A COORDINATE VIEW OF LORA: FIXED BASES AND LEARNED COMBINATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Adapting pretrained models to many tasks makes it increasingly valuable to store task-specific capabilities with fewer parameters. LoRA and fixed-basis variants reduce this storage cost, but parameter counts alone do not explain what enables adaptation or how the stored parameters affect task performance. We study these questions through a coordinate view, $\Delta W = ANB$: shared fixed bases A, B define the available updates, while task-specific coordinates N represent each learned update. This view lets us examine basis information and scale, coordinate connectivity, and the task directions along which coordinates can be changed. Normalized random bases closely approach pretrained-SVD performance, while learning-rate calibration substantially recovers raw Gaussian performance. At comparable coordinate budgets, wider sparse systems can outperform narrower dense ones, with connectivity effects also depending on update scale. Scaling task residuals forward or backward produces structured changes in accuracy; a single few-shot gradient also identifies beneficial directions from weak starting states. Across six tasks, the largest 10% of a 64-example direction's coordinates retain approximately 93% of its accuracy gain from zero and randomized starts; relocating the same values largely removes the benefit. These findings connect compact task-specific storage with interpretable, direction- and location-dependent changes in task performance.

1 INTRODUCTION

The growing demand to adapt pretrained models to different tasks makes it important to retain task-specific capabilities with fewer parameters. Storing a full model for every task is costly, whereas sharing pretrained weights and storing only the task-specific changes reduces this burden. Low-Rank Adaptation (LoRA) offers a practical way to do so: it freezes the pretrained weights and learns two low-rank factors (Hu et al., 2022). This makes parameter-efficient adaptation valuable not only for training, but also for maintaining many task-specific models on a shared backbone.

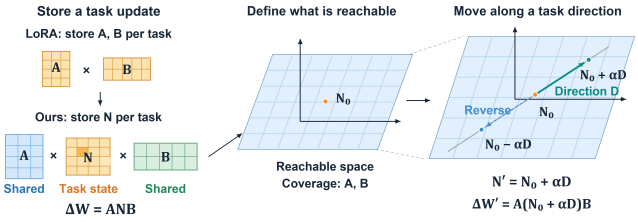


Figure 1: Share bases A, B, store task coordinates N, and probe signed changes within the fixed space.

Subsequent work further reduces or reorganizes the learned parameters through two choices of adaptation directions. *Weight-informed methods* exploit pretrained structure: PiSSA initializes trainable factors from singular components, whereas LoRA-XS and SVFT freeze singular directions and learn their coefficients (Meng et al., 2024; Bałazy et al., 2024; Lingam et al., 2024). *Random-basis methods* dispense with that information: VeRA and NOLA learn scaling vectors or combination coefficients over frozen random matrices or components (Kopiczko et al., 2024; Koohpayegani et al., 2024). These results show that fixed directions can support adaptation with few learned coefficients. However, comparisons across designs often vary basis information, scale, and coefficient counts together. They also leave open how changing those coefficients affects task performance and whether a few examples can identify useful changes.

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.

We take a coordinate view of adaptation to examine these questions (Figure 1), rather than propose a new adapter. In $\Delta W = ANB$, shared frozen bases A, B define reachable updates; only the coordinates N are learned and stored per task. Learning changes N, while $N \mapsto N + \alpha D$ moves it along a direction with controlled sign and strength. At $N = 0$, the pretrained function is unchanged. Building on fixed-factor adaptation (Bałazy et al., 2024; Lingam et al., 2024; Guo et al., 2025), we study what the shared space provides and how the stored state affects performance. Interpretability here means task-level responses to controlled edits, not individual coordinates with distinct semantic meanings.

This view separates coverage from combination. Bases supply input and output directions; allowed entries of N select interacting pairs, and their values set interaction strengths and signs. We vary basis source, width, and connectivity separately, controlling update scale where needed. Our reachable-space characterization makes this distinction precise: with linearly independent basis directions, the allowed coordinates determine dimension, and their connection pattern determines maximum rank. Holding this space fixed then isolates the effects of coordinate edits.

The investigation follows three questions (Figure 2): *Does the basis need to know the model? How much coordinate connectivity is needed? What structure do task directions reveal?* The last question distinguishes the existence of an effective direction from its discovery: trained-target residuals probe signed movement, whereas few-shot gradients test whether limited supervision identifies improvements on held-out data. Masking and relocation then test where these improvements are concentrated.

Our main findings are:

- **Basis information.** Normalized random bases approach LoRA performance without extracting singular directions from pretrained weights. Normalization is important: raw Gaussian bases perform poorly at the common learning rate but recover at smaller rates, highlighting the role of basis scale.
- **Coordinate sparsity.** Effective adaptation does not require dense connections between basis directions. At comparable parameter budgets, a wider basis with sparse connections can outperform a narrower dense basis; adding connections is not uniformly beneficial, and its effect also depends on update scale.
- **Task directions.** Coordinate space supports interpretable task-level interventions: moving toward a trained target improves accuracy from weak starts, while reversing direction generally reduces it. Few-shot gradients can also identify useful directions on held-out data. Their effect is concentrated at specific locations: from zero and randomized starts, the top 10% of a 64-example direction retains about 93% of its gain, whereas relocating the same values largely removes the benefit.

2 Related Work

Adaptation subspaces. Adaptation methods differ in how they obtain update directions. LoRA learns its factors, PiSSA initializes them from pretrained singular components, and LoRA-XS and SVFT learn coefficients over frozen singular directions (Hu et al., 2022; Meng et al., 2024; Bałazy et al., 2024; Lingam et al., 2024). NoRA factorizes these coefficients, while NLoRA/IntTune and QR-LoRA use weight-informed decompositions (Lin et al., 2024; Guo et al., 2025; Liang & Bharadwaj, 2025). VeRA, NOLA, and RandLoRA instead learn scalings or combinations of random factors (Kopiczko et al., 2024; Koohpayegani et al., 2024; Albert et al., 2025). LoRA-XS also compares random and SVD bases. These results show that learning new outer factors is not the only way to adapt a model. Comparing different parameterizations, however, can change both the available directions and the number or scale of their learned coefficients. Building on these studies, we use the same ANB form to compare basis sources at matched width and support, with normalization and learning-rate controls to examine scale.

Parameter composition. Model soups, TIES, and DARE merge trained weights, while LoraHub and modular LoRA libraries combine or route adapters (Wortsman et al., 2022; Yadav et al., 2023; Yu et al., 2024; Huang et al., 2023; Ostapenko et al., 2024). Task Singular Vectors and Core Space use update structure to reduce merging interference (Gargiulo et al., 2025; Panariello et al., 2026).

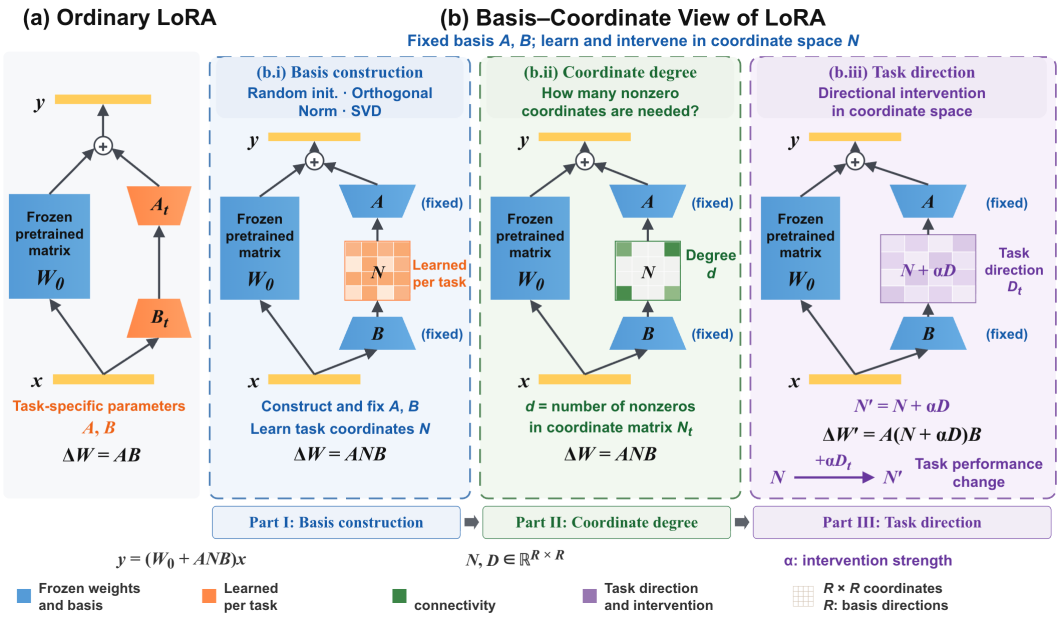


Figure 2: A basis–coordinate view of LoRA. Ordinary LoRA learns task-specific low-rank factors, whereas we fix a shared basis A, B and learn only the coordinate matrix N. This view separates three questions: the source of the basis, the required coordinate connectivity, and the amount of task-specific coordinate editing.

