DIRECTION OR MAGNITUDE? LEARNING SCALARS OVER TASK-CALIBRATED DIRECTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter-efficient fine-tuning (PEFT) reduces the cost of adapting pretrained models, yet standard low-rank methods still optimize matrix-valued factors whose parameter, optimizer-state, and memory costs grow with model width and adapter rank. Fixed-basis approaches further compress task-specific parameters, but purely random directions may be poorly aligned with downstream data, while storing and combining large collections of fixed bases can still incur substantial memory and computational overhead. Therefore, we propose **SCALAR**, a scalar-only adaptation method that combines activation-derived input directions with fixed random output atoms and optimizes only layer-specific coefficients. The activation-calibrated subspace provides task-relevant input directions, whereas grouped random atoms form a compact output dictionary that can additionally be shared across compatible layers. Experiments on commonsense reasoning, mathematical reasoning, code generation, and general instruction tuning show that SCALAR achieves performance comparable to LoRA with up to 55$\times$ fewer trainable parameters. In representative 7B-model experiments, SCALAR reduces peak training memory by 20% and improves training throughput by 6% relative to LoRA, while yielding larger system-efficiency gains over existing random-basis methods. These results show that scalar-only adaptation can improve not only nominal parameter efficiency, but also practical memory consumption and training speed.

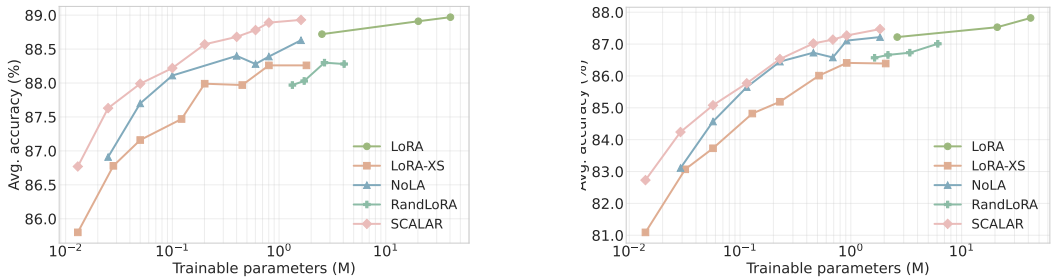
1 INTRODUCTION

Large language models (LLMs) achieve strong general-purpose performance, but full-parameter fine-tuning incurs substantial training memory, optimization, and task-specific storage costs. Parameter-efficient fine-tuning (PEFT) alleviates these costs by updating only a small subset of parameters, with LoRA (Hu et al., 2022) serving as a widely adopted baseline. Nevertheless, for an $m \times n$ linear layer, its trainable parameter count still scales as $R(m+n)$. Recent methods further reduce this cost by learning compact coefficients over fixed bases (Bałazy et al., 2024; Kopiczko et al., 2024; Koohpayegani et al., 2024; Albert et al., 2025), making the construction of these bases increasingly important. Random bases are inexpensive but may be poorly aligned with downstream data, whereas weight- or activation-derived subspaces (Meng et al., 2024; Yang et al., 2024b; Paischer et al., 2025) provide stronger task relevance but generally retain trainable matrix-valued factors. Meanwhile, recent evidence suggests that useful task-improving perturbations may be abundant in the neighborhood of pretrained weights (Gan & Isola, 2026). This motivates a natural question:

Can effective adaptation be achieved by constructing task-relevant directions before fine-tuning and learning only how strongly they should contribute?

We first investigate this question through a learned-direction reweighting diagnostic. We train a LoRA adapter, freeze its learned update direction for each adapted matrix, and re-optimize only one scalar coefficient per matrix. On a 7B model, this reduces the number of trainable parameters from 2,523,136 to only 196, while closely reproducing LoRA's optimization behavior and achieving comparable or slightly better downstream accuracy. This observation suggests that much of the task-specific adaptation can be retained by appropriately reweighting informative directions.

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.



(a) Qwen2.5-7B (b) LLaMA3-8B

Figure 1: **Average accuracy on 8 commonsense reasoning tasks across trainable parameter budgets.** SCALAR achieves a favorable parameter–accuracy trade-off among PEFT methods.

Motivated by this observation, we propose **SCALAR: Subspace-Calibrated Adaptation via Layer-wise Atom Reweighting**. SCALAR constructs a fixed input-side subspace from the leading singular vectors of downstream activations and associates each input direction with multiple fixed random output atoms. Adaptation is performed by learning scalar coefficients over this grouped dictionary. The activation-derived basis provides task-calibrated input directions, while the random output atoms expand the candidate update space without requiring trainable matrix-valued factors. Importantly, grouping multiple output atoms around each input direction decouples the dictionary capacity $K = RH$ from the input-subspace dimension and maximum update rank R. The output atoms within each group can be merged before being applied to the activations, allowing the candidate dictionary to grow without making the token-dependent computation scale directly with K.

We theoretically characterize why activation-derived directions are suitable for this parameterization. The leading activation singular vectors maximize captured activation energy among all subspaces of the same dimension and, under an alignment condition, provide a favorable initial gradient signal for optimization. Empirically, SCALAR achieves competitive performance on commonsense reasoning, mathematical reasoning, and code generation while using substantially fewer trainable parameters than representative PEFT baselines. System measurements further show that the compact parameterization translates into lower training memory and higher throughput.

Our main contributions are summarized as follows:

Task-calibrated and decoupled parameterization. We introduce SCALAR, which combines an activation-calibrated input subspace with a grouped random output dictionary. Its design decouples the number of candidate rank-one atoms from the input-subspace dimension and maximum update rank, providing a flexible adaptation space under a compact trainable parameter budget.

Analysis and diagnostic evidence. We establish the activation-energy optimality of the calibrated input subspace and analyze how its alignment with downstream activations affects the scalar gradient signal. A learned-direction reweighting diagnostic further shows that much of LoRA's observed adaptation behavior can be retained by reweighting fixed task-informed directions.

Broad empirical validation. We evaluate SCALAR on commonsense reasoning, mathematical reasoning, code generation, and general-domain knowledge evaluation. The experiments cover three model families and multiple parameter scales up to 32B, with additional large-model evaluations in the Appendix. Across these settings, SCALAR provides a favorable parameter–performance trade-off relative to full fine-tuning and representative PEFT methods. System measurements and ablation studies further assess its computational efficiency and sensitivity to key design choices.

2 Related Work

Parameter-efficient fine-tuning adapts pretrained models using a small set of task-specific parameters. We focus on two directions most relevant to SCALAR: task-aware low-rank adaptation and fixed-basis coefficient learning.

Table 1: **Comparison of trainable parameters across methods.** Let $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ and $d = \min(m, n)$. LoRA, LoRA-XS, VeRA, NOLA, and SCALAR use rank R. NOLA uses k_A and k_B random bases, while SCALAR uses H random output atoms per input direction.

	LoRA	LoRA-XS	VeRA	NOLA	RandLoRA	SCALAR
$\Delta\boldsymbol{W}$	$\boldsymbol{BA}$	$\boldsymbol{BMA}$	$\boldsymbol{\Lambda}_b\boldsymbol{B}\boldsymbol{\Lambda}_d\boldsymbol{A}$	$\left(\sum_{i=1}^{k_B}\alpha_i\boldsymbol{B}_i\right)\left(\sum_{j=1}^{k_A}\beta_j\boldsymbol{A}_j\right)$	$\sum_{j=1}^{q}\boldsymbol{B}_j\boldsymbol{\Lambda}_j\boldsymbol{A}\boldsymbol{\Gamma}_j$	$\sum_{r=1}^{R}\sum_{h=1}^{H}\gamma_{r,h}\boldsymbol{b}_{r,h}\boldsymbol{a}_r^\top$
Trainable params	$R(m+n)$	R^2	$m+R$	k_A+k_B	$\lceil d/R\rceil(d+R)$	RH

LoRA (Hu et al., 2022) represents each weight update with two trainable low-rank factors. Subsequent variants improve rank allocation, parameterization, optimization, and routing (Zhang et al., 2023; 2024; Liu et al., 2024; Tian et al., 2024; Wang et al., 2024; 2025; Guo et al., 2025; Morris et al., 2026). Another line of work constructs more informative adaptation subspaces. PiSSA (Meng et al., 2024) derives its initialization from pretrained weights, while EVA (Paischer et al., 2025) uses dominant activation directions. CorDA (Yang et al., 2024b) combines weight information with task-dependent activation statistics. TLoRA (Lin et al., 2026) freezes a data-driven input-side factor, whereas Astra (Liu et al., 2026) constructs adapters from activation tail directions. These methods improve task awareness but generally retain trainable matrix-valued factors.

Fixed-basis methods instead learn compact coefficients over frozen directions. LoRA-XS (Bałazy et al., 2024) optimizes a small core between fixed projections. Other methods use spectral, frequency-domain, tensor, or random bases (Nooralinejad et al., 2023; Lingam et al., 2024; Gao et al., 2024; Gu et al., 2026). Among random-basis approaches, NOLA (Koohpayegani et al., 2024) learns combinations of fixed random matrices, VeRA (Kopiczko et al., 2024) learns scaling vectors over shared random projections, and RandLoRA (Albert et al., 2025) combines multiple random low-rank bases with diagonal scalings. SCALAR differs by calibrating its fixed input directions from downstream activations, pairing them with random output atoms, and optimizing only layer-specific scalar weights.

3 Motivation

In this section, we motivate SCALAR from two complementary perspectives. We first present an oracle diagnostic showing that, once useful adaptation directions are given, re-optimizing only their scalar magnitudes can closely recover the behavior of a trained LoRA adapter with orders of magnitude fewer trainable parameters. This raises the central question of this work: can useful directions be constructed before fine-tuning, without first training LoRA or performing backward optimization? We then analyze why dominant activation directions provide an effective fixed input subspace.

3.1 Learned-Direction Reweighting

We begin with a diagnostic experiment that separates adaptation directions from their magnitudes. Any trained rank-r LoRA update can be decomposed through singular value decomposition as

$$\Delta\boldsymbol{W}_{\text{LoRA}} = \sum_{i=1}^{r}\sigma_i\boldsymbol{u}_i\boldsymbol{v}_i^\top,$$

where each pair $(\boldsymbol{u}_i, \boldsymbol{v}_i)$ defines a rank-one adaptation direction and σ_i specifies its magnitude. We freeze these directions and re-optimize only their scalar coefficients:

$$\Delta\boldsymbol{W} = \sum_{i=1}^{r}\gamma_i\boldsymbol{u}_i\boldsymbol{v}_i^\top. \tag{1}$$

We report the Qwen2.5-7B results in Table 2. In this setting, a rank-one LoRA adapter contains 2,523,136 trainable parameters, whereas the learned-direction reweighting model optimizes only 196 coefficients. Despite this substantial reduction, it closely follows the optimization behavior of LoRA and achieves slightly higher average accuracy. The same trend is observed on LLaMA3-8B; detailed training trajectories and additional results are provided in Appendix C.1.

