} P_{i,l}^{\top}, \qquad M_l = P_{o,l}^{\top} \mathcal{D}(\theta_l) P_{i,l}.$$

Here $P_{o,l}$ and $P_{i,l}$ are the output-row and input-column permutation matrices for layer l, $\widetilde{W}_{0,l}$ is the permuted pretrained weight, and M_l is the contraction restored to the original order. The vector θ_l contains the layer's dense cores. The permutations are shared across training seeds, and initialization is accepted only when $\|M_l(0) - W_{0,l}\|_F / \|W_{0,l}\|_F < 10^{-5}$ for every adapted layer. Here $M_l(0)$ denotes the initial contraction, and $\|\cdot\|_F$ is the Frobenius norm, the square root of the sum of squared entries. The effective update is $U_l = \gamma_l M_l$, where γ_l is the layer's trainable scale. The inverse permutations restore the original row and column order. Core shapes, bond dimensions, gate initialization, and parameter count match the default MPO, while the assignment of weight entries to local index groups changes. Initialization uses CUDA `gesvd` in 64-bit floating point (FP64); the resulting cores are stored and trained in FP32.

Fixed-rank update and core layouts. At each optimizer step, the fixed-rank branch reconstructs $A = \mathcal{D}(\theta_A)$ for each adapted layer, computes the LoRA-Pro target d_A, and computes $(K_A + \lambda I)v$ through MPO contractions without forming the matrix. The outer-core preconditioner in Equation 24 is fixed during the solve. PCG starts from zero and updates the standard residual, search direction, and preconditioned residual until the relative residual reaches 10^{-3} or eight iterations have been completed. A vector–Jacobian product then maps the solution to the core direction $J_A^{\top} z$. Only the final contracted factor is exported.

The input and output modes below are the local column sizes j_k and row sizes i_k, respectively. At rank 8, the three-core construction uses input modes $[16, 16, 16]$ for width 4096 and $[16, 43, 16]$ for width 11008, output modes $[2, 2, 2]$, and full bonds $[1, 32, 32, 1]$. An 8×4096 factor has 34,816 temporary core parameters and contracts to 32,768 entries; an 8×11008 factor has 90,112 temporary parameters and contracts to 88,064 entries. Across all adapted projections, the parameterization contains 20,447,232 trainable parameters and exports the same 19,988,480-parameter rank-8 adapter as LoRA-Pro.

At rank 16, only A is factorized and B remains dense. The balanced three-core layout uses output modes $[2, 4, 2]$, input modes $[16, 16, 16]$ for width 4096 and $[16, 43, 16]$ for width 11008, and full bonds $[1, 32, 32, 1]$ without truncation. LoRA-Pro trains 39,976,960 adapter parameters; MPO-PRO trains 40,435,712 temporary parameters. Both export 39,976,960 parameters in a standard rank-16 adapter.

B MATCHED FIXED-RANK PROTOCOL

The matched MPO-PRO and reproduced LoRA-Pro runs use the same rank, initialization, target modules, and forward scaling at each of ranks 8 and 16, as in Section 4. Both use a LoRA scaling hyperparameter $\alpha = 16$ with rank-stabilized scaling $16/\sqrt{r}$, giving $16/\sqrt{8}$ at rank 8 and 4 at rank 16. Training uses SGD at learning rate 2×10^{-5}, no momentum, no weight decay, cosine decay, and a 0.03 warmup ratio. Each run performs 3,125 optimizer steps at effective batch size 32 over one epoch on two H100 GPUs. The same seven attention and MLP projection types are adapted in every run. The dense A factor begins from the reference LoRA initialization, and the MPO cores are an exact full-bond decomposition of that same matrix.

The rank-8 comparison uses three runs. The rank-16 results use seeds 0, 1042, and 3407; the launcher default instead ends in 3047. Both fixed-rank settings use the GSM8K and HumanEval decoding protocols in Appendix F.

At rank 16, both methods use zero dropout, sequence length 1024, BF16, and FlashAttention2. The targets are `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`, covering 224 modules across 32 layers. Our rank-16 LoRA-Pro configuration extends the official rank-8 training logic.

The Gram-CG configuration uses an eight-iteration cap, relative residual tolerance 10^{-3}, and curvature-relative damping coefficient $\mu = 10^{-4}$ as defined in Appendix A.4. The rank-16 configu-

ration also sets the correction cap to 100. For the recorded rank-8 direction diagnostics, the mean solve count remains below six iterations, with residuals below tolerance. HumanEval evaluation uses $164 \times 200 = 32,800$ generations per run; GSM8K generates one greedy completion per question.