These approaches reuse information from trained models or adapters, making the choice and interaction of the combined components central to their design. At a finer level, SVFT combines fixed singular-vector pairs and studies sparse support (Lingam et al., 2024). Our analysis concerns this coefficient-level composition, where N_{ij} weights $a_i b_j^\top$. Here sparsity limits which basis pairs interact, rather than which trained adapters participate in a merge. At matched nominal Rd, we vary width and degree to study how a parameter budget can be allocated between basis directions and their interactions, while accounting for update scale.

Parameter editing. ROME and MEMIT edit facts; WISE and ELDER address repeated edits (Meng et al., 2022; 2023; Wang et al., 2024; Li et al., 2025). LoRA memory studies and Deep Context Distillation store knowledge in adapters; Text-to-LoRA, Doc-to-LoRA, and SHINE generate parameters from descriptions or context (Back et al., 2026; Caccia et al., 2025; Charakorn et al., 2025; 2026; Liu et al., 2026). These studies connect parameter changes to desired behavior, but differ in whether the change is specified by an edit, learned from examples, or generated from context. Our focus is task-level performance under coordinate changes. Signed residual scaling probes directional effects; masking and relocation probe coordinate locations. These interventions operate on an existing direction, whereas SVFT and LoRASculpt impose sparsity during training (Lingam et al., 2024; Liang et al., 2025). A separate few-shot gradient experiment asks whether useful directions can be found from limited examples and improve held-out accuracy. This separates examining a known direction's effect from discovering a direction without the trained target.

3 Method

We first define a fixed basis and its learned coordinates, then characterize the updates permitted by their connectivity, and finally construct and edit task directions within this space.

3.1 Fixed-Basis Coordinate Adaptation

For a pretrained linear layer $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, we write

$$W = W_0 + sANB, \qquad A \in \mathbb{R}^{d_{\text{out}} \times R}, \quad N \in \mathbb{R}^{R \times R}, \quad B \in \mathbb{R}^{R \times d_{\text{in}}}, \tag{1}$$

where W_0, A, B remain frozen and only N is learned; $s \neq 0$ is a fixed multiplier. Column a_i of A and row $b_j^\top$ of B specify an output and an input direction. A support $\mathcal{S} \subseteq [R] \times [R]$ selects which pairs can contribute:

$$\Delta W = s \sum_{(i,j)\in\mathcal{S}} N_{ij} a_i b_j^\top, \qquad N_{ij} = 0 \quad \text{for } (i,j) \notin \mathcal{S}. \tag{2}$$

Thus each N_{ij} weights the fixed rank-one atom $a_i b_j^\top$. The support denotes allowed locations, not the currently nonzero entries; an allowed coordinate remains trainable even when its value is zero.

For a multilayer model, N also denotes the collection of these matrices, and f_N the resulting model. Norms and masks act on the P concatenated allowed coefficients, which we call active coordinates. Task training minimizes $\mathcal{L}(N) = |\mathcal{D}|^{-1} \sum_{(x,y)\in\mathcal{D}} \ell(f_N(x), y)$ from $N = 0$, preserving the pretrained function at initialization. Let $N^\star$ denote the trained target state.

Constructing the basis. Our default basis uses thin QR decompositions of independent Gaussian matrices:

$$G_A = Q_A T_A, \qquad G_B^\top = Q_B T_B, \qquad A = Q_A, \quad B = Q_B^\top. \tag{3}$$

Here T_A, T_B are upper triangular, and the resulting factors satisfy $A^\top A = BB^\top = I$ without using pretrained weights or task data. Three alternatives vary the source or geometry of these factors: Raw Gaussian sets $A = G_A, B = G_B$; Unit Gaussian normalizes each column of G_A and row of G_B; and Pretrained-SVD uses $A = U_{:R}, B = V_{:R}^\top$ from $W_0 = U\Sigma V^\top$, retaining the leading singular vectors but not their singular values.

3.2 Connectivity and Reachable Updates

Width R controls *coverage*, the number of available input and output directions. The support controls *connectivity*, the permitted pairs of those directions. A row-degree-d support gives each output direction d distinct input neighbors, hence $|\mathcal{S}| = Rd$ for $d \leq R$. Dense connectivity corresponds to $d = R$. For effective layer width $R_\ell = \min(R, d_{\text{out},\ell}, d_{\text{in},\ell})$, the total coordinate count is $P = \sum_\ell R_\ell \min(d, R_\ell)$.

These choices determine both the dimension of the reachable update space and its maximum matrix rank, but the two need not grow together. The structural-rank identity makes this distinction precise.

Theorem 1 (Reachable space and structural rank)**.** *Let $A \in \mathbb{R}^{m\times R}$ have full column rank, $B \in \mathbb{R}^{R\times n}$ have full row rank, and $s \neq 0$. Define $\mathcal{C}_\mathcal{S} = \{N \in \mathbb{R}^{R\times R} : N_{ij} = 0 \text{ outside } \mathcal{S}\}$ and $\mathcal{U}_\mathcal{S} = \{sANB : N \in \mathcal{C}_\mathcal{S}\}$. Let $\nu(\mathcal{S})$ be the maximum matching size of the bipartite graph whose edges are the allowed coordinate pairs. For $N \in \mathcal{C}_\mathcal{S}$,*

$$\dim \mathcal{U}_\mathcal{S} = |\mathcal{S}|, \qquad \operatorname{rank}(sANB) = \operatorname{rank}(N), \qquad \max_{\Delta W \in \mathcal{U}_\mathcal{S}} \operatorname{rank}(\Delta W) = \nu(\mathcal{S}). \tag{4}$$

Almost every assignment of the allowed coordinates attains this maximum rank.

A perfect matching permits rank-R updates with only R coordinates, whereas dense connectivity has R^2 degrees of freedom. Thus coordinate count and attainable rank are distinct; neither alone guarantees task accuracy.

Under the same full-rank assumptions, dense N represents a matrix M exactly when its column and row spaces lie in those of A and B. Its unique coordinates are $s^{-1}A^\dagger M B^\dagger$, where $\dagger$ denotes the Moore–Penrose pseudoinverse. A sparse support further requires these coordinates to vanish outside $\mathcal{S}$. Appendix A proves the theorem and this characterization. Exact reconstruction of a particular LoRA update is therefore a separate question from matching its task performance.

3.3 Direction Discovery and Coordinate Editing

Having specified the reachable space, we vary the direction, scale, and locations of a coordinate change while keeping $A, B, \mathcal{S}$, and s fixed.

Starting states and known task directions. Write an intervention as $N(\gamma) = N_0 + \gamma \mathbf{v}$, where N_0 is the starting state, $\mathbf{v}$ a direction on the active coordinates, and γ its signed scale. Our main interventions start from zero or randomized coordinates; random-state constructions and a shared-state control are specified in Appendix C.3.

A trained target defines the residual direction

$$\mathbf{v}^\star = N^\star - N_0, \qquad N^{\mathrm{oracle}}(\alpha) = N_0 + \alpha \mathbf{v}^\star. \tag{5}$$

The endpoints $\alpha = 0, 1$ recover the start and target. Negative values reverse the residual, and values above one extrapolate beyond the target. This oracle intervention probes a known task direction; it requires the trained target rather than discovering a direction from new examples.

Discovering a direction from few examples. To discover a direction without $N^\star$, we use m target training examples $\mathcal{D}_m$. For multiple-choice input x with candidate set $\mathcal{C}(x)$ and correct choice y, let $z_c(x; N)$ be the summed continuation log-probability of candidate c. The discovery loss is

$$\mathcal{L}_m(N) = -\frac{1}{m} \sum_{(x,y)\in\mathcal{D}_m} \log \frac{\exp z_y(x; N)}{\sum_{c\in\mathcal{C}(x)} \exp z_c(x; N)}. \tag{6}$$

For each start and sample subset, compute the active-coordinate gradient $\mathbf{g}_m = \nabla_N \mathcal{L}_m(N_0)$ and normalize it as $\mathbf{u}_m = -\mathbf{g}_m / \|\mathbf{g}_m\|_2$, setting $\mathbf{u}_m = 0$ if it vanishes. We evaluate $N^{\mathrm{few}}(\eta) = N_0 + \eta \mathbf{u}_m$ with this direction fixed, without iterative training. For nonzero gradients, $|\eta|$ is the displacement norm and its sign selects forward or reverse motion; oracle α instead scales the unnormalized residual.

Localizing the edit. To identify which locations carry a direction's effect, let $\mathcal{I}_k(\mathbf{u})$ index its k largest $|u_j|$ among the P active coordinates, and let $\mathbf{m}_k$ be the corresponding mask. Keeping a fraction $q = k/P$ gives $N^{\mathrm{edit}}(\eta, k) = N_0 + \eta(\mathbf{m}_k \odot \mathbf{u})$, so $\|N^{\mathrm{edit}} - N_0\|_0 \le k$ and other coordinates remain at N_0. Selected values retain their signs and magnitudes without reoptimization or renormalization. A location control randomly relocates the same values within the allowed support, preserving their multiset, sparsity, and norm. The next result links these edits to projection and sparse approximation.