Table 2: **Accuracy of LoRA and learned-direction reweighting on 8 commonsense reasoning tasks.** Learned-direction reweighting achieves comparable performance while optimizing fewer than 0.01% as many parameters as LoRA.

Method	Params	BoolQ	PIQA	SIQA	HS	WG	ARC-E	ARC-C	OBQA	Avg.
LoRA	2,523,136	74.1	90.3	81.8	95.7	88.7	96.2	89.4	91.8	88.5
Learned-direction reweighting	196	73.9	90.1	81.8	96.0	89.0	96.4	89.9	92.4	88.7

These observations suggest that, once informative adaptation directions are available, a substantial part of the remaining adaptation can be expressed by learning only their magnitudes. In our experiments, this restriction preserves and can slightly improve the reported downstream accuracy. The oracle is not practical because obtaining its directions requires first training a complete LoRA adapter. SCALAR is therefore motivated by constructing useful directions before fine-tuning and learning only a compact set of scalar coefficients.

3.2 Theoretical Analysis

The central question behind SCALAR is whether the adaptation directions must be learned during fine-tuning, or whether they can be constructed before optimization so that training only determines their magnitudes. We first isolate the role of the input-side LoRA factor and show that the dominant activation subspace provides a principled choice of fixed input directions.

3.2.1 Input-Side Subspace in Low-Rank Adaptation

Consider a frozen linear transformation with a LoRA branch:

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{A}\boldsymbol{x},$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is the input activation, $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ is the frozen pretrained weight, and $\boldsymbol{A} \in \mathbb{R}^{R\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times R}$ are the input- and output-side factors, respectively. Since the factorization is scale-indeterminate, rescaling $\boldsymbol{A}$ and inversely rescaling $\boldsymbol{B}$ leaves the update unchanged. We therefore analyze the row subspace of $\boldsymbol{A}$ under a normalized representation. We collect the assumptions used throughout the analysis below.

Assumption 3.1 (Finite second moments)**.** The activation $\boldsymbol{x} \in \mathbb{R}^n$ and the back-propagated gradient $\boldsymbol{g} \in \mathbb{R}^m$ satisfy

$$\mathbb{E}\big[\|\boldsymbol{x}\|_2^2\big] < \infty, \qquad \mathbb{E}\big[\|\boldsymbol{g}\|_2^2\|\boldsymbol{x}\|_2^2\big] < \infty.$$

Define the activation second-moment matrix as

$$\boldsymbol{\Sigma} := \mathbb{E}\big[\boldsymbol{x}\boldsymbol{x}^\top\big]. \tag{2}$$

Assumption 3.2 (Normalized subspace representation)**.** The R-dimensional input subspace is represented by $\boldsymbol{A} \in \mathbb{R}^{R\times n}$ with orthonormal rows:

$$\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_R, \qquad 1 \le R < n.$$

This assumption fixes the scale ambiguity only for analysis; the practical adapter does not enforce orthogonality. Any full-row-rank factor admits an equivalent row-orthonormal representative with the same row subspace.

Assumption 3.3 (Standard zero-output initialization)**.** The loss $\ell(\boldsymbol{z}, y)$ is differentiable with respect to $\boldsymbol{z}$, and the output-side factor is initialized as $\boldsymbol{B}^0 = \boldsymbol{0}$. Define

$$\boldsymbol{g} := \nabla_{\boldsymbol{z}}\ell(\boldsymbol{z}, y)|_{\boldsymbol{z}=\boldsymbol{W}\boldsymbol{x}}.$$

Assumption 3.4 (Covariance alignment)**.** Define the gradient-weighted activation second moment as

$$\boldsymbol{M} := \mathbb{E}\big[\|\boldsymbol{g}\|_2^2\boldsymbol{x}\boldsymbol{x}^\top\big]. \tag{3}$$

There exist $c > 0$ and $\varepsilon \ge 0$ such that

$$\|\boldsymbol{M} - c\boldsymbol{\Sigma}\|_2 \le \varepsilon.$$

The quantity ε measures how strongly gradient weighting distorts the activation geometry. This assumption does not require $\boldsymbol{x}$ and $\boldsymbol{g}$ to be independent. The special case $\varepsilon = 0$ includes settings in which $\mathbb{E}[\|\boldsymbol{g}\|_2^2 \mid \boldsymbol{x}]$ is constant.

Assumption 3.5 (Idealized normalized subspace dynamics)**.** For the analysis of direction learning, let $\boldsymbol{A}^t \in \mathbb{R}^{R\times n}$ be a continuously differentiable trajectory for $t \geq 0$ satisfying

$$\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t} = \boldsymbol{A}^t\boldsymbol{\Sigma} - \left(\boldsymbol{A}^t\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right)\boldsymbol{A}^t, \quad \boldsymbol{A}^0(\boldsymbol{A}^0)^\top = \boldsymbol{I}_R. \tag{4}$$

Equation 4 is an idealized gradient flow. It is used only to characterize subspace alignment and is not assumed to exactly describe practical optimization with Adam (Kingma & Ba, 2014).

3.2.2 Why Activation-SVD?

For a normalized input subspace, define the captured activation energy as

$$\Phi(\boldsymbol{A}) := \mathbb{E}\left[\|\boldsymbol{A}\boldsymbol{x}\|_2^2\right].$$

Let

$$\boldsymbol{\Sigma} = \boldsymbol{U}\,\mathrm{Diag}\,(\lambda_1,\ldots,\lambda_n)\,\boldsymbol{U}^\top, \lambda_1 \geq \cdots \geq \lambda_n \geq 0,$$

and let $\boldsymbol{U}_R \in \mathbb{R}^{n\times R}$ contain the leading R eigenvectors. We define $\boldsymbol{A}_{\mathrm{svd}} := \boldsymbol{U}_R^\top$.

Theorem 3.6 (Optimal activation coverage)**.** *Under Assumptions 3.1 and 3.2, we have*

$$\Phi(\boldsymbol{A}) = \mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\right), \text{ and } \max_{\boldsymbol{A}\boldsymbol{A}^\top=\boldsymbol{I}_R} \Phi(\boldsymbol{A}) = \sum_{i=1}^{R}\lambda_i.$$

The maximum is attained by $\boldsymbol{A}_{\mathrm{svd}}$. If $\boldsymbol{A}_{\mathrm{rand}}$ represents a uniformly random R-dimensional row subspace, then

$$\mathbb{E}_{\boldsymbol{A}_{\mathrm{rand}}}\left[\Phi(\boldsymbol{A}_{\mathrm{rand}})\right] = \frac{R}{n}\,\mathrm{Tr}(\boldsymbol{\Sigma}).$$

Consequently,

$$\Phi(\boldsymbol{A}_{\mathrm{svd}}) \geq \mathbb{E}_{\boldsymbol{A}_{\mathrm{rand}}}\left[\Phi(\boldsymbol{A}_{\mathrm{rand}})\right].$$

The inequality is strict when $R < n$ and $\boldsymbol{\Sigma}$ is not a scalar multiple of $\boldsymbol{I}_n$.

Theorem 3.6 establishes that the leading activation directions retain the maximum possible input energy among all subspaces of the same dimension. A random subspace captures only an R/n fraction of the total activation energy in expectation.

At the initialization in Assumption 3.3, define the expected gradient energy received by the output-side factor as

$$\mathcal{G}(\boldsymbol{A}) := \mathbb{E}\left[\|\nabla_{\boldsymbol{B}}\ell\|_F^2\right].$$

Theorem 3.7 (Near-optimal initial output-side gradient signal)**.** *Under Assumptions 3.1, 3.2, 3.3, and 3.4, we have*

$$\mathcal{G}(\boldsymbol{A}) = \mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{M}\boldsymbol{A}^\top\right).$$

Let $\boldsymbol{A}_* \in \arg\max_{\boldsymbol{A}\boldsymbol{A}^\top=\boldsymbol{I}_R} \mathcal{G}(\boldsymbol{A})$. Then

$$0 \leq \mathcal{G}(\boldsymbol{A}_*) - \mathcal{G}(\boldsymbol{A}_{\mathrm{svd}}) \leq 2R\varepsilon.$$

In particular, $\boldsymbol{A}_{\mathrm{svd}}$ is exactly optimal when $\varepsilon = 0$. Moreover,

$$\mathbb{E}_{\boldsymbol{A}_{\mathrm{rand}}}\left[\mathcal{G}(\boldsymbol{A}_{\mathrm{rand}})\right] = \frac{R}{n}\,\mathrm{Tr}(\boldsymbol{M}).$$

Theorem 3.7 connects activation coverage to optimization. When the gradient-weighted geometry remains close to the activation geometry, activation-SVD also provides a near-optimal input subspace for the initial output-side learning signal.

Theorem 3.8 (Alignment under the idealized subspace flow)**.** *Under Assumptions 3.1 and 3.5, we have*

$$\boldsymbol{A}^t(\boldsymbol{A}^t)^\top = \boldsymbol{I}_R, \forall t \geq 0$$

Moreover,

$$\frac{\mathrm{d}}{\mathrm{d}t}\Phi(\boldsymbol{A}^t) = 2\left\|\left(\boldsymbol{I}_n - (\boldsymbol{A}^t)^\top \boldsymbol{A}^t\right)\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right\|_F^2 \geq 0.$$

Therefore,

$$\Phi(\boldsymbol{A}^t) \geq \Phi(\boldsymbol{A}^0), \forall t \geq 0$$

If $\boldsymbol{A}^0$ is a uniformly random row subspace, then

$$\mathbb{E}\left[\Phi(\boldsymbol{A}^t)\right] \geq \frac{R}{n}\operatorname{Tr}(\boldsymbol{\Sigma}), \forall t \geq 0$$

Theorem 3.8 provides an idealized interpretation of learned direction alignment: optimization can move a random input subspace toward higher-coverage directions. This also helps explain why activation-based LoRA initialization can be effective. SCALAR goes one step further: instead of using the activation subspace only as an initialization, it fixes the activation-derived directions and spends the trainable budget on scalar reweighting. We empirically examine this behavior in Appendix C.2, where LoRA adapters on a 7B model exhibit row-normalized input subspaces that capture more activation energy than their random-initialized counterparts during fine-tuning.

Together, these results support our input-side design: SCALAR constructs $\boldsymbol{A}$ from the leading activation directions and keeps it fixed. We next address the output side using a shared random dictionary and layer-specific scalar reweighting, avoiding a trainable dense matrix-valued factor.