Update-decomposition protocol. The four-branch experiment uses Llama-2-7B, rank 8, three seeds, BF16 core and base storage, effective batch size 32, SGD with learning rate 2×10^{-5}, and 3,125 optimizer steps. At each step, P, L, E, and H are computed in reconstructed A-space as in Equation 8. The implementation names the four branches `native_pro` (P), `full` ($P + E + H$), `no_h` ($P + E$), and `no_e` ($P + H$). The intervention acts on the reconstructed update used by each branch, and the three-seed endpoint statistics are reported in Table 4.

B.1 LoRA-Over Reference-Code Protocol

Our LoRA-Over implementation follows the authors' source code, using $\alpha = 2r$ and ordinary LoRA scaling $\alpha/r = 2$ in both the initial LoRA stage and the LoRA-Over stage. At rank 16, this gives $\alpha = 32$, whereas LoRA-Pro and MPO-Pro use $\alpha = 16$ and rank-stabilized scaling 4.

For each rank, seed, and task, we first train vanilla LoRA for one epoch with learning rate 2×10^{-5}, cosine decay, and warmup ratio 0.03. LoRA-Over starts from this trained adapter and runs for another epoch, using runtime decomposition with `topN=30`, `split_num=8`, and zero warmup. Math uses main and MPO learning rates 10^{-4} and 9.3×10^{-5}; Code uses 1.2×10^{-4} and 1.1×10^{-4}.

The rank-16 runs use Llama-2-7B, seeds $0, 1042, 3407$, zero dropout, sequence length 1024, BF16, and FlashAttention2. All linear projections are adapted, excluding embeddings, normalization, and the LM head. Both tasks use the same cached 100K training and 10K held-out examples. Two GPUs, each processing four microbatches of four examples per optimizer step, give an effective batch size of 32. GSM8K uses greedy decoding on all 1,319 questions. HumanEval uses 200 samples for each of 164 tasks, temperature 0.8, and nucleus-sampling threshold $p = 0.95$, reporting pass@1, pass@10, and pass@100.

The comparisons in Table 2 share the backbone, data, adapter rank, rank-16 seeds, effective batch size, precision, and evaluation settings. LoRA-Over uses its reference-code optimizer, scaling, learning rates, and two-stage schedule. LoRA-Pro and MPO-Pro use one epoch of LoRA-Pro updates. The table therefore compares the complete training recipes at the same deployment rank, with different total training budgets.

B.2 MT-Bench Evaluation

The rank-8 dialogue experiment uses a separate WizardLM-52K training pipeline with seeds $0, 1042, 3047$, $\alpha = 16$, rank-stabilized scaling, one epoch, learning rate 2×10^{-5}, cosine decay, and warmup ratio 0.03. These dialogue adapters are trained separately from the math and code adapters.

FastChat generates one two-turn conversation for each of its 80 questions, using a maximum of 1,024 new tokens and the local `lorapro_table2_instruction` prompt template. The generation temperatures are 0.7 for writing and roleplay; 0 for extraction, math, coding, and reasoning; and 0.1 for STEM and humanities. The generation code does not explicitly set the nucleus-sampling threshold p.

The judge is the legacy `gpt-4` model in single-answer grading mode, with serial execution (`parallel=1`). Full evaluation contains 80 two-turn conversations and 160 turn judgments. The reported score uses first-turn judgments for question IDs 81–160. For each model, question, and turn, we retain the first successful judgment with a nonnegative score, average over the 80 questions within each seed, and report the three-seed mean and sample standard deviation. The saved evaluation configuration specifies neither a FastChat commit nor a dated GPT-4 snapshot. The question, judge-prompt, and reference-answer files are available from the FastChat main branch; their exact revisions are not recorded in the snapshot.

The rank-16 dialogue evaluation uses the same FastChat generation and GPT-4 judging protocol described above.

C PER-RUN ENDPOINT VALUES

Compared with reproduced LoRA-Pro, rank-8 held-out loss falls from 0.1222 to 0.1181 on Math and from 0.2715 to 0.2673 on Code. At rank 16, the corresponding changes are from 0.118889 to 0.116475 and from 0.265514 to 0.261512.

Table 6 lists the final held-out loss for the three rank-8 MPO-PRO runs; Table 7 gives the matched rank-16 results. Here s.d. denotes the sample standard deviation across runs.