Theorem 2 (Projected task directions and sparse edits)**.** *Assume differentiable $\mathcal{L}_m$, $A^\top A = BB^\top = I$, and $s \neq 0$. Write $\mathbf{g} = \mathbf{g}_m$, $\mathbf{u} = \mathbf{u}_m$, $G = \|\mathbf{g}\|_2 > 0$, and $H = \nabla_W \mathcal{L}_m$ at N_0. With $\Pi_{\mathcal{U}_S}$ the Frobenius-orthogonal projector, layerwise zero-filling of $\mathbf{g}$ gives*

$$sA(-\mathbf{g})B = -s^2 \Pi_{\mathcal{U}_S} H. \tag{7}$$

For a mask $I \subseteq [P]$, write $\mathbf{u}_I = \mathbf{1}_I \odot \mathbf{u}$ and $\rho_I = \|\mathbf{g}_I\|_2^2 / G^2$. Then

$$\|\mathbf{u} - \mathbf{u}_I\|_2^2 = 1 - \rho_I, \qquad \frac{d}{d\eta} \mathcal{L}_m(N_0 + \eta \mathbf{u}_I)\Big|_{\eta=0} = -G\rho_I. \tag{8}$$

Thus top-k is a best k-sparse Euclidean approximation and maximizes first-order descent among size-k masks at a common positive scale. Reversing $\mathbf{u}_I$ gives first-order ascent when $\rho_I > 0$.

The proof and finite-step bound are in Appendix B. This local loss result motivates direction selection; held-out accuracy and large-scale signed responses remain empirical questions.

4 Experiments

Our experiments test three sources of freedom in the task state N: information in the fixed basis, connectivity among its directions, and the coordinate changes needed for specialization. Each part uses controlled interventions to distinguish a representational choice from its optimization or task-level effect.

Experimental setup. We evaluate Qwen2.5-3B and Qwen3.5-2B on six multiple-choice tasks: PIQA, ARC-Easy, ARC-Challenge, OpenBookQA, HellaSwag, and WinoGrande. Each task has a separately trained adapter; we report accuracy and the unweighted task average. Base and

Table 1: **Adaptation with fixed bases.** Six-task accuracy (%) at the common learning rate. Params denotes task-trainable parameters; % LoRA is relative to LoRA-$r16$ on the same backbone. Best results within each model block are bold; the top three, including ties, are underlined. HellaSwag uses 3,000 evaluation examples.

Method	Source	Norm.	Ortho.	Params (M)	% LoRA	PIQA	ARC-E	ARC-C	OBQA	Hella.	Wino.	Avg.
Qwen2.5-3B: global $R = 256, d = 32$												
Base	–	–	–	0.00	0.00	78.29	84.01	67.66	65.20	44.10	56.83	66.01
LoRA-$r16$	Random	×	×	29.93	100.00	**85.04**	**94.49**	82.68	**88.40**	**88.73**	**76.48**	**85.97**
Raw Gaussian	Random	×	×	2.06	6.90	53.65	29.63	19.03	14.80	25.73	50.36	32.20
Unit Gaussian	Random	✓	×	2.06	6.90	84.17	**94.49**	82.34	86.00	87.20	73.09	84.55
QR Random	Random	✓	✓	2.06	6.90	84.39	94.44	**82.94**	86.20	87.13	72.61	84.62
SVD(W_0)	W_0 Decomp.	✓	✓	2.06	6.90	84.49	94.32	82.68	86.00	87.50	73.16	84.69
Qwen3.5-2B: global $R = 256, d = 32$												
Base	–	–	–	0.00	0.00	64.58	71.68	56.91	46.00	28.63	52.17	53.33
LoRA-$r16$	Random	×	×	16.82	100.00	**80.85**	**94.07**	83.28	**86.00**	**82.90**	**70.24**	**82.89**
Raw Gaussian	Random	×	×	1.24	7.36	53.70	29.25	19.88	15.20	26.00	50.36	32.40
Unit Gaussian	Random	✓	×	1.24	7.36	80.69	93.27	83.70	82.60	81.83	68.19	81.71
QR Random	Random	✓	✓	1.24	7.36	80.14	92.89	**83.96**	83.80	81.73	67.64	81.69
SVD(W_0)	W_0 Decomp.	✓	✓	1.24	7.36	80.30	93.52	82.08	82.20	81.67	68.75	81.42

LoRA-$r16$ provide reference points. The basis comparison holds training data, coordinate support, zero initialization, and learning rate 2×10^{-4} fixed within each model. Both models use global $(R, d) = (256, 32)$ coordinates in the main tables. Earlier multi-bank runs are excluded from this matched-configuration comparison. Subsequent mechanism experiments use global QR coordinates on Qwen3.5. We additionally evaluate mathematical reasoning and code generation on GSM8K, MATH, and MBPP. Data sizes, optimization settings, and selection protocols are given in Appendix C.

4.1 Part I: Does the basis need to know the model?

We examine how basis source and normalization affect adaptation, then use learning-rate controls to understand the role of basis scale.

A generic basis can support effective adaptation. Normalized random and pretrained-SVD bases perform similarly in Table 1: their six-task averages span 84.55–84.69% on Qwen2.5 and 81.42–81.71% on Qwen3.5, using 6.90% and 7.36% of LoRA's trainable parameters. LoRA reaches 85.97% and 82.89%, respectively. These results approach LoRA performance while storing only the learned coordinates per task: the outer factors need neither task-specific training nor singular directions extracted from pretrained weights. Unit Gaussian is also competitive with QR, suggesting that strict orthogonality is not necessary in these settings. The common feature is controlled basis scale, whose role we examine through learning-rate sensitivity below.

Coordinate training need not reconstruct LoRA. An independent reconstruction control asks whether a random basis succeeds by approximating the particular update found by LoRA. In a fixed width-16 QR basis, we solve $\min_N \|ANB - \Delta W_{\mathrm{LoRA}}\|_F^2$. The fitted update captures only 0.63% of the LoRA Frobenius norm and reaches 53.62% accuracy, close to Base's 53.33%. Yet task training of N in the same basis reaches 72.74%. Theorem 1 explains the distinction: reconstruction tests alignment with a particular update, whereas task training searches for a useful update in the reachable space. The same basis therefore supports a useful task update despite its poor reconstruction of the LoRA solution.

Extension to reasoning and code. On Qwen2.5, QR and SVD reach 56.64% and 57.55% three-task accuracy, compared with LoRA's 57.42% and Base's 39.58% (Table 2). On Qwen3.5, Unit, QR, and SVD reach 54.43%, 54.69%, and 53.52%. Unit is strongest on MATH, while QR has the highest average on this backbone. The ranking varies across tasks, but normalized random coordinates remain competitive without access to pretrained singular directions. With method-specific optimization settings, fixed coordinates thus support reasoning and code generation as well as multiple-choice adaptation.

Table 2: **Mathematical reasoning and code generation.** Accuracy (%). Fixed-basis methods use global $R = 256, d = 32$ coordinates. Optimization settings are reported in Appendix C.4; this is not a common-LR comparison. Raw Gaussian uses the common rate; calibrated results are discussed separately. Bold marks the best result; underlines mark the top three, including ties.

Method	Source	Norm.	Ortho.	% LoRA	GSM8K	MATH	MBPP	Avg.
Qwen2.5-3B: global $R = 256, d = 32$								
Base	–	–	–	0.00	44.14	25.78	48.83	39.58
LoRA-$r16$	Random	×	×	100.00	**80.08**	37.50	54.69	57.42
Raw Gaussian	Random	×	×	6.90	0.00	0.00	0.00	0.00
Unit Gaussian	Random	✓	×	6.90	72.66	34.77	**55.08**	54.17
QR Random	Random	✓	✓	6.90	76.95	38.67	54.30	56.64
SVD(W_0)	W_0 Decomp.	✓	✓	6.90	77.73	**40.62**	54.30	**57.55**
Qwen3.5-2B: global $R = 256, d = 32$								
Base	–	–	–	0.00	72.66	43.36	38.28	51.43
LoRA-$r16$	Random	×	×	100.00	74.61	36.72	**42.58**	51.30
Raw Gaussian	Random	×	×	7.36	1.17	2.34	0.00	1.17
Unit Gaussian	Random	✓	×	7.36	72.66	**49.22**	41.41	54.43
QR Random	Random	✓	✓	7.36	**75.78**	46.48	41.80	**54.69**
SVD(W_0)	W_0 Decomp.	✓	✓	7.36	73.05	46.09	41.41	53.52

Raw bases require scale calibration. On the seven-point dev grid, Raw reaches 82.42% on Qwen2.5 at 10^{-7} and 74.87% on Qwen3.5 at 2×10^{-6}, but degrades sharply near the common rate 2×10^{-4}. Unit and QR remain effective at 10^{-4}–2×10^{-4}. Large basis norms amplify coordinate changes in weight space, so equal learning rates need not give comparable updates. Raw's recovery shows that useful directions can be obscured by an unsuitable scale; normalization shifts the effective learning-rate range. The Qwen3.5 control uses archived multi-bank coordinates (Appendix C). Stars mark dev maxima; the dotted line marks the common rate.