4 SCALAR

Section 3 motivates fixing the input-side adaptation subspace, but leaves the output-side parameterization unspecified. SCALAR pairs each activation-derived input direction with multiple fixed random output atoms and learns only their scalar coefficients. By grouping several output atoms around the same input direction, SCALAR increases the number of candidate output directions without proportionally increasing the effective update rank.

Consider a frozen linear layer $\boldsymbol{W} \in \mathbb{R}^{m\times n}$. SCALAR parameterizes its update as

$$\Delta\boldsymbol{W} = s\sum_{r=1}^{R}\sum_{h=1}^{H}\gamma_{r,h}\boldsymbol{b}_{r,h}\boldsymbol{a}_r^\top, \tag{5}$$

where $\boldsymbol{a}_r \in \mathbb{R}^n$ is a fixed task-aware input direction, $\boldsymbol{b}_{r,h} \in \mathbb{R}^m$ is a fixed random output atom, $\gamma_{r,h} \in \mathbb{R}$ is trainable, and s is a fixed scaling factor. Each adapted layer therefore contains only $K = RH$ trainable scalars.

4.1 Task-Aware Input Directions

We collect S input activations from a small calibration set and stack them into $\boldsymbol{X} \in \mathbb{R}^{S\times n}$. Following the second-moment analysis in Section 3, $\boldsymbol{X}$ is not mean-centered. We compute the SVD and retain the leading R right singular vectors:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top, \qquad \boldsymbol{A} = \boldsymbol{V}_{:,1:R}^\top \in \mathbb{R}^{R\times n}. \tag{6}$$

The resulting directions span the dominant empirical activation subspace and remain frozen throughout fine-tuning. Unlike activation-aware LoRA initialization, SCALAR uses them as a fixed input basis rather than as the initial value of a trainable factor.

4.2 Grouped and Shared Random Output Dictionaries

For each input direction $\boldsymbol{a}_r$, we sample H fixed output atoms and collect them into

$$\boldsymbol{B}_r = [\boldsymbol{b}_{r,1} \quad \cdots \quad \boldsymbol{b}_{r,H}] \in \mathbb{R}^{m\times H}.$$

Table 3: **Commonsense reasoning accuracy of LLaMA3-8B and Qwen2.5-7B across 8 benchmarks.** Results are averaged over three independent runs. Bold values indicate the best results among methods with the same or comparable low trainable-parameter budgets.

Model	Method	Params (%)	BoolQ	PIQA	SIQA	HS	WG	ARC-E	ARC-C	OBQA	Avg.
Qwen2.5-7B	FT	100.00	75.3	89.7	82.0	96.1	89.7	95.6	88.4	93.1	$88.7_{\pm 0.1}$
	LoRA ($r = 1$)	0.0331	74.3	90.3	81.9	95.8	88.7	96.1	89.2	91.7	$88.5_{\pm 0.1}$
	LoRA ($r = 8$)	0.2643	74.6	90.4	81.9	96.3	89.3	95.9	89.5	92.3	$88.8_{\pm 0.1}$
	VeRA	0.0208	73.7	90.2	81.5	95.6	87.7	96.3	88.8	91.5	$88.2_{\pm 0.2}$
	NOLA	0.0105	73.6	90.0	82.0	95.8	88.6	96.4	89.2	91.5	$88.4_{\pm 0.0}$
	LoRA-XS	0.0105	73.4	89.8	82.4	95.6	87.2	96.5	88.5	90.9	$88.0_{\pm 0.2}$
	RandLoRA	0.0175	73.9	89.5	81.2	95.8	88.3	96.1	89.1	90.8	$88.1_{\pm 0.2}$
	SCALAR ($K = 1024$)	0.0026	73.4	90.3	82.4	96.0	87.5	96.8	89.0	92.3	$\mathbf{88.5}_{\pm 0.1}$
	SCALAR ($K = 2048$)	0.0052	73.7	90.0	82.5	96.1	88.2	96.7	89.2	91.9	$\mathbf{88.6}_{\pm 0.0}$
	SCALAR ($K = 4096$)	0.0105	74.0	90.3	82.2	96.2	89.3	96.5	89.5	93.0	$\mathbf{88.9}_{\pm 0.1}$
LLaMA3-8B	FT	100.00	76.2	90.4	83.0	96.7	89.4	94.1	85.1	88.8	$88.0_{\pm 0.2}$
	LoRA ($r = 1$)	0.0326	74.3	89.9	81.9	96.3	89.5	93.3	83.3	89.0	$87.2_{\pm 0.2}$
	LoRA ($r = 8$)	0.2604	75.9	90.0	83.0	96.6	89.4	93.3	83.1	87.8	$87.4_{\pm 0.3}$
	VeRA	0.0199	74.8	88.9	81.6	95.9	87.1	92.8	82.1	87.5	$86.3_{\pm 0.3}$
	NOLA	0.0114	74.7	89.5	82.8	96.4	88.3	92.9	82.9	87.4	$86.9_{\pm 0.1}$
	LoRA-XS	0.0114	73.3	88.3	82.7	96.0	88.1	92.4	81.9	87.7	$86.2_{\pm 0.2}$
	RandLoRA	0.0203	75.1	89.5	82.3	95.5	86.7	93.2	81.7	88.4	$86.6_{\pm 0.0}$
	SCALAR ($K = 1024$)	0.0028	73.5	89.2	82.4	96.2	88.0	92.8	82.3	87.6	$\mathbf{86.5}_{\pm 0.3}$
	SCALAR ($K = 2048$)	0.0057	74.2	89.7	82.5	96.3	87.5	93.3	83.5	88.3	$\mathbf{86.9}_{\pm 0.1}$
	SCALAR ($K = 4096$)	0.0114	74.0	89.6	82.8	96.8	88.8	93.1	83.9	89.2	$\mathbf{87.3}_{\pm 0.1}$

The atoms are sampled once from a zero-mean isotropic distribution and remain frozen. Let $\boldsymbol{\gamma}_r = [\gamma_{r,1}, \ldots, \gamma_{r,H}]^\top$ and $\widetilde{\boldsymbol{b}}_r = \boldsymbol{B}_r \boldsymbol{\gamma}_r$. Then equation 5 can be written as

$$\Delta \boldsymbol{W} = s \sum_{r=1}^{R} \widetilde{\boldsymbol{b}}_r \boldsymbol{a}_r^\top = s \widetilde{\boldsymbol{B}} \boldsymbol{A}, \qquad \operatorname{rank}(\Delta \boldsymbol{W}) \le R.$$

Here, R determines the input-subspace dimension and maximum update rank, while H controls the number of candidate output directions associated with each input direction. Since the H output atoms can be merged before multiplication with the activations, the token-dependent computation scales with R rather than $K = RH$. Appendix C.6.1 provides the corresponding complexity analysis and empirical throughput comparison. For an input $\boldsymbol{x}$, the adapter output is

$$\Delta \boldsymbol{W} \boldsymbol{x} = s \sum_{r=1}^{R} \left(\boldsymbol{a}_r^\top \boldsymbol{x} \right) \boldsymbol{B}_r \boldsymbol{\gamma}_r.$$

Under isotropic random output atoms, the expected initial gradient energy of the scalar coefficients is proportional to the activation energy captured by $\boldsymbol{A}$. This motivates the activation-SVD input basis; the complete derivation is provided in Appendix A.4.

Each adapted layer maintains its own activation-derived basis $\boldsymbol{A}$ and trainable coefficient matrix $\boldsymbol{\Gamma}$. Layers of the same module type with compatible output dimensions may share the fixed output dictionaries $\{\boldsymbol{B}_r\}_{r=1}^{R}$. This sharing reduces basis storage while preserving layer-specific adaptation through $\boldsymbol{A}$ and $\boldsymbol{\Gamma}$. As shown in Appendix C.6.1, using separate dictionaries increases memory consumption without improving accuracy.

5 Experimental Results

In this section, we evaluate the performance, parameter efficiency, and system-level cost of the proposed method across several language adaptation tasks. The Appendix includes additional diagnostic analyses, comparisons with enhanced LoRA variants, general instruction fine-tuning, larger-model evaluations, and extensive ablations on the parameterization, calibration, rank, initialization, and cross-layer sharing, together with full implementation details and parameter-budget comparisons.

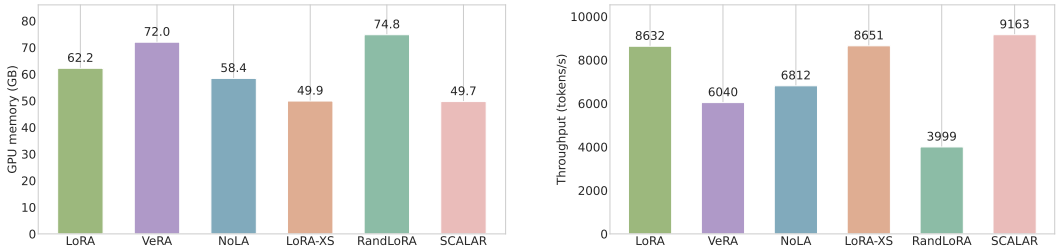
Training Configuration. We evaluate SCALAR on commonsense reasoning with Qwen2.5 (Yang et al., 2024a) and LLaMA3 (Grattafiori et al., 2024), and further test transfer to mathematical reasoning and code generation. We also ablate the initialization of the fixed bases $\boldsymbol{A}$ and $\boldsymbol{B}$.

Baselines. We compare SCALAR with full-parameter fine-tuning and five representative PEFT methods: **Full Fine-tuning (FT)** as a full-capacity reference; **LoRA** (Hu et al., 2022), the standard trainable low-rank parameterization; **VeRA** (Kopiczko et al., 2024), which shares random low-rank projections and learns layer-specific scaling vectors; **LoRA-XS** (Bałazy et al., 2024), which optimizes a compact matrix between fixed projections; **NOLA** (Koohpayegani et al., 2024), which learns combinations of fixed random bases; and **RandLoRA** (Albert et al., 2025), which aggregates random low-rank bases with trainable diagonal scalings.

5.1 Commonsense Reasoning Benchmarks

We present the main accuracy results in Table 3 and the parameter–performance comparison in Figure 1. We fine-tune all methods on the Commonsense170K training set (Hu et al., 2023) and evaluate on 8 commonsense reasoning benchmarks. We conduct experiments on both LLaMA3-8B and Qwen2.5-7B, and report the mean accuracy over three independent runs. A more comprehensive comparison covering multiple trainable-parameter budgets for NOLA, LoRA-XS, and the other PEFT baselines is provided in Appendix Table 10.