Table 6: Per-run held-out loss for three-core rank-8 adaptation.

Task	Run 1	Run 2	Run 3	Mean ± s.d.
Math	0.117941	0.118178	0.118054	0.118058±0.000119
Code	0.267381	0.267156	0.267215	0.267251±0.000117

Table 7: Per-run held-out loss for the matched rank-16 comparison.

Task	Method	Seed 0	Seed 1042	Seed 3407	Mean ± s.d.
Math	LoRA-Pro	0.119121	0.118698	0.118847	0.118889±0.000215
Math	MPO-PRO	**0.116590**	**0.116504**	**0.116331**	**0.116475±0.000132**
Code	LoRA-Pro	0.265668	0.265407	0.265466	0.265514±0.000137
Code	MPO-PRO	**0.261860**	**0.261302**	**0.261374**	**0.261512±0.000304**

D OPTIMIZATION TRAJECTORIES AND CONTRACTED GEOMETRY

Comparing LoRA-Pro with MPO-PRO in the representative math layer of Figure 2A (seed 1042, layer 3 MLP up-projection), the leading energy share falls from 37.98% to 18.15% and stable rank rises from 2.63 to 5.51. For each of the 224 adapted modules, we compare the methods' three-run mean energy shares. Directions 5–8 gain share in 96.0% of math modules and 91.5% of code modules. Mean stable rank rises from 4.807 to 5.522 on Math and 6.119 to 6.365 on Code.

The training-loss curves in Figure 3 follow similar paths early in training and separate later. On math, MPO-PRO maintains lower training loss over the final 500 steps. Final held-out losses appear in Appendix C, and task scores in Table 2.

Table 8: Rank-8 weight-update statistics averaged across adapted modules and three runs. Here ΔW is the final weight update, W_0 is the pretrained weight, and $\|\cdot\|_F$ is the Frobenius norm. Top-1 energy is the fraction of update energy in the leading singular direction. HE p@1 is the task-level HumanEval percentage from Table 2.

	Math			Code		
Method	$\|\Delta W\|_F/\|W_0\|_F$	Stable rank	Top-1 energy	$\|\Delta W\|_F/\|W_0\|_F$	Stable rank	HE p@1
LoRA-Pro	0.0372	4.807	0.2273	0.0593	6.119	20.242
Three-core MPO-PRO	0.0727	**5.522**	**0.1898**	0.0838	**6.365**	**21.588**

E LANGUAGE-UNDERSTANDING BREADTH CHECK

The matched five-task average is nearly unchanged: 88.21 for LoRA-Pro and 88.23 for MPO-PRO. The differences are −0.03, 0.00, −0.03, −0.03, and +0.17 points on MNLI, SST-2, CoLA, QNLI, and MRPC, respectively.