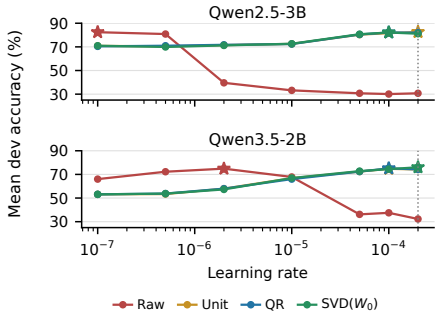


Figure 3: Learning-rate sensitivity on six-task dev accuracy.

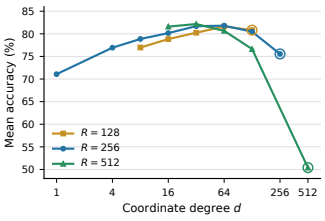
Scale effects also appear in generation. Complementing the dev sensitivity in Figure 3, calibrated Raw reaches three-task averages of 54.04% on Qwen2.5 at 5×10^{-7} and 49.09% on Qwen3.5 at 10^{-7}. These are separate calibrated runs, not the common-rate Raw entries in Table 2. Smaller learning rates also restore useful adaptation in generation. Together with the six-task dev curves, this shows that basis scale affects the useful learning-rate range across both evaluation settings.

4.2 Part II: How much coordinate connectivity is needed?

With a usable basis fixed, adaptation capacity can be spent on exposing more directions or connecting existing directions more densely. We separate these choices by varying coverage R and degree d. Theorem 1 distinguishes their effects on expressivity: coordinate count determines update-space dimension, whereas attainable rank also depends on the support's matching structure. The experiments below measure the task-level consequences of these allocations.

Coverage and connectivity are not interchangeable. At the same nominal edge budget $Rd = 16{,}384$, widening the basis from $(128, 128)$ to $(512, 32)$ increases accuracy from 80.81% to 82.16% (Figure 4, right). At $Rd = 4{,}096$, the corresponding configurations span 79.98–80.32%, while the $Rd = 8{,}192$ group is nearly flat at 81.60–81.69%. A broader, sparsely connected system can therefore outperform a narrower dense one, but the benefit depends on the budget. The nominal budget refers to full-width maps; exact storage additionally accounts for dimensional caps on narrow projections.

More edges do not always help. Figure 4, left, shows the degree response at three widths. For $R = 256$, accuracy rises from 71.08% at $d = 1$ to 81.80% at $d = 64$, then drops to 75.55% at full connectivity. The dense drop is even larger at $R = 512$. These curves reveal an effective intermediate connectivity regime under the fixed optimization protocol, rather than a monotonic benefit from additional coordinates.



Fixed budgets

Rd	R	d	Acc. (%)
4,096	64	64	79.98
	128	32	80.24
	256	16	80.16
	512	8	**80.32**
8,192	128	64	81.63
	256	32	**81.69**
	512	16	81.60
16,384	128	128	80.81
	256	64	81.80
	512	32	**82.16**

Figure 4: **Coverage and connectivity.** Six-task accuracy on Qwen3.5. Rings mark dense endpoints; bold marks the best allocation per budget.

The dense penalty is partly an optimization effect. Increasing degree also changes the aggregate update scale. To control for this, we multiply the adapter scale by $\sqrt{32/d}$ and repeat $d \in \{32, 64, 128, 256\}$ with two seeds on ARC-Challenge, HellaSwag, and WinoGrande. Mean accuracy becomes 78.12%, 78.43%, 78.34%, and 77.79%, respectively: dense coordinates remain competitive after rescaling. The fixed-scale drop should therefore not be interpreted as an intrinsic expressivity disadvantage. The consistent conclusion is that dense interaction is unnecessary in some regimes, while its optimization depends on both coverage and scale.

4.3 Part III: From task directions to coordinate edits

We next ask whether the useful changes in N have identifiable structure. All interventions hold the QR basis and support fixed at $R = 256, d = 32$. Starting states are zero, randomized coordinates, or a leave-one-task-out average N_{shared} of the other five task states. We measure the response along known task residuals, ask whether a few target examples reveal useful directions, and test which coordinates carry their effect.

Known task directions exhibit a signed dose response. We use the trained target $N^\star$ to define a residual direction and evaluate 13 scales from -1 to 2. Moving toward the target improves accuracy from weak starts across the six tasks (Figure 5), whereas reversing the direction generally reduces it. The sign of the coordinate change therefore distinguishes improvement from degradation along the same path. Beyond the target, accuracy saturates or declines depending on the task, showing how intervention strength shapes the response. These oracle directions provide a reference for the few-shot directions examined next.

A few target examples reveal a useful local direction. Without access to the trained target, a single gradient from 8–128 training examples identifies a direction that improves held-out accuracy (Figure 6). At $\eta = 0.5$, randomized-start mean accuracy rises from 53.86% to 61.99% with 8 examples and 67.25% with 128. Thus, additional examples improve the usefulness of the estimated direction at the same displacement scale. With 64 examples, positive and negative doses yield 66.78% and 37.91%, respectively: following the direction improves performance, while reversing it removes the benefit and lowers accuracy below the starting level. This signed response extends to held-out data, demonstrating local generalization from limited supervision. Figure 6 reports all tested doses; unlike oracle α, $|\eta|$ denotes the coordinate displacement norm.

Coordinate locations matter. We next examine where the few-shot direction carries its effect by retaining its largest absolute entries at $\eta = 0.5$, without renormalization. For the 64-example direction, top-10% preserves approximately 93% of the full-direction gain from zero and randomized starts (Table 3). Increasing support to 20–40% closes most of the remaining gap. Larger supports then saturate near the full result, with small non-monotonic fluctuations: most of the improvement is already provided by a small subset of coordinates. Relocating the same signed values preserves count and norm but leaves accuracy near the starting level. The contrast shows that the benefit depends on both the retained values and their specific coordinate locations. Starting-state dependence and per-task curves appear in Appendix C.3.

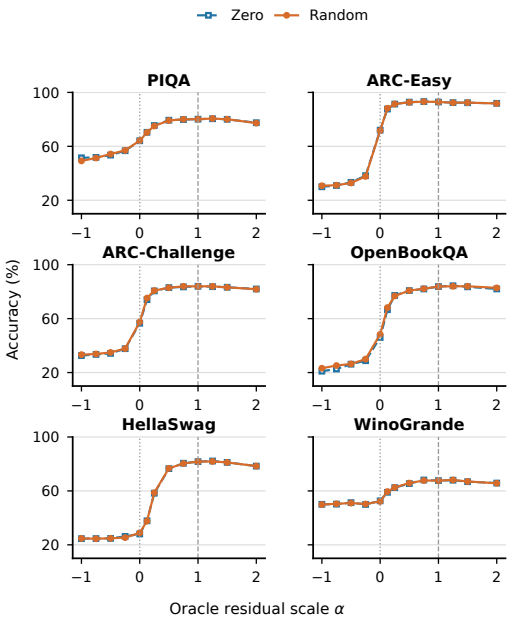


Figure 5: **Oracle task directions.** Accuracy under signed residual scaling from zero and randomized starts. $\alpha = 0$ recovers the starting state and $\alpha = 1$ the trained target. Shared-start results appear in Appendix C.3.

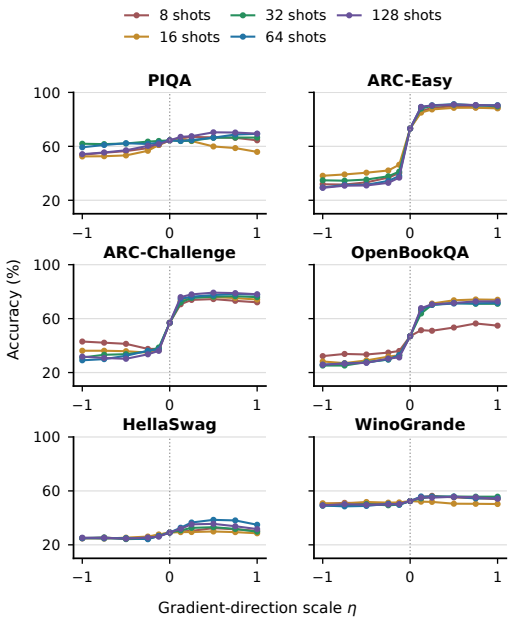
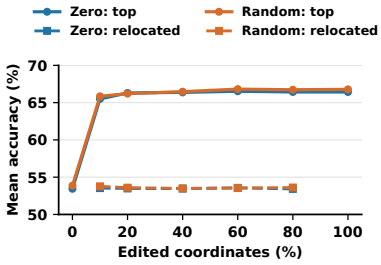


Figure 6: **Few-shot directions.** Randomized-start accuracy for 8–128 examples. Positive η follows a single negative gradient; negative η reverses it. No iterative fine-tuning is used; all tested doses are shown.