Main Results. Table 3 shows that SCALAR achieves competitive performance with substantially fewer trainable parameters. It reaches 87.3 average accuracy on LLaMA3-8B, close to rank-8 LoRA at 87.4, and obtains 88.9 on Qwen2.5-7B, compared with 88.7 for full fine-tuning and 88.8 for rank-8 LoRA. These small differences should be interpreted together with the reported variation. Figure 1 further shows that SCALAR provides a favorable parameter–performance trade-off, outperforming other PEFT baselines at comparable budgets while approaching LoRA-level performance.



(a) Memory comparison (b) Training tokens/s

Figure 2: **Memory and throughput for fine-tuning Qwen2.5-7B on commonsense reasoning.** We report peak GPU memory with a device batch size of 16 and token throughput on 2 NVIDIA H20 GPUs. SCALAR uses less memory and achieves higher throughput than the compared baselines.

Memory and Speed. To evaluate the system-level efficiency of our method, we report the actual GPU memory consumption and training token throughput when fine-tuning Qwen2.5-7B. The results are shown in Figure 2. As shown, SCALAR achieves the lowest actual memory consumption among all PEFT methods, reducing memory usage by 20% compared with LoRA, 15% compared with NOLA, and 33% compared with RandLoRA, which exhibits the highest memory consumption. In addition, SCALAR improves training throughput by 6% over LoRA, 34% over NOLA, and approximately 130% over RandLoRA.

5.2 Mathematical Reasoning and Code Generation

We further evaluate the transferability of our method to mathematical reasoning and code generation tasks. For mathematical reasoning, all methods are trained on MetaMath (Yu et al., 2023) and evaluated on GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b). For code generation, the models are trained on CodeFeedback (Zheng et al., 2024) and evaluated on HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). The results are summarized in Table 4.

Table 4: **Comparison on mathematical and code generation benchmarks on Qwen2.5-7B.** SCALAR achieves comparable performance on mathematical reasoning and code generation.

Method	Params (%)	GSM8K	MATH	Avg.	HumanEval	MBPP	Avg.
FT	100.00	84.6	48.8	66.7	74.4	82.5	78.4
LoRA ($r = 1$)	0.0331	84.8	48.3	66.6	74.4	81.5	78.0
LoRA ($r = 8$)	0.2643	84.8	48.2	66.5	70.1	83.3	76.7
VeRA	0.0189	84.4	48.7	66.6	72.6	82.5	77.6
NOLA	0.0013	84.3	47.6	66.0	73.2	82.0	77.6
LoRA-XS	0.0014	83.0	47.1	65.1	72.6	81.0	76.8
RandLoRA	0.0175	84.3	48.0	66.2	72.0	84.4	78.2
SCALAR ($K = 128$)	0.0003	83.5	48.5	66.0	73.2	81.7	77.5
SCALAR ($K = 256$)	0.0006	84.3	48.5	66.4	76.8	82.0	**79.4**
SCALAR ($K = 512$)	0.0013	85.0	48.7	**66.9**	73.8	84.1	79.0

Across both domains, SCALAR remains competitive with full fine-tuning and LoRA despite using substantially fewer trainable parameters. The 0.0013% configuration obtains the highest reported mathematical average, while the 0.0006% configuration obtains the highest reported code-generation average. Although the margins are small, the results indicate that the compact scalar-only parameterization transfers beyond commonsense reasoning to mathematical and code adaptation.

5.3 On the Initialization Methods

We investigate the effects of different initialization strategies for the input-side basis $\boldsymbol{A}$ and the output-side dictionary $\boldsymbol{B}$. As shown in Table 5, activation-SVD consistently yields the strongest initialization for $\boldsymbol{A}$, outperforming both random and weight-SVD initialization under either choice of $\boldsymbol{B}$. This result supports our design of extracting $\boldsymbol{A}$ from the dominant activation subspace, which better captures the downstream input geometry.

Table 5: **Commonsense reasoning Acc. on Qwen2.5-3B with different initialization methods.**

	Random-$\boldsymbol{A}$	Weight-SVD-$\boldsymbol{A}$	Activation-SVD-$\boldsymbol{A}$
Random-$\boldsymbol{B}$	84.3	84.4	**85.2**
Weight-SVD-$\boldsymbol{B}$	84.5	84.6	85.1

In contrast, weight-SVD initialization of $\boldsymbol{B}$ provides no consistent advantage over random initialization. The combination of activation-SVD $\boldsymbol{A}$ and random $\boldsymbol{B}$ achieves the best overall accuracy. Moreover, random $\boldsymbol{B}$ can be shared across layers of the same module type, whereas weight-SVD $\boldsymbol{B}$ is layer-specific and therefore incurs higher storage and memory overhead (see Appendix C.6.1). We therefore adopt activation-SVD $\boldsymbol{A}$ and random, shareable $\boldsymbol{B}$ as our default configuration.

6 Conclusion

In this paper, we introduce SCALAR, a PEFT method that separates the construction of adaptation directions from the learning of their magnitudes. SCALAR fixes a layer-specific activation-SVD input basis, pairs each direction with multiple random output atoms, and optimizes only scalar coefficients. Our analysis motivates the activation-derived subspace through activation coverage and the initial gradient signal. Across commonsense reasoning, mathematical reasoning, and code generation, SCALAR provides a strong parameter–performance trade-off relative to representative PEFT baselines. System measurements further show that SCALAR achieves lower training memory consumption and higher throughput than all evaluated PEFT baselines, demonstrating that its parameter reduction translates into practical system efficiency rather than only a smaller nominal adapter size. The initialization and structural ablations further support activation-SVD input directions and random output atoms as a favorable balance between task awareness and efficiency. The current method requires a calibration pass and restricts each layer to a fixed input subspace; studying adaptive calibration and richer fixed dictionaries remains a promising direction for future work.

AI USE STATEMENT

Generative AI tools were used during manuscript preparation for language editing and as an auxiliary tool in parts of the theoretical analysis, including discussing derivation steps, checking proof arguments, and improving the presentation of theoretical results. The core theoretical ideas, mathematical claims, assumptions, and final proofs were developed and independently verified by the authors. Generative AI was not used to conduct experiments, process experimental data, or interpret experimental results. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final manuscript and its contents.

ETHICS STATEMENT

This work focuses on improving the efficiency of parameter-efficient fine-tuning for large language models. It does not involve human subjects, private or personally identifiable data, or the collection of new datasets. The proposed method is intended as a general-purpose optimization technique and does not introduce application-specific ethical risks beyond those associated with adapting and deploying large language models more broadly.

REPRODUCIBILITY STATEMENT

We provide the information required to reproduce the proposed method and experiments throughout the paper and appendix. The method section specifies the parameterization, activation-based calibration procedure, scaling factors, and trainable variables. The experimental sections report the models, datasets, target modules, optimization settings, parameter budgets, and evaluation metrics used for each task. Additional implementation details, hyperparameter ranges, system configurations, dataset descriptions, and ablation settings are provided in the appendix. The assumptions and complete derivations supporting the theoretical results are also included there. To further facilitate reproducibility, we intend to publicly release the implementation and configuration files upon acceptance.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Hn5eoTunHN.

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

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. Lora-xs: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

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

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. *arXiv preprint arXiv:2603.12228*, 2026.

Ziqi Gao, Qichao Wang, Aochuan Chen, Zijing Liu, Bingzhe Wu, Liang Chen, and Jia Li. Parameter-efficient fine-tuning with discrete fourier transform. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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

Yuxuan Gu, Wuyang Zhou, Giorgos Iacovides, and Danilo Mandic. Tera: Vector-based random tensor network for high-rank adaptation of large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2314–2329, 2026.

Chenlu Guo, Yi Chang, and Yuan Wu. NLoRA: Nyström-initiated low-rank adaptation for large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 1371–1385, Suzhou, China, November 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.findings-emnlp.72/.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53 (2):217–288, 2011.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

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.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Soroush Abbasi Koohpayegani, Navaneet K L, Parsa Nooralinejad, Soheil Kolouri, and Hamed Pirsiavash. NOLA: Compressing lora using linear combination of random basis. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=TjfXcDgvzk.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024.

Weicheng Lin, Yi Zhang, Jiawei Dang, and Liang-Jie Zhang. Tlora: Task-aware low rank adaptation of large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29252–29269, 2026.

Vijay Lingam, Atula Tejaswi Neerkaje, Aditya Vavre, Aneesh Shetty, Gautham Krishna Gudur, Joydeep Ghosh, Eunsol Choi, Alex Dimakis, Aleksandar Bojchevski, and sujay sanghavi. SVFT: Parameter-efficient fine-tuning with singular vectors. In *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*, 2024.

Kainan Liu, Yong Zhang, Ning Cheng, Yun Zhu, Yanmeng Wang, Shaojun Wang, and Jing Xiao. Astra: Activation-space tail-eigenvector low-rank adaptation of large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 31833–31854, 2026.

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 *Forty-first International Conference on Machine Learning*, 2024.

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.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

John X Morris, Niloofar Mireshghallah, Mark Ibrahim, and Saeed Mahloujifar. Learning to reason in 13 parameters. *arXiv preprint arXiv:2602.04118*, 2026.

Parsa Nooralinejad, Ali Abbasi, Soroush Abbasi Koohpayegani, Kossar Pourahmadi Meibodi, Rana Muhammad Shahroz Khan, Soheil Kolouri, and Hamed Pirsiavash. Pranc: Pseudo random networks for compacting deep models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17021–17031, 2023.

Fabian Paischer, Lukas Hauzenberger, Thomas Schmied, Benedikt Alkin, Marc Peter Deisenroth, and Sepp Hochreiter. Parameter efficient fine-tuning via explained variance adaptation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=movsqor65f.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Gemma Team. Gemma 3. 2025. URL https://arxiv.org/abs/2503.19786.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37: 9565–9584, 2024.

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

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

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024a. doi: 10.48550/ARXIV.2412.15115. URL https://doi.org/10.48550/arXiv.2412.15115.

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

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

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.

Ruiyi Zhang, Rushi Qiang, Sai Ashish Somayajula, and Pengtao Xie. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5048–5060, 2024.