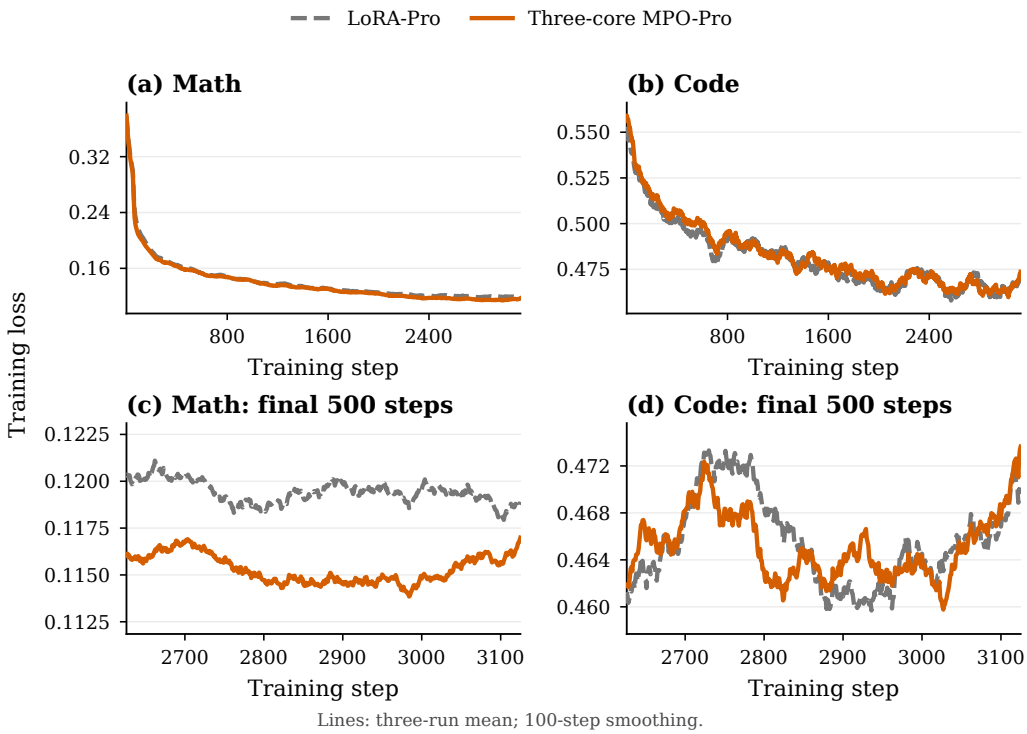


Figure 3: Training loss for LoRA-Pro and three-core MPO-Pro on Math and Code. Full trajectories appear in the top row and the final 500 steps in the bottom row. Lines are three-run means with 100-step smoothing.

Table 9: T5-Base (Raffel et al., 2020) rank-8 adaptation on five GLUE tasks (Wang et al., 2019). The published block transcribes the results reported by LoRA-Pro (Wang et al., 2025) and LoRA-Over (Gao et al., 2026); the shared baseline rows have identical values in both papers. The lower block gives our matched comparison with batch size 640. Scores are reported on a 0–100 scale as three-seed means ± sample standard deviations; Avg. is the arithmetic mean across tasks. Each block uses its own training protocol.

Method	MNLI	SST-2	CoLA	QNLI	MRPC	Avg.
Full FT	86.33±0.00	94.75±0.21	80.70±0.24	93.19±0.22	84.56±0.73	87.91
LoRA	85.30±0.04	94.04±0.11	69.35±0.05	92.96±0.09	68.38±0.01	82.08
PiSSA	85.75±0.07	94.07±0.06	74.27±0.39	93.15±0.14	76.31±0.51	84.71
rsLoRA	85.73±0.10	94.19±0.23	72.32±1.12	93.12±0.09	52.86±2.27	79.64
LoRA+	85.81±0.09	93.85±0.24	77.53±0.20	93.14±0.03	74.43±1.39	84.95
LoRA-GA	85.70±0.09	94.11±0.18	80.57±0.20	93.18±0.06	85.29±0.24	87.77
DoRA	85.67±0.09	94.04±0.53	72.04±0.94	93.04±0.06	68.08±0.51	82.57
AdaLoRA	85.45±0.11	93.69±0.20	69.16±0.24	91.66±0.05	68.14±0.28	81.62
LoRA-Pro	86.03±0.19	94.19±0.13	81.94±0.24	93.42±0.05	86.60±0.14	88.44
LoRA-Over-SVD	85.33±0.06	94.13±0.29	74.52±0.39	93.05±0.11	75.76±0.23	84.56
LoRA-Over-MPO	85.42±0.09	94.35±0.32	80.79±0.39	93.14±0.39	82.84±0.53	87.31
LoRA-Over-MPO$_P$	85.59±0.08	94.41±0.14	81.02±0.16	93.19±0.05	84.07±0.69	87.66
LoRA-Over-MPO$_R$	85.79±0.12	94.50±0.58	81.21±0.41	93.26±0.08	85.38±1.22	88.03
LoRA-Pro (matched)	85.91±0.05	94.04±0.20	81.53±0.29	93.32±0.07	86.27±0.25	88.21
MPO-Pro	85.88±0.04	94.04±0.11	81.50±0.17	93.29±0.01	86.44±0.37	88.23

F Dense Three-Seed Protocol

The Qwen3.5-0.8B study uses seeds 0, 1042, and 3047. For each task, MPO-Pro and full fine-tuning share the corresponding 100K training dataset, held-out split, causal language modeling objective, Adam optimizer, zero weight decay, and evaluation implementation. The MPO parameterization uses learning rate 10^{-4} and the main full-fine-tuning baseline uses 10^{-5}. Table 10 reports both tested

learning rates. These fixed settings are shared by all three seeds of the corresponding method. The 10^{-5} full-fine-tuning runs and MPO runs use two H100 GPUs per run, whereas the 10^{-4} full-fine-tuning control and Scrambled MPO use one H100 with gradient accumulation; all configurations retain an effective batch size of 32. Training costs for the main comparison are reported in Table 1. The table sums GPU-hours over both tasks and all three seeds.