	Zero		Random	
Retained	Top	Reloc.	Top	Reloc.
Start	53.44	53.44	53.86	53.86
10%	65.50	53.53	65.85	53.77
20%	66.30	53.49	66.21	53.59
40%	66.37	53.46	66.47	53.50
60%	66.53	53.57	66.81	53.54
80%	66.43	53.42	66.73	53.61
100%	66.43	66.43†	66.78	66.78†

Table 3: **Coordinate support.** Six-task accuracy (%) for 64-shot directions at $\eta = 0.5$. Curves and table compare top-magnitude edits with relocation of the same values (three-run mean). Relocation is evaluated at 10–80%; † repeats the full original direction for reference, not a measured full relocation.

5 Conclusion

We studied low-rank adaptation through a coordinate view: shared fixed bases define the available updates, while learned coordinates store the task-specific changes. Normalized random bases approach LoRA performance without extracting pretrained singular directions, and learning-rate controls show why basis scale matters. Sparse coordinate connections can support effective adaptation; at comparable budgets, broader coverage can be more useful than dense interaction. Within this space, signed edits reveal task directions, few-shot gradients identify useful changes on held-out data, and relocation controls show that coordinate locations matter. These effects depend on the starting state and optimization scale. Together, the results connect storing fewer task-specific parameters with understanding how those parameters affect performance. The contribution is an analytical perspective on adaptation, not a new adapter architecture.

AI USE STATEMENT

Generative AI assisted with writing and polishing, and with retrieval and discovery of related literature. The authors checked cited sources and reviewed the resulting text. AI tools also assisted with analysis and figure code. For the reasoning suite, GPT-generated candidate training solutions were retained only after verification. These synthetic solutions are part of the training data, not evaluation labels. The authors remain responsible for all claims, experimental results, and code.

REFERENCES

Paul Albert, Frederic Z. Zhang, Hemanth Saratchandran, Cristian Rodriguez-Opazo, Anton van den Hengel, and Ehsan Abbasnejad. Randlora: Full-rank parameter-efficient fine-tuning of large models. In *International Conference on Learning Representations*, 2025.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, SK Hong, Youngjune Gwon, and Sungjin Ahn. Understanding lora as knowledge memory: An empirical analysis. *arXiv preprint arXiv:2603.01097*, 2026.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. In *ECAI 2025: 28th European Conference on Artificial Intelligence 25-30 October 2025, Bologna, Italy-Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025) Proceedings*, pp. 3194–3201. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 2024.

Lucas Caccia, Alan Ansell, Ivan Vulić, Edoardo Ponti, and Alessandro Sordoni. Training plug n'play knowledge modules with deep context distillation. In *ICLR 2025 Workshop on Modularity for Collaborative, Decentralized, and Continual Deep Learning*, 2025.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption. *arXiv preprint arXiv:2506.06105*, 2025.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-lora: Learning to instantly internalize contexts. *arXiv preprint arXiv:2602.15902*, 2026.

Antonio Andrea Gargiulo, Donato Crisostomi, Maria Sofia Bucarelli, Simone Scardapane, Fabrizio Silvestri, and Emanuele Rodola. Task singular vectors: Reducing task interference in model merging. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 18695–18705, 2025.

Chenlu Guo, Yi Chang, and Yuan Wu. Nlora: Nyström-initiated low-rank adaptation for large language models. In *EMNLP (Findings)*, pp. 1371–1385, 2025.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 1(2):3, 2022.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. Lorahub: Efficient cross-task generalization via dynamic lora composition. In *First Conference on Language Modeling*, 2023.

Soroush Abbasi Koohpayegani, K. L. Navaneet, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. Nola: Compressing lora using linear combination of random basis. In *International Conference on Learning Representations*, 2024.

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.

Jiaang Li, Quan Wang, Zhongnan Wang, Yongdong Zhang, and Zhendong Mao. Elder: Enhancing lifelong model editing with mixture-of-lora. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24440–24448, 2025.

Jessica Liang and Anirudh Bharadwaj. Qr-lora: Qr-based low-rank adaptation for efficient fine-tuning of large language models. *arXiv preprint arXiv:2508.21810*, 2025.

Jian Liang, Wenke Huang, Guancheng Wan, Qu Yang, and Mang Ye. Lorasculpt: Sculpting lora for harmonizing general and specialized knowledge in multimodal large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 26170–26180, 2025.

Cheng Lin, Lujun Li, Dezhi Li, Jie Zou, Wei Xue, and Yike Guo. Nora: Nested low-rank adaptation for efficient fine-tuning large models. *arXiv preprint arXiv:2408.10280*, 2024.

Vijay Lingam, Atula Tejaswi, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Alex Dimakis, Eunsol Choi, Aleksandar Bojchevski, and Sujay Sanghavi. Svft: Parameter-efficient fine-tuning with singular vectors. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. *arXiv preprint arXiv:2602.06358*, 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.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023.

Oleksiy Ostapenko, Zhan Su, Edoardo Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards modular llms by building and reusing a library of loras. In *International Conference on Machine Learning*, pp. 38885–38904. PMLR, 2024.

Aniello Panariello, Daniel Marczak, Simone Magistri, Angelo Porrello, Bartłomiej Twardowski, Andrew Bagdanov, Simone Calderara, and Joost van de Weijer. Accurate and efficient low-rank model merging in core space. *Advances in Neural Information Processing Systems*, 38:61793–61825, 2026.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797, 2024.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. PMLR, 2022.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

A Reachable Space and Structural Rank

Proof of Theorem 1. **1. Coordinate dimension.** Full column rank of A and full row rank of B give

$$L = (A^\top A)^{-1}A^\top, \qquad C = B^\top(BB^\top)^{-1}, \qquad LA = BC = I_R.$$

The linear map $\Phi : \mathcal{C}_\mathcal{S} \to \mathcal{U}_\mathcal{S}$, $\Phi(N) = sANB$, is surjective by definition and injective because

$$s^{-1}L\Phi(N)C = N.$$

In particular, two coordinate matrices cannot produce the same update. The elementary matrices with a single one at each $(i, j) \in \mathcal{S}$ form a basis of $\mathcal{C}_\mathcal{S}$. This space has dimension $|\mathcal{S}|$, and the linear isomorphism Φ preserves that dimension.

2. Rank preservation. Matrix multiplication cannot increase rank, so $\mathrm{rank}(ANB) \le \mathrm{rank}(N)$. Conversely, $N = L(ANB)C$ implies $\mathrm{rank}(N) \le \mathrm{rank}(ANB)$. Multiplication by the nonzero scalar s preserves rank, giving $\mathrm{rank}(sANB) = \mathrm{rank}(N)$.

3. Maximum rank and matching. Let $k = \nu(\mathcal{S})$. A matching of size k consists of allowed pairs $(i_1, j_1), \ldots, (i_k, j_k)$ with all row indices distinct and all column indices distinct. Put ones at these pairs and zeros elsewhere to obtain N_{match}. Its submatrix on these rows and columns is a permutation matrix, so its rank is at least k; only k rows are nonzero, so its rank is exactly k.

For the upper bound, suppose an allowed matrix had rank $r > k$. It would contain a nonsingular $r \times r$ submatrix, with row indices $p_1, \ldots, p_r$ and column indices $q_1, \ldots, q_r$. Its determinant is

$$\sum_{\sigma \in \mathfrak{S}_r} \mathrm{sgn}(\sigma) \prod_{a=1}^{r} N_{p_a, q_{\sigma(a)}},$$

where $\mathfrak{S}_r$ is the set of permutations of $[r]$. A nonzero determinant requires at least one nonzero product. Its r entries must all belong to $\mathcal{S}$ and have distinct rows and columns, so they form a matching of size r, contradicting maximality of k. Thus every allowed matrix has rank at most k, and the construction attains this bound. Rank preservation transfers it to $\mathcal{U}_\mathcal{S}$.