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

APPENDIX CONTENTS

Appendix Contents **14**

A Proofs and Supporting Analysis **15**
A.1 Proof of Theorem 3.6 . . . 15
A.2 Proof of Theorem 3.7 . . . 16
A.3 Proof of Theorem 3.8 . . . 17
A.4 Gradient Signal of the Scalar Coefficients . . . 20

B Implementation Details **20**
B.1 Implementation Optimizations . . . 21
B.1.1 Optional Sharing of Output Dictionaries . . . 21
B.1.2 Fast SVD for Input Calibration . . . 21
B.1.3 Variance-Preserving Atom Scaling . . . 22
B.1.4 Efficient Adapter Forward Pass . . . 22
B.2 Hyperparameters . . . 22
B.3 Baseline Implementations . . . 23
B.4 Complete Commonsense Reasoning Results . . . 24

C Additional Results and Ablations **24**
C.1 Learned-Direction Reweighting . . . 24
C.2 Does LoRA Learn High-Energy Input Subspaces? . . . 26
C.3 Comparison with Enhanced LoRA Variants . . . 27
C.4 General Instruction Fine-tuning . . . 28
C.5 Scaling to Larger Models . . . 29
C.6 Ablation Study . . . 30
C.6.1 Grouped Parameterization and Cross-Layer Sharing . . . 30
C.6.2 Effect of the Update Rank . . . 31
C.6.3 Effect of the Calibration Size . . . 32
C.6.4 Effect of the Output-Atom Initialization . . . 32

D Adapter Storage Analysis **33**

E Benchmark Details **34**
E.1 Fine-Tuning Datasets . . . 34
E.2 Evaluation Benchmarks . . . 35

A Proofs and Supporting Analysis

This section provides the proofs for the theoretical results in Section 3 and an auxiliary analysis connecting the activation-derived input subspace to the gradients of SCALAR's trainable coefficients.

A.1 Proof of Theorem 3.6

Proof. Using $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top] = \boldsymbol{\Sigma}$ and cyclic invariance of the trace,

$$\begin{aligned}\Phi(\boldsymbol{A}) &= \mathbb{E}\big[\|\boldsymbol{A}\boldsymbol{x}\|_2^2\big] = \mathbb{E}\big[\boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x}\big] \\ &= \mathbb{E}\big[\mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{A}^\top\right)\big] = \mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top\right).\end{aligned}$$

Let

$$\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top, \qquad \boldsymbol{\Lambda} = \mathrm{Diag}(\lambda_1, \dots, \lambda_n), \qquad \lambda_1 \geq \cdots \geq \lambda_n \geq 0,$$

and define $\boldsymbol{C} := \boldsymbol{A}\boldsymbol{U}$. Since $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}_R$ and $\boldsymbol{U}$ is orthogonal,

$$\begin{aligned}\boldsymbol{C}\boldsymbol{C}^\top &= \boldsymbol{A}\boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{A}^\top = \boldsymbol{I}_R, \\ \Phi(\boldsymbol{A}) &= \mathrm{Tr}\left(\boldsymbol{C}\boldsymbol{\Lambda}\boldsymbol{C}^\top\right) = \sum_{i=1}^{n} \lambda_i \|\boldsymbol{c}_i\|_2^2,\end{aligned}$$

where $\boldsymbol{c}_i$ is the i-th column of $\boldsymbol{C}$.

Set

$$q_i := \|\boldsymbol{c}_i\|_2^2, \qquad \boldsymbol{P} := \boldsymbol{C}^\top \boldsymbol{C}.$$

Then

$$\begin{aligned}\boldsymbol{P}^2 &= \boldsymbol{C}^\top \boldsymbol{C}\boldsymbol{C}^\top \boldsymbol{C} = \boldsymbol{C}^\top \boldsymbol{C} = \boldsymbol{P}, \\ q_i &= \boldsymbol{e}_i^\top \boldsymbol{P}\boldsymbol{e}_i, \qquad 0 \leq q_i \leq 1, \\ \sum_{i=1}^{n} q_i &= \mathrm{Tr}(\boldsymbol{P}) = \mathrm{Tr}(\boldsymbol{C}\boldsymbol{C}^\top) = R.\end{aligned}$$

Thus,

$$\begin{aligned}\Phi(\boldsymbol{A}) &= \sum_{i=1}^{R} \lambda_i q_i + \sum_{i=R+1}^{n} \lambda_i q_i \leq \sum_{i=1}^{R} \lambda_i q_i + \lambda_R \sum_{i=R+1}^{n} q_i \\ &= \sum_{i=1}^{R} \lambda_i q_i + \lambda_R \left(R - \sum_{i=1}^{R} q_i\right) = R\lambda_R + \sum_{i=1}^{R} (\lambda_i - \lambda_R) q_i \qquad (7) \\ &\leq R\lambda_R + \sum_{i=1}^{R} (\lambda_i - \lambda_R) = \sum_{i=1}^{R} \lambda_i.\end{aligned}$$

For $\boldsymbol{A}_{\mathrm{svd}} = \boldsymbol{U}_R^\top$,

$$\begin{aligned}\boldsymbol{A}_{\mathrm{svd}} \boldsymbol{\Sigma} \boldsymbol{A}_{\mathrm{svd}}^\top &= \mathrm{Diag}(\lambda_1, \dots, \lambda_R), \\ \Phi(\boldsymbol{A}_{\mathrm{svd}}) &= \sum_{i=1}^{R} \lambda_i,\end{aligned}$$

so the bound in equation 7 is attained.

For the uniformly random R-dimensional row subspace, let

$$\boldsymbol{P}_{\mathrm{rand}} := \boldsymbol{A}_{\mathrm{rand}}^\top \boldsymbol{A}_{\mathrm{rand}}.$$

This is a rank-R orthogonal projector. Rotational invariance gives, for every orthogonal $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$,

$$\begin{aligned}\boldsymbol{P}_{\mathrm{rand}} &\overset{\mathrm{d}}{=} \boldsymbol{Q}^\top \boldsymbol{P}_{\mathrm{rand}} \boldsymbol{Q}, \\ \boldsymbol{S} := \mathbb{E}[\boldsymbol{P}_{\mathrm{rand}}] &= \boldsymbol{Q}^\top \boldsymbol{S}\boldsymbol{Q}, \qquad \forall\, \boldsymbol{Q} \in O(n).\end{aligned}$$

Hence $\boldsymbol{S} = a\boldsymbol{I}_n$ for some scalar a, and

$$an = \mathrm{Tr}(\boldsymbol{S}) = \mathbb{E}\left[\mathrm{Tr}(\boldsymbol{P}_{\mathrm{rand}})\right] = R, \text{and } \mathbb{E}[\boldsymbol{P}_{\mathrm{rand}}] = \frac{R}{n}\boldsymbol{I}_n.$$

Therefore,

$$\begin{aligned}\mathbb{E}_{\boldsymbol{A}_{\mathrm{rand}}}\left[\Phi(\boldsymbol{A}_{\mathrm{rand}})\right] &= \mathbb{E}\left[\mathrm{Tr}\left(\boldsymbol{A}_{\mathrm{rand}}\boldsymbol{\Sigma}\boldsymbol{A}_{\mathrm{rand}}^{\top}\right)\right] \\ &= \mathbb{E}\left[\mathrm{Tr}\left(\boldsymbol{P}_{\mathrm{rand}}\boldsymbol{\Sigma}\right)\right] \\ &= \mathrm{Tr}\left(\mathbb{E}[\boldsymbol{P}_{\mathrm{rand}}]\boldsymbol{\Sigma}\right) \\ &= \frac{R}{n}\mathrm{Tr}(\boldsymbol{\Sigma}) = \frac{R}{n}\sum_{i=1}^{n}\lambda_i.\end{aligned} \tag{8}$$

Finally, for $R < n$,

$$\frac{1}{R}\sum_{i=1}^{R}\lambda_i \geq \frac{1}{n-R}\sum_{i=R+1}^{n}\lambda_i \geq \frac{1}{n}\sum_{i=1}^{n}\lambda_i \Longrightarrow \sum_{i=1}^{R}\lambda_i \geq \frac{R}{n}\sum_{i=1}^{n}\lambda_i.$$

Together with equation 7 and equation 8,

$$\Phi(\boldsymbol{A}_{\mathrm{svd}}) = \sum_{i=1}^{R}\lambda_i \geq \mathbb{E}_{\boldsymbol{A}_{\mathrm{rand}}}\left[\Phi(\boldsymbol{A}_{\mathrm{rand}})\right].$$

Equality for $R < n$ requires equality between the average of the first R eigenvalues and that of the remaining $n - R$ eigenvalues. Since $\lambda_1 \geq \cdots \geq \lambda_n$, this occurs only when $\lambda_1 = \cdots = \lambda_n$, equivalently $\boldsymbol{\Sigma} = a\boldsymbol{I}_n$ for some $a \geq 0$. Hence the comparison is strict whenever $R < n$ and $\boldsymbol{\Sigma}$ is not a scalar multiple of $\boldsymbol{I}_n$.

The proof is completed. □

A.2 Proof of Theorem 3.7

Proof. At $\boldsymbol{B} = \boldsymbol{0}$, for an arbitrary perturbation $\mathrm{d}\boldsymbol{B}$,

$$\begin{aligned}&\mathrm{d}\boldsymbol{z} = \mathrm{d}\boldsymbol{B}\boldsymbol{A}\boldsymbol{x}, \\ &\mathrm{d}\ell = \boldsymbol{g}^{\top}\mathrm{d}\boldsymbol{z} = \boldsymbol{g}^{\top}\mathrm{d}\boldsymbol{B}\boldsymbol{A}\boldsymbol{x} = \mathrm{Tr}\left(\boldsymbol{g}^{\top}\mathrm{d}\boldsymbol{B}\boldsymbol{A}\boldsymbol{x}\right) = \mathrm{Tr}\left[\left(\boldsymbol{g}(\boldsymbol{A}\boldsymbol{x})^{\top}\right)^{\top}\mathrm{d}\boldsymbol{B}\right].\end{aligned}$$

Hence

$$\begin{aligned}\nabla_{\boldsymbol{B}}\ell &= \boldsymbol{g}(\boldsymbol{A}\boldsymbol{x})^{\top}, \\ \|\nabla_{\boldsymbol{B}}\ell\|_F^2 &= \left\|\boldsymbol{g}(\boldsymbol{A}\boldsymbol{x})^{\top}\right\|_F^2 = \|\boldsymbol{g}\|_2^2\|\boldsymbol{A}\boldsymbol{x}\|_2^2.\end{aligned}$$

Therefore, using equation 3,

$$\begin{aligned}\mathcal{G}(\boldsymbol{A}) &= \mathbb{E}\left[\|\nabla_{\boldsymbol{B}}\ell\|_F^2\right] = \mathbb{E}\left[\|\boldsymbol{g}\|_2^2\|\boldsymbol{A}\boldsymbol{x}\|_2^2\right] = \mathbb{E}\left[\|\boldsymbol{g}\|_2^2\boldsymbol{x}^{\top}\boldsymbol{A}^{\top}\boldsymbol{A}\boldsymbol{x}\right] \\ &= \mathrm{Tr}\left(\boldsymbol{A}\mathbb{E}\left[\|\boldsymbol{g}\|_2^2\boldsymbol{x}\boldsymbol{x}^{\top}\right]\boldsymbol{A}^{\top}\right) = \mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{M}\boldsymbol{A}^{\top}\right).\end{aligned} \tag{9}$$