Table 10: Full-fine-tuning learning-rate comparison on Qwen3.5-0.8B. Values are three-seed means $\pm$ sample standard deviations. Cost sums GPU-hours over all six Math and Code runs.

Learning rate	GSM8K ↑	HumanEval p@1 ↑	Cost (GPU-h)
10^{-5}	61.031 ± 1.070	22.860 ± 0.618	27.79
10^{-4}	54.107 ± 0.835	22.701 ± 0.888	40.71

Table 11: Recorded training wall time for the dense Qwen3.5-0.8B study. Math and Code times are three-run means.

Method	Execution topology	Math wall time (h)	Code wall time (h)
Full-FT, LR 10^{-5}	2 H100, batch 8/GPU, accumulation 2	1.9712	2.6602
Full-FT, LR 10^{-4}	1 H100, batch 8, accumulation 4	5.0420	8.5271
Scrambled MPO	1 H100, batch 8, accumulation 4	10.5452	12.0450
MPO-PRO	2 H100, batch 8/GPU, accumulation 2	2.3502	3.2003

In the two-GPU runs, each H100 processes two microbatches of eight examples per optimizer step; the single-GPU runs process four microbatches sequentially. All configurations use an effective batch size of 32 and 3,125 steps. Thus the full-fine-tuning rows in Table 11 differ in both learning rate and GPU configuration. MPO-PRO shares the two-GPU configuration of the baseline at 10^{-5}. Scrambled MPO trains 2.016B temporary parameters on one GPU, with matrix reconstruction and permutations. The wall times capture these differences in parameterization and execution.

Recorded training GPU-hours are the trainer-reported runtime in hours multiplied by the number of GPUs per run, summed over the six runs. This accounts for device count while retaining differences in execution efficiency; it excludes benchmark evaluation and operations outside the trainer interval.

GSM8K evaluation covers all 1,319 questions with greedy decoding. HumanEval evaluation covers 164 tasks with 200 samples per task, batch size 96, temperature 0.8, nucleus-sampling threshold $p = 0.95$, and a shared decoding seed base. Evaluation uses the merged checkpoints after contraction. Table 12 gives the per-seed scores underlying the 10^{-5} full-fine-tuning and MPO-PRO rows in Table 1.

Table 12: Per-seed results for full fine-tuning at LR 10^{-5} and MPO-PRO underlying Table 1. Values are percentages.

Task	Method	Seed 0	Seed 1042	Seed 3047
GSM8K	Full-FT	62.017	61.183	59.894
GSM8K	MPO-PRO	63.988	63.078	64.367
HumanEval p@1	Full-FT	23.503	22.805	22.271
HumanEval p@1	MPO-PRO	25.753	25.558	25.140

G DENSE OBJECTIVE AND DEPLOYED-UPDATE DIAGNOSTICS

Let $\Delta W = W_{\text{final}} - W_0$ be the deployed weight update. Its stable rank is $\|\Delta W\|_F^2 / \|\Delta W\|_2^2$, where $\|\cdot\|_F$ is the Frobenius norm and $\|\cdot\|_2$ is the matrix spectral norm, equal to the largest singular value. In the illustrated layer, stable rank is 35.59 for MPO-PRO. MPO-PRO attains a final validation loss of 0.1254.

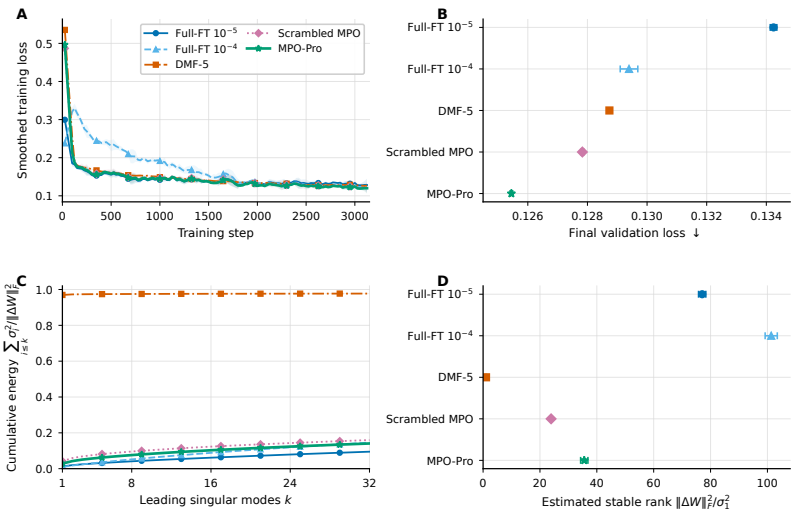


Figure 4: **Training parameterizations affect loss and the final dense update.** (A) Three-seed MetaMathQA trajectories. (B) Final loss on the held-out validation split. (C) Cumulative singular energy and (D) stable rank of the deployed BF16 update in layer 3 `mlp.up_proj`.