4. Almost-everywhere attainment. Identify $\mathcal{C}_\mathcal{S}$ with $\mathbb{R}^{|\mathcal{S}|}$ via its allowed entries. For $k > 0$, fix the $k \times k$ minor used in N_{match}. Its determinant is a polynomial in these entries and is not identically zero, since it equals ± 1 at N_{match}. The zero set of a nonzero real polynomial has Lebesgue measure zero. Every matrix of rank below k makes this minor vanish, so all such assignments lie in that zero set. The upper bound rules out rank above k, proving that almost every assignment has rank exactly k. If $k = 0$, the support is empty and the only update is zero, so the statement holds directly. □

Exact representability and unique coordinates. Under the same full-rank assumptions, dense coordinates give

$$\{sANB : N \in \mathbb{R}^{R\times R}\} = \{M : \mathrm{col}(M) \subseteq \mathrm{col}(A),\ \mathrm{row}(M) \subseteq \mathrm{row}(B)\}. \tag{9}$$

If $M = sANB$, every column of M is a linear combination of columns of A, and every row is a linear combination of rows of B. This proves necessity of the two inclusions. For sufficiency, $AA^\dagger$ and $B^\dagger B$ are the orthogonal projectors onto these column and row spaces, respectively. The inclusions therefore imply $AA^\dagger M = M$ and $MB^\dagger B = M$. Setting $N_M = s^{-1}A^\dagger MB^\dagger$ yields

$$sAN_MB = AA^\dagger MB^\dagger B = M.$$

If any N represents M, multiplying by $A^\dagger$ and $B^\dagger$ recovers $N = N_M$, since $A^\dagger A = BB^\dagger = I_R$; hence the coordinates are unique. For a restricted support, the two inclusions remain necessary, but exact representation additionally requires N_M to vanish outside $\mathcal{S}$. This condition is also sufficient by the same construction.

B Projected Task Directions and Sparse Edits

Proof of Theorem 2. **1. Orthogonal atoms and the reachable-space projector.** Let $\mathsf{P}_\mathcal{S}$ zero entries outside the allowed support. For one layer, define $E_{ij} = a_i b_j^\top$ and use the Frobenius inner product $\langle X, Y\rangle_F = \mathrm{tr}(X^\top Y)$. The assumptions $A^\top A = BB^\top = I$ give

$$\langle E_{ij}, E_{pq}\rangle_F = (a_i^\top a_p)(b_j^\top b_q) = \delta_{ip}\delta_{jq}.$$

Thus the allowed atoms are orthonormal and span $\mathcal{U}_\mathcal{S}$; the nonzero multiplier s does not change the span. Since $\langle H, E_{ij}\rangle_F = a_i^\top H b_j = (A^\top H B^\top)_{ij}$, the orthogonal projection is

$$\Pi_{\mathcal{U}_\mathcal{S}} H = \sum_{(i,j)\in\mathcal{S}} \langle H, E_{ij}\rangle_F E_{ij} = A\mathsf{P}_\mathcal{S}(A^\top H B^\top)B.$$

2. The coordinate gradient is a projected weight gradient. Evaluate all derivatives at N_0. An allowed perturbation D in coordinates changes the weight by $sADB$, so the chain rule gives

$$\frac{d}{d\epsilon}\mathcal{L}_m(N_0 + \epsilon D)\Big|_{\epsilon=0} = \langle H, sADB\rangle_F = \langle sA^\top H B^\top, D\rangle_F.$$

Because D vanishes outside $\mathcal{S}$, the active-coordinate gradient, written as a matrix with zeros elsewhere, is $\mathbf{g} = s\mathsf{P}_\mathcal{S}(A^\top H B^\top)$. Mapping its negative back to weight space gives

$$sA(-\mathbf{g})B = -s^2 A\mathsf{P}_\mathcal{S}(A^\top H B^\top)B = -s^2\Pi_{\mathcal{U}_\mathcal{S}} H,$$

which proves Equation 7. For any reachable perturbation Z with $\|Z\|_F = 1$, orthogonality and Cauchy–Schwarz give

$$\langle H, Z\rangle_F = \langle \Pi_{\mathcal{U}_\mathcal{S}} H, Z\rangle_F \geq -\|\Pi_{\mathcal{U}_\mathcal{S}} H\|_F.$$

If the projection is nonzero, equality is attained by its normalized negative. This is the steepest local descent direction within the reachable space. For several layers, use the direct-sum inner product $\langle H, Z\rangle = \sum_\ell \langle H_\ell, Z_\ell\rangle_F$. The projector acts blockwise, so the same conclusion holds for the joint update with the common multiplier s.

3. Masking gives the best sparse approximation. Concatenate the active coordinates into $\mathbb{R}^P$ and suppose $G = \|\mathbf{g}\|_2 > 0$. For $\mathbf{u} = -\mathbf{g}/G$ and $\mathbf{u}_I = \mathbf{1}_I \odot \mathbf{u}$,

$$\|\mathbf{u}_I\|_2^2 = \frac{\sum_{j\in I} g_j^2}{G^2} = \rho_I, \qquad \|\mathbf{u} - \mathbf{u}_I\|_2^2 = \frac{\sum_{j\notin I} g_j^2}{G^2} = 1 - \rho_I.$$

To see optimality beyond masks, consider any vector v supported on I. Disjoint supports give the exact decomposition

$$\|\mathbf{u} - v\|_2^2 = \sum_{j\notin I} u_j^2 + \sum_{j\in I}(u_j - v_j)^2 \geq \sum_{j\notin I} u_j^2.$$

Equality is attained by $v = \mathbf{u}_I$. Optimizing over supports of size at most k therefore reduces to maximizing $\sum_{j\in I} g_j^2$. If a selected entry has smaller magnitude than an unselected entry, exchanging them increases this sum. Repeating such exchanges yields the top-k support, proving best-k-term approximation. Tied magnitudes may give several equally optimal supports; any tie-breaking rule suffices.

4. First-order effect and its sign. The directional derivative of the few-shot objective is

$$\frac{d}{d\eta}\mathcal{L}_m(N_0 + \eta\mathbf{u}_I)\Big|_{\eta=0} = \mathbf{g}^\top \mathbf{u}_I = -\frac{\sum_{j\in I} g_j^2}{G} = -G\rho_I.$$

Together with Step 3, this proves Equation 8. The full direction has derivative $-G$, so ρ_I is precisely its retained fraction of first-order decrease. At a common positive scale, top-k maximizes this decrease among masks that keep the original values. Reversing $\mathbf{u}_I$ changes the derivative to $+G\rho_I$. For $\rho_I > 0$, differentiability thus implies descent along $\mathbf{u}_I$ and ascent along $-\mathbf{u}_I$ for sufficiently small positive steps. If $\rho_I = 0$, the masked direction is zero. □

Finite-step descent. Assume $\nabla\mathcal{L}_m$ is L-Lipschitz, $L > 0$, in a neighborhood containing the edit segment. For $\eta > 0$, integration along that segment and Cauchy–Schwarz yield

$$\begin{aligned}&\mathcal{L}_m(N_0 + \eta\mathbf{u}_I) - \mathcal{L}_m(N_0) - \eta\mathbf{g}^\top\mathbf{u}_I \\ &\quad= \int_0^\eta \big(\nabla\mathcal{L}_m(N_0 + t\mathbf{u}_I) - \mathbf{g}\big)^\top \mathbf{u}_I\, dt \leq \int_0^\eta Lt\|\mathbf{u}_I\|_2^2\, dt = \frac{L\eta^2}{2}\|\mathbf{u}_I\|_2^2.\end{aligned}$$

Substituting the derivative and norm identities gives

$$\begin{aligned}\mathcal{L}_m(N_0 + \eta\mathbf{u}_I) &\leq \mathcal{L}_m(N_0) + \eta\mathbf{g}^\top\mathbf{u}_I + \frac{L\eta^2}{2}\|\mathbf{u}_I\|_2^2 \\ &= \mathcal{L}_m(N_0) - \rho_I\left(\eta G - \frac{L\eta^2}{2}\right).\end{aligned} \tag{10}$$

For $\rho_I > 0$ and $0 < \eta < 2G/L$ within that neighborhood, this bound guarantees a strict decrease in $\mathcal{L}_m$.

Scope. These are standard projection and approximation properties specialized to our coordinates, not guarantees of semantic disentanglement or generalization. The orthogonal-projection interpretation requires orthonormal factors; the coordinate-space masking identities and descent bound do not. The mask optimizes the first-order decrease within the stated family: each selected entry keeps its original normalized-gradient value. It need not minimize the nonlinear loss at a finite step, and the result does not assert that few-shot gradient energy equals held-out accuracy gain. No concentration assumption is imposed: when gradient energy is diffuse, a small coordinate budget may retain only a small fraction of the local descent. All quantities depend on the starting state at which the gradient is evaluated.

C Protocols and Additional Controls

C.1 Datasets and training

Table 4 lists the data used in our experiments, rather than the full benchmark sizes. We train one adapter per task and report unweighted task averages. Multiple-choice predictions maximize continuation log-likelihood; reasoning and code tasks use deterministic generation.

Table 4: Dataset sizes used in the experiments. Train denotes the main-run training cap; Dev denotes data used only in the indicated selection studies. SFT training counts are verified solution records.