Since the row-orthonormal feasible set is compact and $\mathcal{G}$ is continuous, a maximizer $\boldsymbol{A}_*$ exists. Define

$$\boldsymbol{E} := \boldsymbol{M} - c\boldsymbol{\Sigma}.$$

By Assumption 3.4, $\|\boldsymbol{E}\|_2 \leq \varepsilon$. Since $\boldsymbol{E}$ is symmetric, for any feasible $\boldsymbol{A}$,

$$\begin{aligned}&-\varepsilon\boldsymbol{I}_n \preceq \boldsymbol{E} \preceq \varepsilon\boldsymbol{I}_n \\ &\quad\Longrightarrow -\varepsilon\boldsymbol{A}\boldsymbol{A}^{\top} \preceq \boldsymbol{A}\boldsymbol{E}\boldsymbol{A}^{\top} \preceq \varepsilon\boldsymbol{A}\boldsymbol{A}^{\top} \\ &\quad\Longrightarrow -\varepsilon\boldsymbol{I}_R \preceq \boldsymbol{A}\boldsymbol{E}\boldsymbol{A}^{\top} \preceq \varepsilon\boldsymbol{I}_R \\ &\quad\Longrightarrow \left|\mathrm{Tr}\left(\boldsymbol{A}\boldsymbol{E}\boldsymbol{A}^{\top}\right)\right| \leq R\varepsilon.\end{aligned} \tag{10}$$

Using $\boldsymbol{M} = c\boldsymbol{\Sigma} + \boldsymbol{E}$, Theorem 3.6, and equation 10,

$$\begin{aligned}\mathcal{G}(\boldsymbol{A}_*) - \mathcal{G}(\boldsymbol{A}_{\text{svd}}) &= c\left[\operatorname{Tr}\left(\boldsymbol{A}_*\boldsymbol{\Sigma}\boldsymbol{A}_*^\top\right) - \operatorname{Tr}\left(\boldsymbol{A}_{\text{svd}}\boldsymbol{\Sigma}\boldsymbol{A}_{\text{svd}}^\top\right)\right] \\ &\quad + \operatorname{Tr}\left(\boldsymbol{A}_*\boldsymbol{E}\boldsymbol{A}_*^\top\right) - \operatorname{Tr}\left(\boldsymbol{A}_{\text{svd}}\boldsymbol{E}\boldsymbol{A}_{\text{svd}}^\top\right) \\ &\le \operatorname{Tr}\left(\boldsymbol{A}_*\boldsymbol{E}\boldsymbol{A}_*^\top\right) - \operatorname{Tr}\left(\boldsymbol{A}_{\text{svd}}\boldsymbol{E}\boldsymbol{A}_{\text{svd}}^\top\right) \\ &\le 2R\varepsilon.\end{aligned} \tag{11}$$

Since $\boldsymbol{A}_*$ maximizes $\mathcal{G}$,

$$0 \le \mathcal{G}(\boldsymbol{A}_*) - \mathcal{G}(\boldsymbol{A}_{\text{svd}}) \le 2R\varepsilon.$$

In particular, when $\varepsilon = 0$,

$$\mathcal{G}(\boldsymbol{A}_*) = \mathcal{G}(\boldsymbol{A}_{\text{svd}}),$$

so $\boldsymbol{A}_{\text{svd}}$ is also a maximizer.

For a uniformly random R-dimensional row subspace, let

$$\boldsymbol{P}_{\text{rand}} := \boldsymbol{A}_{\text{rand}}^\top \boldsymbol{A}_{\text{rand}}.$$

From the proof of Theorem 3.6,

$$\mathbb{E}\left[\boldsymbol{P}_{\text{rand}}\right] = \frac{R}{n}\boldsymbol{I}_n.$$

Thus,

$$\begin{aligned}\mathbb{E}_{\boldsymbol{A}_{\text{rand}}}\left[\mathcal{G}(\boldsymbol{A}_{\text{rand}})\right] &= \mathbb{E}\left[\operatorname{Tr}\left(\boldsymbol{A}_{\text{rand}}\boldsymbol{M}\boldsymbol{A}_{\text{rand}}^\top\right)\right] = \mathbb{E}\left[\operatorname{Tr}\left(\boldsymbol{P}_{\text{rand}}\boldsymbol{M}\right)\right] \\ &= \operatorname{Tr}\left(\mathbb{E}\left[\boldsymbol{P}_{\text{rand}}\right]\boldsymbol{M}\right) = \frac{R}{n}\operatorname{Tr}(\boldsymbol{M}).\end{aligned}$$

The proof is completed. □

A.3 Proof of Theorem 3.8

Proof. For each $t \ge 0$, define

$$\boldsymbol{Q}^t := \boldsymbol{A}^t(\boldsymbol{A}^t)^\top$$

and

$$\boldsymbol{H}^t := \boldsymbol{A}^t\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top.$$

Because $\boldsymbol{\Sigma}$ is symmetric, $\boldsymbol{H}^t$ is symmetric for all $t \ge 0$:

$$(\boldsymbol{H}^t)^\top = \boldsymbol{H}^t.$$

Differentiating $\boldsymbol{Q}^t$ with respect to t and applying the product rule gives

$$\frac{\mathrm{d}\boldsymbol{Q}^t}{\mathrm{d}t} = \frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}(\boldsymbol{A}^t)^\top + \boldsymbol{A}^t\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top.$$

By equation 4,

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}(\boldsymbol{A}^t)^\top &= \left[\boldsymbol{A}^t\boldsymbol{\Sigma} - \boldsymbol{H}^t\boldsymbol{A}^t\right](\boldsymbol{A}^t)^\top \\ &= \boldsymbol{H}^t - \boldsymbol{H}^t\boldsymbol{Q}^t.\end{aligned}$$

Taking the transpose of equation 4 and using the symmetry of $\boldsymbol{\Sigma}$ and $\boldsymbol{H}^t$ gives

$$\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top = \boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top - (\boldsymbol{A}^t)^\top\boldsymbol{H}^t.$$

Multiplying this equality on the left by $\boldsymbol{A}^t$ yields

$$\begin{aligned}\boldsymbol{A}^t\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top &= \boldsymbol{A}^t\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top - \boldsymbol{A}^t(\boldsymbol{A}^t)^\top\boldsymbol{H}^t \\ &= \boldsymbol{H}^t - \boldsymbol{Q}^t\boldsymbol{H}^t.\end{aligned}$$

Combining the preceding identities gives

$$\frac{\mathrm{d}\boldsymbol{Q}^t}{\mathrm{d}t} = 2\boldsymbol{H}^t - \boldsymbol{H}^t\boldsymbol{Q}^t - \boldsymbol{Q}^t\boldsymbol{H}^t.$$

Define the deviation from row orthonormality as

$$\boldsymbol{D}^t := \boldsymbol{Q}^t - \boldsymbol{I}_R.$$

Since $\boldsymbol{Q}^t = \boldsymbol{I}_R + \boldsymbol{D}^t$, substitution into the previous differential equation gives

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{D}^t}{\mathrm{d}t} &= 2\boldsymbol{H}^t - \boldsymbol{H}^t\left(\boldsymbol{I}_R + \boldsymbol{D}^t\right) - \left(\boldsymbol{I}_R + \boldsymbol{D}^t\right)\boldsymbol{H}^t \\ &= -\boldsymbol{H}^t\boldsymbol{D}^t - \boldsymbol{D}^t\boldsymbol{H}^t.\end{aligned}$$

The trajectory $\boldsymbol{A}^t$ is continuously differentiable by Assumption 3.5; hence, $\boldsymbol{H}^t$ is continuous in t. The preceding equation is therefore a linear ordinary differential equation in the entries of $\boldsymbol{D}^t$ and has a unique solution for each initial value. Assumption 3.5 gives

$$\begin{aligned}\boldsymbol{D}^0 &= \boldsymbol{A}^0(\boldsymbol{A}^0)^\top - \boldsymbol{I}_R \\ &= \boldsymbol{0}.\end{aligned}$$

The identically zero trajectory $\boldsymbol{D}^t = \boldsymbol{0}$ satisfies both the differential equation and this initial condition. Uniqueness therefore implies

$$\boldsymbol{D}^t = \boldsymbol{0}$$

for every $t \geq 0$. Consequently,

$$\boldsymbol{A}^t(\boldsymbol{A}^t)^\top = \boldsymbol{I}_R$$

for every $t \geq 0$.

The captured activation energy at time t is

$$\Phi(\boldsymbol{A}^t) = \operatorname{Tr}\left(\boldsymbol{A}^t\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right).$$

Differentiating this expression and applying the product rule gives

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\Phi(\boldsymbol{A}^t) = \operatorname{Tr}&\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right) \\ &+ \operatorname{Tr}\left(\boldsymbol{A}^t\boldsymbol{\Sigma}\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top\right).\end{aligned}$$

The second trace equals the first because the trace of a scalar is unchanged by transposition and $\boldsymbol{\Sigma}$ is symmetric:

$$\operatorname{Tr}\left(\boldsymbol{A}^t\boldsymbol{\Sigma}\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top\right) = \operatorname{Tr}\left[\left(\boldsymbol{A}^t\boldsymbol{\Sigma}\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\right)^\top\right)^\top\right] = \operatorname{Tr}\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right).$$

It follows that

$$\frac{\mathrm{d}}{\mathrm{d}t}\Phi(\boldsymbol{A}^t) = 2\operatorname{Tr}\left(\frac{\mathrm{d}\boldsymbol{A}^t}{\mathrm{d}t}\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right). \tag{12}$$

Substituting equation 4 and using the definition of $\boldsymbol{H}^t$ gives

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\Phi(\boldsymbol{A}^t) &= 2\operatorname{Tr}\left[\left(\boldsymbol{A}^t\boldsymbol{\Sigma} - \boldsymbol{H}^t\boldsymbol{A}^t\right)\boldsymbol{\Sigma}(\boldsymbol{A}^t)^\top\right] \\ &= 2\operatorname{Tr}\left(\boldsymbol{A}^t\boldsymbol{\Sigma}^2(\boldsymbol{A}^t)^\top\right) - 2\operatorname{Tr}\left((\boldsymbol{H}^t)^2\right).\end{aligned}$$

Define the projector onto the row space of $\boldsymbol{A}^t$ by

$$\boldsymbol{P}^t := (\boldsymbol{A}^t)^\top\boldsymbol{A}^t.$$

The already established identity $\boldsymbol{A}^t(\boldsymbol{A}^t)^\top = \boldsymbol{I}_R$ implies

$$\begin{aligned}(\boldsymbol{P}^t)^2 &= (\boldsymbol{A}^t)^\top \boldsymbol{A}^t (\boldsymbol{A}^t)^\top \boldsymbol{A}^t \\ &= (\boldsymbol{A}^t)^\top \boldsymbol{I}_R \boldsymbol{A}^t \\ &= \boldsymbol{P}^t.\end{aligned}$$

Since $\boldsymbol{P}^t$ is also symmetric, both $\boldsymbol{P}^t$ and $\boldsymbol{I}_n - \boldsymbol{P}^t$ are orthogonal projectors.