Figure 5 expands panels A–B of Figure 4. For each seed, training-loss values are linearly interpolated onto a common step grid spanning steps 25–3,125 and smoothed with a centered five-point moving mean. Bands and endpoint bars show sample standard deviations across three seeds.

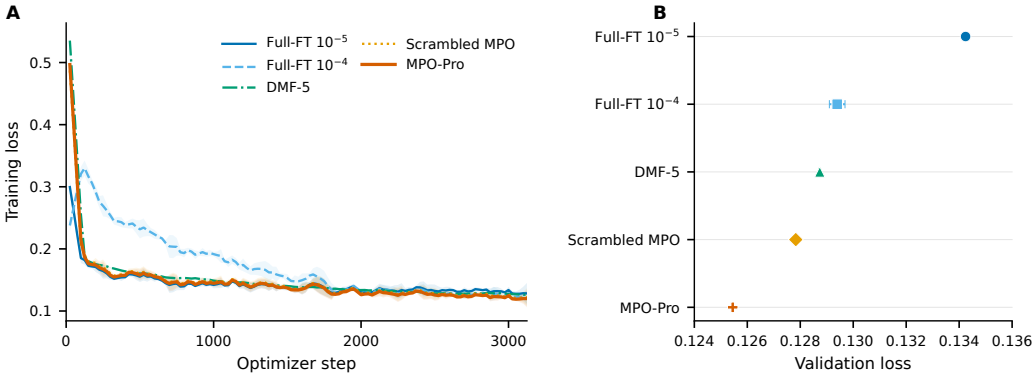


Figure 5: **Training and validation loss for dense Qwen3.5-0.8B post-training.** (A) MetaMathQA training loss across one epoch. (B) Final loss on the held-out validation split. MPO-PRO reaches the lowest final validation loss among the four configurations, all of which export the same model architecture.

Figure 6 provides additional spectral details for the deployed updates in Figure 4C–D. All quantities are computed after MPO contraction or factor merging from the deployed BF16 update $\Delta W = W_{\text{final}} - W_0$.

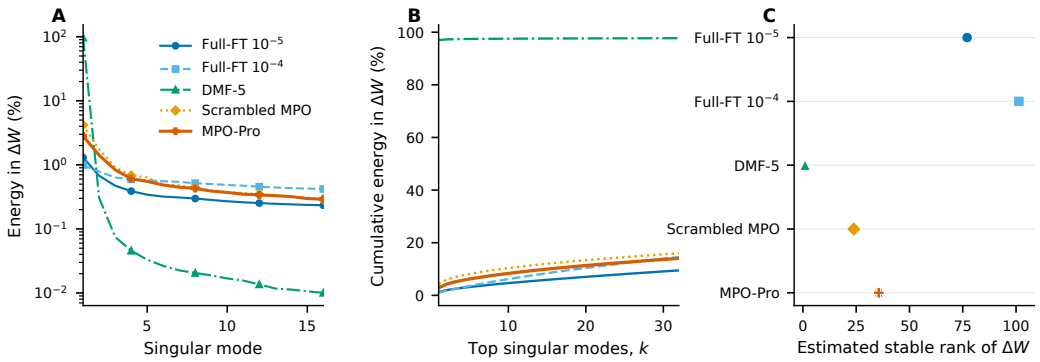


Figure 6: **Deployed-update spectra under four training configurations.** (A) Energy assigned to each of the leading 16 singular modes on a logarithmic scale. (B) Cumulative energy in the leading 32 modes. (C) Estimated stable rank. The analyzed weight matrix is layer 3 `mlp.up_proj`; curves, points, and error bars summarize seeds 0, 1042, and 3047.

H Deployment Equivalence

The dense branch adds the reconstructed update to W_0 and saves an ordinary checkpoint. The fixed-rank branch contracts the MPO cores into an $r \times n$ matrix A and retains the standard adapter (B, A). Both exports preserve the deployed matrix shapes and remove the temporary cores. Reconstruction and merging reproduce the training representation to floating-point precision. All reported dense evaluations use the merged checkpoint.