Dataset	Train	Dev	Final eval.	Eval. split
Multiple-choice tasks				
PIQA	4,096	128†	1,838	validation
ARC-Easy	2,251	128†	2,376	test
ARC-Challenge	1,119	128†	1,172	test
OpenBookQA	4,096	128†	500	test
HellaSwag	4,096	128†	3,000	validation, capped
WinoGrande	4,096	128†	1,267	validation
Reasoning and code generation				
GSM8K	512	64‡	256	test, capped
MATH	512	128§	256	test, capped
MBPP	512	–	256	test, capped

† Six-task LR study: 384 training and 128 disjoint internal dev examples per task; the common-LR main runs do not use this dev split. ‡ GSM8K selection: 64 held-out source questions and at most 384 training records, followed by formal training with the listed cap. § Qwen3.5 MATH recovery only. An official validation split in the last column serves as final evaluation, not hyperparameter-selection dev.

The principal multiple-choice protocol trains for one epoch, with physical batch size 1, effective batch size 8, and bfloat16 computation. The common-rate comparison uses 2×10^{-4} and the terminal checkpoint, not a separately optimized rate for each basis. Qwen3.5-2B adapts 186 linear maps across attention and MLP blocks. For global coordinates, narrow projections cap R_ℓ at their available dimensions, giving $P_N = \sum_\ell R_\ell \min(d, R_\ell)$ stored coefficients. The Part III global $R = 256, d = 32$ state contains 1,238,016 coefficients. The main tables exclude the earlier 16-bank width-32 runs because they use a different coordinate parameterization. Qwen2.5-3B's global state has 2,064,384 coefficients. Frozen factors are shared model state, not per-task parameters.

The separate LR experiment uses the internal split in Table 4. Its Qwen3.5 control uses the archived multi-bank parameterization, rather than the global coordinates of the main table. Rates are $\{10^{-7}, 5 \times 10^{-7}, 2 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 2 \times 10^{-4}\}$. Seven rates, four bases, six tasks, and two models give 336 conditions. Each gate run trains for one epoch. Figure 3 stars are dev-grid maxima, not formal test estimates.

Randomness and aggregation. Unless otherwise specified, all experiments are run with three random seeds (42, 43, and 44), and we report the mean across the three runs. Per-task results are averaged across seeds before computing the unweighted average across tasks.

Table 5: **Per-task coordinate edits.** Accuracy (%) for 64-example directions at $\eta = 0.5$. Reloc. averages random relocations of the top-10% signed values.

	Zero start				Random start			
Task	Start	Top-10%	Reloc.	Full	Start	Top-10%	Reloc.	Full
PIQA	64.91	65.45	64.60	66.81	64.42	65.51	64.69	66.43
ARC-Easy	71.55	90.53	72.10	91.08	73.23	90.87	72.84	91.25
ARC-Challenge	57.00	76.28	56.71	77.13	56.91	76.19	56.48	77.47
OpenBookQA	46.40	69.40	46.93	70.00	47.00	71.00	46.80	71.20
HellaSwag	28.70	36.40	28.72	38.30	29.13	36.93	29.24	38.53
WinoGrande	52.09	54.93	52.14	55.25	52.49	54.62	52.57	55.80
Average	53.44	65.50	53.53	66.43	53.86	65.85	53.77	66.78

C.2 Basis identity and reconstruction

All coordinate interventions reuse identical basis tensors and support masks.

For width-16 reconstruction, orthonormal factors give $N^\star = A^\top \Delta W_{\mathrm{LoRA}} B^\top$, accounting for the adapter's multiplier before loading the state. Capture is $\|AN^\star B\|_F / \|\Delta W_{\mathrm{LoRA}}\|_F$, aggregated over adapted maps within each task, then averaged across tasks. It is not squared energy. Projected and directly trained coordinates share the same basis in this control. The result is not a reconstruction estimate at $R = 256$.

C.3 Direction and support protocols

Discoverability depends on the starting state. The same dose is not uniformly beneficial from a strong shared state: 64-example mean accuracy falls from 74.59% to 71.57% at $\eta = 0.5$. Zero and randomized gradients have mean cosine similarity 0.957, but their similarity to shared-start gradients is only about 0.09. Useful directions are therefore local to the coordinate state, rather than a single globally beneficial displacement.

Intervention protocol. The oracle direction is $N^\star - N_0$, evaluated at

$$\alpha \in \{-1, -0.75, -0.5, -0.25, 0, 0.125, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2\}.$$

The shared control uses $N_{\mathrm{shared}} = |\mathcal{J}|^{-1} \sum_{j \in \mathcal{J}} N^{(j)}$, where $\mathcal{J}$ contains the other five tasks and excludes the target. All source states use the same fixed basis and allowed support. Oracle random starts are norm-matched synthetic states. Gradient-study random starts instead permute absolute shared-state values within each module and assign random signs, preserving module norms. The two studies thus use different random-state constructions. Each starting score is evaluated alongside its interventions.

Few-shot discovery uses subsets of 8, 16, 32, 64, or 128 examples from the target training split. A new gradient is computed for every starting state and sample budget. Each answer choice is scored by summed continuation log-probability; cross-entropy over choice scores supplies the gradient. Only active N entries receive gradients. The concatenated negative gradient is normalized to unit norm, with $\eta \in \{-1, -0.75, -0.5, -0.25, -0.125, 0, 0.125, 0.25, 0.5, 0.75, 1\}$. Neither the trained target nor dev selection determines the gradient or dose. The reference $\eta = 0.5$ is fixed across tasks, not a per-task optimum.

Support editing sorts the 64-example gradient globally by absolute value. It keeps 10, 20, 40, 60, or 80% of active coordinates; 100% reuses the full direction. Original signed values are retained without renormalization. Each relocated control places the same values on a random active subset, randomly permuting their assignment. This preserves cardinality, signed-value multiset, and norm while changing coordinate location.

C.4 Reasoning and code evaluation

Qwen2.5 SFT runs train for one epoch. The original Qwen3.5 protocol selects four epochs from $\{4, 8, 12\}$ for all five methods using the GSM8K gate in Table 4; records sharing a source question

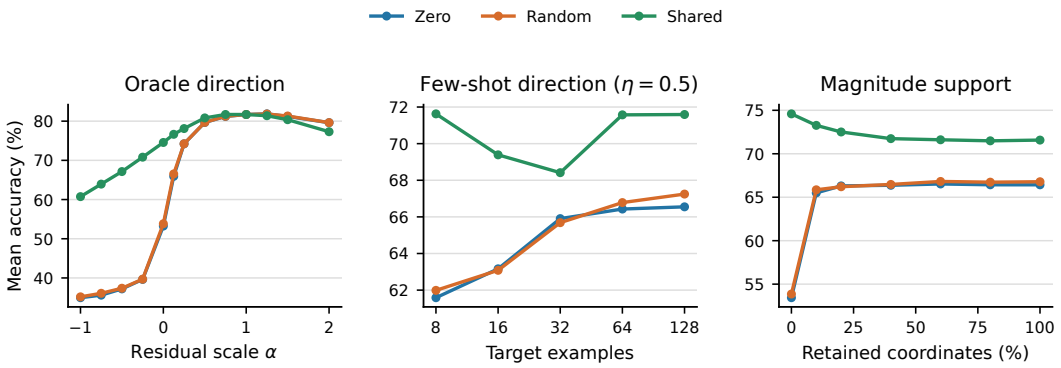


Figure 7: **Three starting states, six-task macro accuracy.** Left: oracle residual intervention. Middle: sample-size response at fixed $\eta = 0.5$, not the best coefficient per task. Right: magnitude-support response at the same dose for 64-example gradients. Strong shared states have less room to improve and can be harmed by the same dose.

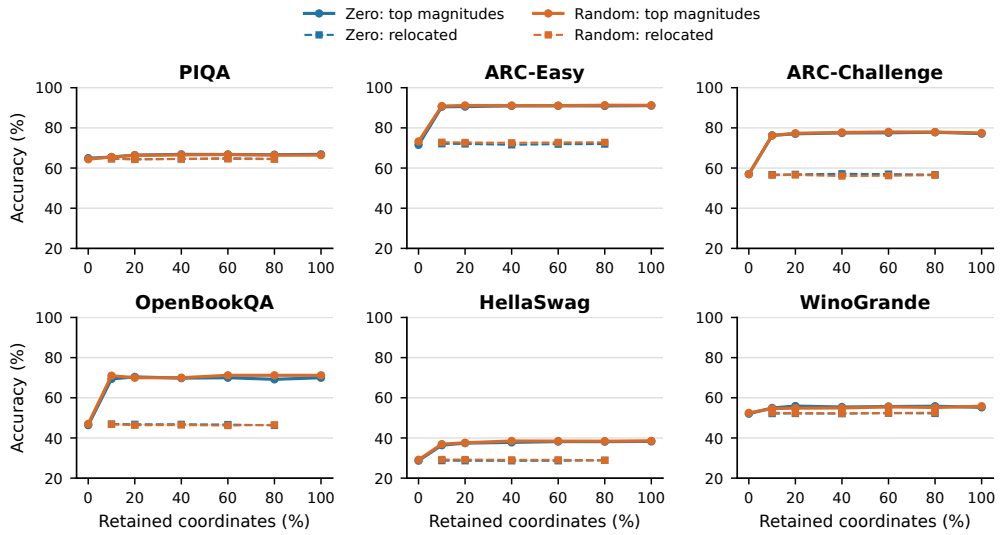


Figure 8: **Complete support curves.** Accuracy for magnitude-selected 64-example directions at $\eta = 0.5$. Solid curves preserve coordinate locations; dashed curves relocate the same signed values. Bands show one sample standard deviation across relocations. The 100% endpoint is the full original direction; relocation controls cover 10–80%.

remain in the same split. For Qwen2.5, LoRA/Unit/QR/SVD use rates 5×10^{-5}, 5×10^{-5}, 10^{-4}, and 10^{-4}, respectively; for Qwen3.5 the original generation runs use 5×10^{-5}, 10^{-4}, 10^{-4}, and 10^{-4}. The separately discussed calibrated Raw runs use 5×10^{-7} and 10^{-7}, respectively; the table's Raw rows instead use 2×10^{-4}. The reasoning table therefore uses method-specific optimization settings.