By the definition of the Frobenius norm,

$$\begin{aligned}&\left\| \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right\|_F^2 \\ &= \operatorname{Tr}\left[\left(\left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right)^\top \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right] \\ &= \operatorname{Tr}\left[\boldsymbol{A}^t \boldsymbol{\Sigma} \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right)^\top \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right].\end{aligned}$$

Because $\boldsymbol{I}_n - \boldsymbol{P}^t$ is symmetric and idempotent,

$$\left(\boldsymbol{I}_n - \boldsymbol{P}^t\right)^\top \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) = \boldsymbol{I}_n - \boldsymbol{P}^t.$$

Therefore,

$$\begin{aligned}&\left\| \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right\|_F^2 \\ &= \operatorname{Tr}\left[\boldsymbol{A}^t \boldsymbol{\Sigma} \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right] \\ &= \operatorname{Tr}\left(\boldsymbol{A}^t \boldsymbol{\Sigma}^2 (\boldsymbol{A}^t)^\top \right) - \operatorname{Tr}\left(\boldsymbol{A}^t \boldsymbol{\Sigma} \boldsymbol{P}^t \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right).\end{aligned}$$

Using $\boldsymbol{P}^t = (\boldsymbol{A}^t)^\top \boldsymbol{A}^t$ and the definition of $\boldsymbol{H}^t$, the final trace is

$$\begin{aligned}&\operatorname{Tr}\left(\boldsymbol{A}^t \boldsymbol{\Sigma} \boldsymbol{P}^t \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right) \\ &= \operatorname{Tr}\left(\boldsymbol{A}^t \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \boldsymbol{A}^t \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right) \\ &= \operatorname{Tr}\left((\boldsymbol{H}^t)^2 \right).\end{aligned}$$

Consequently,

$$\begin{aligned}&\left\| \left(\boldsymbol{I}_n - \boldsymbol{P}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right\|_F^2 \\ &= \operatorname{Tr}\left(\boldsymbol{A}^t \boldsymbol{\Sigma}^2 (\boldsymbol{A}^t)^\top \right) - \operatorname{Tr}\left((\boldsymbol{H}^t)^2 \right).\end{aligned}$$

Substituting this identity into the derivative of $\Phi(\boldsymbol{A}^t)$ yields

$$\frac{\mathrm{d}}{\mathrm{d}t} \Phi(\boldsymbol{A}^t) = 2 \left\| \left(\boldsymbol{I}_n - (\boldsymbol{A}^t)^\top \boldsymbol{A}^t\right) \boldsymbol{\Sigma} (\boldsymbol{A}^t)^\top \right\|_F^2 \geq 0.$$

Integrating over $[0, t]$ gives

$$\Phi(\boldsymbol{A}^t) - \Phi(\boldsymbol{A}^0) = 2 \int_0^t \left\| \left(\boldsymbol{I}_n - (\boldsymbol{A}^s)^\top \boldsymbol{A}^s\right) \boldsymbol{\Sigma} (\boldsymbol{A}^s)^\top \right\|_F^2 \, \mathrm{d}s \geq 0.$$

Hence,

$$\Phi(\boldsymbol{A}^t) \geq \Phi(\boldsymbol{A}^0)$$

for every $t \geq 0$.

If $\boldsymbol{A}^0$ represents a uniformly random row subspace, Theorem 3.6 gives

$$\mathbb{E}\left[\Phi(\boldsymbol{A}^0)\right] = \frac{R}{n} \operatorname{Tr}(\boldsymbol{\Sigma}).$$

The inequality $\Phi(\boldsymbol{A}^t) \geq \Phi(\boldsymbol{A}^0)$ holds for every trajectory. Taking expectations therefore yields

$$\mathbb{E}\left[\Phi(\boldsymbol{A}^t)\right] \geq \frac{R}{n} \operatorname{Tr}(\boldsymbol{\Sigma})$$

for every $t \geq 0$.

This completes the proof. □

A.4 Gradient Signal of the Scalar Coefficients

We provide the detailed connection between the activation-derived input basis and the optimization of the scalar coefficients.

For a single adapted linear layer, let $\boldsymbol{x} \in \mathbb{R}^n$ denote its input and $\boldsymbol{g} \in \mathbb{R}^m$ the gradient of the loss with respect to its output. From Eq. equation 5, the gradient of an individual coefficient is

$$\frac{\partial \ell}{\partial \gamma_{r,h}} = s \left(\boldsymbol{b}_{r,h}^\top \boldsymbol{g}\right) \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right). \tag{13}$$

Therefore, the gradient received by each scalar factorizes into the activation along its input direction and the projection of the output gradient onto its random output atom.

Let $\boldsymbol{\Gamma} \in \mathbb{R}^{R \times H}$ collect all trainable scalar coefficients. If the output atoms are independent of $(\boldsymbol{x}, \boldsymbol{g})$ and satisfy

$$\mathbb{E}\left[\boldsymbol{b}_{r,h} \boldsymbol{b}_{r,h}^\top\right] = \sigma_b^2 \boldsymbol{I}_m,$$

then

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{B}} \mathbb{E}\left[\|\nabla_{\boldsymbol{\Gamma}} \ell\|_F^2\right] &= s^2 \sum_{r=1}^{R} \sum_{h=1}^{H} \mathbb{E}\left[\left(\boldsymbol{b}_{r,h}^\top \boldsymbol{g}\right)^2 \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right)^2\right] \\
&= s^2 H \sigma_b^2 \mathbb{E}\left[\|\boldsymbol{g}\|_2^2 \sum_{r=1}^{R} \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right)^2\right] \\
&= s^2 H \sigma_b^2 \mathbb{E}\left[\|\boldsymbol{g}\|_2^2 \|\boldsymbol{A}\boldsymbol{x}\|_2^2\right] \\
&= s^2 H \sigma_b^2 \mathcal{G}(\boldsymbol{A}).
\end{aligned} \tag{14}$$

Hence, under the stated isotropy and independence assumptions, the expected initial gradient energy of the scalar coefficients is proportional to the gradient-signal objective analyzed in Section 3. By Theorem 3.7, the activation-SVD subspace is within $2R\varepsilon$ of the optimal normalized input subspace under the covariance-alignment assumption. This result concerns the initial expected gradient energy and does not claim optimality throughout nonlinear training. Figure 3 provides a complementary empirical diagnostic for the gradients associated with differently initialized input factors.

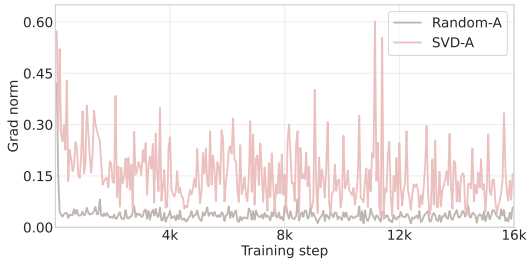


Figure 3: Gradient norms associated with the input-side factor $\boldsymbol{A}$ under activation-SVD and random initialization during Qwen2.5-7B fine-tuning on commonsense reasoning. The two initializations are matched in scale. In this diagnostic run, activation-SVD initialization produces larger gradient norms over most of training; this observation is empirical and is separate from the initial expected-gradient result in Theorem 3.7.

B Implementation Details

This section supplements Section 4 with implementation details, training hyperparameters, baseline implementations, and the complete commonsense reasoning results. Unless stated otherwise, all implementation choices are shared across methods whenever their parameterizations permit.

B.1 Implementation Optimizations

We summarize the main implementation choices used in our experiments. Notation follows Section 4. For each adapted layer, SCALAR maintains a task-aware input matrix $\boldsymbol{A} \in \mathbb{R}^{R\times n}$ and a trainable coefficient matrix $\boldsymbol{\Gamma} \in \mathbb{R}^{R\times H}$, where R controls the maximum rank of the aggregated update and H is the number of random output atoms per input direction. The random output dictionaries $\{\boldsymbol{B}_r\}_{r=1}^{R}$ are frozen and may be shared across layers with the same module type.

B.1.1 Optional Sharing of Output Dictionaries

Because the output atoms are sampled independently of pretrained weights, compatible layers may reuse the same frozen dictionaries $\{\boldsymbol{B}_r\}_{r=1}^{R}$. For example, query-projection layers with the same output dimension can reuse one dictionary while retaining their own activation-derived $\boldsymbol{A}$ and trainable $\boldsymbol{\Gamma}$. We use this option in the default implementation to reduce the storage of frozen random buffers. It is not required by the SCALAR parameterization; the unshared variant is evaluated in Appendix C.6.1.

B.1.2 Fast SVD for Input Calibration

To construct $\boldsymbol{A}$, we collect input activations from a small calibration set. Instead of storing the full activation matrix $\boldsymbol{X} \in \mathbb{R}^{S\times n}$ on the accelerator, we accumulate the Gram matrix

$$\boldsymbol{C} = \boldsymbol{X}^\top \boldsymbol{X}$$

over streaming token blocks. We then extract the leading R right singular directions using randomized eigendecomposition (Halko et al., 2011). This avoids dense SVD on $\boldsymbol{X}$ and is efficient when $R \ll n$. Modules that receive the same hidden state within a transformer layer, such as q_proj, k_proj, and v_proj, share one calibration statistic and one decomposition per layer.