The subsequent Qwen3.5 MATH recovery selects checkpoints on 128 independent dev questions before evaluation on 256 test questions. It searches one and two epochs; the revised Unit/QR/SVD entries use one epoch and LR 2×10^{-6} with 512 training records. Thus dev data are used in both reasoning selection and the six-task LR study, but not to select Part III intervention doses.

C.5 Uncertainty and Limitations

Small gaps are descriptive, not significance claims. Dose sweeps are not dev-selected deployment rules, and sparse storage does not imply speedups.

C.6 TASK DIRECTIONS IN QR AND SVD BASES

We compare oracle residuals and full-training-set gradients in QR and SVD coordinates on Qwen3.5-2B, from zero and randomized starts (Figure 9). The gradient uses all training examples listed in Table 4, accumulated at the starting state. Its negative direction is normalized and scaled by the target residual norm: $N(\alpha) = N_0 - \alpha\|N^\star - N_0\|_F g/\|g\|_F$. Thus, this control uses the trained target to set the displacement scale, whereas the few-shot protocol uses the absolute displacement $|\eta|$.

Both bases show similar oracle responses: positive residual scaling raises accuracy toward the trained target, while negative scaling lowers it. Full-training-set gradients also produce a signed response. From zero, QR and SVD reach 67.31% and 68.03% at $\alpha = 0.25$, compared with 53.33% at the start; randomized starts reach 67.55% and 67.63%. Larger positive doses reduce these gains, with a sharper decline for QR. The directional pattern therefore appears in both random and pre-trained bases, while the response to larger displacements depends on the basis.

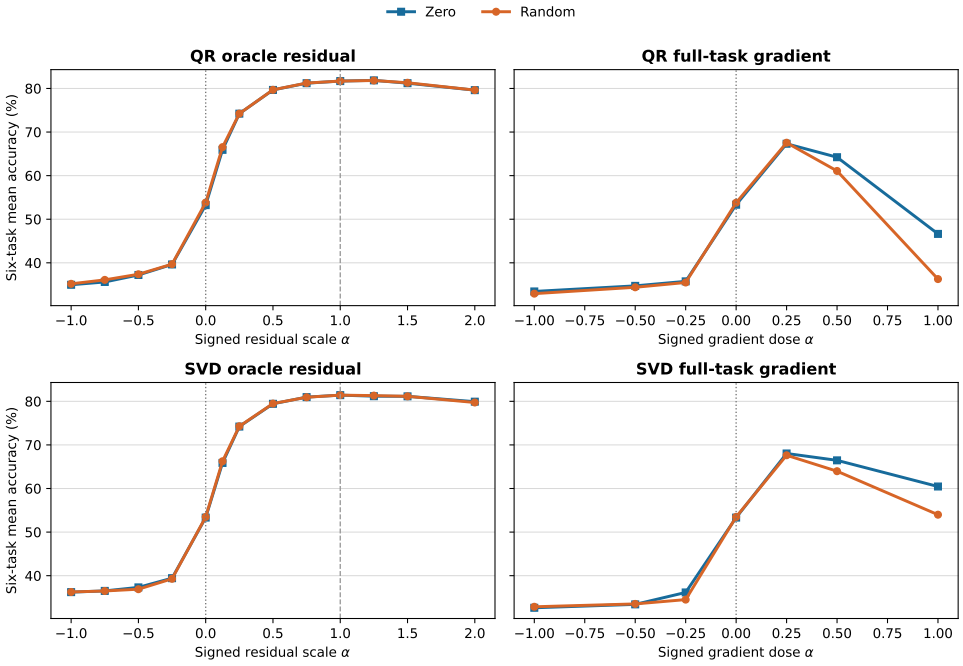


Figure 9: **Task directions across bases.** Six-task mean accuracy on Qwen3.5-2B. Rows compare QR and SVD; columns compare oracle residuals and full-training-set gradients. Both gradient directions are scaled by their respective target residual norms. Curves show zero and randomized starts.

D Implementation Details

Fixed-basis layer. Listing 1 implements $W_0 + sANB$ with frozen outer factors and one trainable value per allowed coordinate. Zero initialization preserves the base layer. Freeze the base model before replacing its linear layers, and keep inputs and factors on the same device and working dtype. This reference materializes N for multiplication.

Listing 1: Fixed bases and learned active coordinates.

```python
"""Minimal reference, not the full experiment runner."""
import torch
from torch import nn
import torch.nn.functional as F


class FixedBasisLinear(nn.Module):
    def __init__(self, base, A, B, support, scale=1.0):
        super().__init__()
        self.base = base.requires_grad_(False)
        self.scale = float(scale)
        A = A.detach().to(base.weight)
        B = B.detach().to(base.weight)
        assert A.shape[0] == base.out_features
        assert B.shape[1] == base.in_features
        assert support.shape == (A.shape[1], B.shape[0])
        self.register_buffer("A", A.clone())
        self.register_buffer("B", B.clone())
        ij = support.to(device=A.device, dtype=torch.bool).nonzero()
        self.register_buffer("rows", ij[:, 0])
        self.register_buffer("cols", ij[:, 1])
        self.values = nn.Parameter(A.new_zeros(len(ij)))

    def coordinates(self):
        N = self.values.new_zeros(self.A.shape[1], self.B.shape[0])
        return N.index_put((self.rows, self.cols), self.values)

    def forward(self, x):
        h = F.linear(x, self.B)
        h = F.linear(h, self.coordinates())
        return self.base(x) + self.scale * F.linear(h, self.A)

    def delta_weight(self):
        return self.scale * (self.A @ self.coordinates() @ self.B)
```

Oracle direction from the trained target. Listing 2 uses the fully trained target, not a single full-batch gradient. At $\alpha = 0$ it recovers the start; at $\alpha = 1$ the target. Negative scales reverse the unnormalized residual.

Listing 2: Oracle residual from full-protocol task training.

```python
import torch
@torch.no_grad()
def oracle_state(start, trained_target, alpha):
    # Same bases/support; target learned by full task training.
    assert start.shape == trained_target.shape
    target = trained_target.to(start)
    residual = target - start
    return start + alpha * residual  # Do not normalize.
```

Directions and coordinate interventions. Listing 3 concatenates active coordinates in a fixed layer order. Compute the mean few-shot loss at the starting state with dropout disabled, then normalize its negative gradient globally. Each dose is applied to the same start, not cumulatively. Masking retains original values; relocation preserves their count and norm but randomizes placement, allowing overlap with the original support. The repeated 100% entries in Table 3 are reference endpoints, not measured permutations.

Listing 3: One-gradient direction discovery, masking, and relocation.

```python
"""Operations on concatenated active coordinates across layers."""
import math
import torch


def gradient_direction(mean_loss, layers):
    params = [layer.values for layer in layers]
    grads = torch.autograd.grad(mean_loss, params)
    g = torch.cat([v.detach().float().flatten() for v in grads])
    norm = g.norm()
    return -g / norm if norm > 0 else torch.zeros_like(g)


def support_edit(direction, fraction, seed=None):
    assert 0 <= fraction <= 1
    # CPU RNG mirrors the archived relocation control.
    u = direction.detach().float().cpu()
    k = min(u.numel(), math.ceil(fraction * u.numel()))
    index = u.abs().topk(k).indices
    values = u[index].clone()
    if seed is not None:
        rng = torch.Generator().manual_seed(seed)
        index = torch.randperm(u.numel(), generator=rng)[:k]
        values = values[torch.randperm(k, generator=rng)]
    edit = torch.zeros_like(u)
    edit[index] = values
    return edit.to(direction)  # No renormalization.


@torch.no_grad()
def apply_direction(layers, start, direction, eta):
    edited = start + eta * direction
    assert edited.numel() == sum(m.values.numel() for m in layers)
    offset = 0
    for layer in layers:
        n = layer.values.numel()
        layer.values.copy_(edited[offset:offset + n].to(layer.values))
        offset += n
```

The listings specify core operations; experimental settings are given in Appendix C.