To validate this Fast SVD against a dense (full) eigendecomposition of $\boldsymbol{C}$, we compare both the wall-clock cost of the top-R solve and the recovered R-dimensional subspaces on the same Gram matrices (Qwen2.5-7B; H=16, r=1; averaged over 84 unique input groups). Let $\boldsymbol{V}_{\text{fast}}, \boldsymbol{V}_{\text{full}} \in \mathbb{R}^{n\times R}$ collect the top-R right singular vectors, and let $\boldsymbol{Q}_{\text{fast}}, \boldsymbol{Q}_{\text{full}}$ be orthonormal bases of their column spaces obtained by QR. The principal angles $\{\theta_i\}_{i=1}^{R}$ satisfy

$$\cos\theta_i = \sigma_i\big(\boldsymbol{Q}_{\text{fast}}^\top \boldsymbol{Q}_{\text{full}}\big), \qquad i = 1, \ldots, R, \tag{15}$$

where $\sigma_i(\cdot)$ denotes the i-th singular value. We report the mean cosine alignment

$$\overline{\cos} = \frac{1}{R}\sum_{i=1}^{R} \cos\theta_i \tag{16}$$

and the principal-angle affinity $\frac{1}{R}\sum_{i=1}^{R}\cos^2\theta_i$. In this implementation, Table 6 shows that the randomized solver reduces the measured top-R eigensolve time by 19.6× on the collected Gram matrices. The recovered subspaces are also close under the reported principal-angle metrics ($\overline{\cos} = 0.998$). These measurements characterize this calibration setting and should not be interpreted as a hardware-independent speedup.

Table 6: Fast (randomized) vs. full eigendecomposition for activation Gram $\boldsymbol{C}$ on Qwen2.5-7B ($R = 16$). Time is the summed top-R solve over all groups; subspace metrics compare Fast and full on the *same* Grams.

	Fast SVD	Full SVD
Eigendecomposition time (s)	6.29	123.0
Speedup vs. full	19.6×	1×
Mean cosine $\overline{\cos}$ equation 16	0.998	
Affinity $\frac{1}{R}\sum_i \cos^2\theta_i$	0.996	
Mean principal angle (deg)	0.500	

B.1.3 VARIANCE-PRESERVING ATOM SCALING

SCALAR represents each layer update as a sum of $K = RH$ rank-one atoms:

$$\Delta \boldsymbol{W} = s \sum_{r=1}^{R} \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h} \boldsymbol{a}_r^\top .$$

Here, the overall update has rank at most R, while each atom $\boldsymbol{b}_{r,h}\boldsymbol{a}_r^\top$ has rank 1. To keep the update scale stable when R or H changes, we use

$$s = \frac{\alpha}{\sqrt{K}}, \qquad \alpha = 1, \qquad K = RH.$$

This can be interpreted as setting α to the rank of each individual perturbation atom, namely $r = 1$. Unlike LoRA-style scaling, this is not treated as a tunable hyperparameter. It only normalizes the sum over K atoms.

The motivation is variance preservation. Conditional on coefficients and input projections of comparable scale, if the random output atoms are zero-mean, independent, and similarly normalized, the cross terms vanish in expectation and the squared norm of the unscaled atom sum grows linearly with K up to the coefficient- and activation-dependent factor:

$$\mathbb{E}\left[\left\| \sum_{r=1}^{R} \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h} \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right) \right\|_2^2\right] = \mathcal{O}(K).$$

Under these simplifying conditions, the typical update norm grows as $\mathcal{O}(\sqrt{K})$ without normalization. Multiplication by $1/\sqrt{K}$ removes this direct dependence on dictionary size. This argument motivates the scaling rule; it does not assume that trained coefficients are identically distributed across different values of K.

The same normalization also stabilizes the initial optimization scale. At zero-initialized coefficients, the gradient of one scalar is

$$\frac{\partial \ell}{\partial \gamma_{r,h}} = \frac{1}{\sqrt{K}} \left(\boldsymbol{b}_{r,h}^\top \boldsymbol{g}\right) \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right). \tag{17}$$

Therefore, increasing K does not automatically increase the effective adapter step size merely because more atoms are summed.

B.1.4 EFFICIENT ADAPTER FORWARD PASS

The adapter output can be evaluated as

$$\Delta \boldsymbol{W} \boldsymbol{x} = s \sum_{r=1}^{R} \left(\boldsymbol{a}_r^\top \boldsymbol{x}\right) \boldsymbol{B}_r \boldsymbol{\gamma}_r.$$

During training, we first compute

$$\boldsymbol{h} = \boldsymbol{A}\boldsymbol{x},$$

then combine the random output atoms using the trainable coefficients $\boldsymbol{\gamma}_r$. This avoids explicitly forming the full update matrix $\Delta \boldsymbol{W}$ during training. Once training is finished, we may cache the effective output directions

$$\widetilde{\boldsymbol{b}}_r = \boldsymbol{B}_r \boldsymbol{\gamma}_r$$

or merge $\Delta \boldsymbol{W}$ into the pretrained weight, preserving the inference-time behavior of standard LoRA-style adapters.

B.2 HYPERPARAMETERS

We detail the hyperparameters used to reproduce our experimental results. All methods are optimized with AdamW using $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1\text{e}{-8}$. We use BF16 precision to reduce memory consumption and train with Distributed Data Parallel (DDP). Gradient synchronization is implemented using PyTorch's (Paszke et al., 2017) *torch.distributed* framework. Following a widely

used configuration for LLM adaptation, we apply all PEFT methods to the same set of Transformer linear layers. This all-linear setting provides sufficient adaptation capacity and ensures that differences across methods arise from their parameterizations rather than from different target-module selections.

Different PEFT methods have different preferred learning-rate ranges and scaling conventions. For baselines with a LoRA-style scaling factor, we use their default setting, namely $\alpha = \text{rank}$. For `SCALAR`, we use the fixed variance-preserving scaling described above, $s = 1/\sqrt{RH}$, equivalently $\alpha/\sqrt{K}$ with $\alpha = 1$ and $K = RH$. This scaling is fixed across all tasks and models and is not tuned as an additional hyperparameter.

For compactness, define the learning-rate search spaces as

$$\mathcal{S}_{\text{peft}} = \{1.0\text{e}{-5}, 2.5\text{e}{-5}, 5.0\text{e}{-5}, 1.0\text{e}{-4}, 2.5\text{e}{-4}, 5.0\text{e}{-4}, 1.0\text{e}{-3}\},$$

$$\mathcal{S}_{\text{small}} = \{1.0\text{e}{-6}, 2.5\text{e}{-6}, 5.0\text{e}{-6}, 1.0\text{e}{-5}, 2.5\text{e}{-5}, 5.0\text{e}{-5}, 1.0\text{e}{-4}\},$$

and

$$\mathcal{S}_{\text{large}} = \{1.0\text{e}{-4}, 2.5\text{e}{-4}, 5.0\text{e}{-4}, 1.0\text{e}{-3}, 2.5\text{e}{-3}, 5.0\text{e}{-3}, 1.0\text{e}{-2}\}.$$

We tune LoRA, NOLA, LoRA-XS, and `SCALAR` over $\mathcal{S}_{\text{peft}}$, RandLoRA and full-parameter fine-tuning over $\mathcal{S}_{\text{small}}$, and VeRA over $\mathcal{S}_{\text{large}}$.

Table 7: Trainable params (%) with different K on Qwen2.5-7B and LLaMA3-8B.

Model	$K = 128$	$K = 256$	$K = 512$
Qwen2.5-7B	25,088 (0.0003%)	50,176 (0.0006%)	100,352 (0.0013%)
LLaMA3-8B	28,672 (0.0003%)	57,344 (0.0006%)	114,688 (0.0014%)
	$K = 1024$	$K = 2048$	$K = 4096$
Qwen2.5-7B	200,704 (0.0026%)	401,408 (0.0052%)	802,816 (0.0105%)
LLaMA3-8B	229,376 (0.0028%)	458,752 (0.0057%)	917,504 (0.0114%)

All methods are fine-tuned for three epochs with batch size 32 and a cosine learning-rate scheduler. The first 100 steps are used for warm-up. We report results over three random seeds, $(0, 1, 2)$. The maximum sequence length is set to 384, which is sufficient for commonsense reasoning inputs. For our method, we evaluate three default settings: K in $\{1024, 2048, 4096\}$. A larger K corresponds to higher model capacity and a larger number of trainable parameters. The trainable parameter counts for different values of K are reported in Table 7. Table 8 summarizes the main hyperparameter settings.

For mathematical reasoning and code generation, we fine-tune on MetaMath (Yu et al., 2023) and CodeFeedback (Zheng et al., 2024), respectively, for one epoch. We use a batch size of 256, a maximum sequence length of 1024, a 100-step warm-up, and cosine learning-rate decay. The complete hyperparameter settings are provided in Table 9.

B.3 Baseline Implementations

All experiments use a common training stack based on Hugging Face `transformers` and `peft` (v0.18.0), launched with `torchrun` and the Hugging Face `Trainer`. LoRA (Hu et al., 2022) and VeRA (Kopiczko et al., 2024) use the standard PEFT APIs (`LoraConfig` and `VeraConfig`). RandLoRA (Albert et al., 2025) uses the upstream `RandLoraConfig` integration. LoRA-XS (Bałazy et al., 2024) is implemented on top of PEFT LoRA with SVD-derived frozen $\boldsymbol{A}$ and $\boldsymbol{B}$ and a trainable $r \times r$ matrix $\boldsymbol{M}$. NOLA (Koohpayegani et al., 2024) uses an in-house adapter module checked against the reference parameterization because it is not available as a native PEFT tuner. Commonsense reasoning uses generative decoding with regex-based answer extraction following the DoRA/LoRA-XS evaluation style; mathematical reasoning follows the PiSSA setup (Meng et al., 2024); and code generation uses the corresponding prompts with `evalplus` pass@1 on the base test suites.

Table 8: Hyperparameters for fine-tuning LLaMA3-8B and Qwen2.5-7B on commonsense reasoning tasks.

Hyperparameters	FT	LoRA	VeRA	NOLA	LoRA-XS	RandLoRA	SCALAR
Optimizer	AdamW						
LR scheduler	Cosine						
Precision	BF16						
Batch size	32						
Epochs	3						
Warmup steps	100						
Seeds	$\{0, 1, 2\}$						
Max sequence length	384						
Target modules	All linear projection modules						
Rank (r)	-	$\{1, 8\}$	$\{1024, 2048\}$	$\{1, 8\}$	$\{32, 48, 64\}$	$\{2048, 4096\}$	$\{8, 16\}$
Scaling (α)	-	r	Default	r	r	r	equation 17
LR (Qwen)	2.5e−5	1.0e−4	2.5e−3	1.0e−3	5.0e−4	2.5e−5	5.0e−4
LR (LLaMA)	1.0e−5	1.0e−4	2.5e−3	5.0e−4	5.0e−4	2.5e−5	5.0e−4

Table 9: Hyperparameters for fine-tuning Qwen2.5-7B on mathematical reasoning and code generation tasks.

Hyperparameters	FT	LoRA	VeRA	NOLA	LoRA-XS	RandLoRA	SCALAR
LR scheduler	Cosine						
Precision	BF16						
Batch size	256						
Epochs	1						
Warmup steps	100						
Max sequence length	1024						
Target modules	All linear projection modules						
Rank (r)	-	$\{1, 8\}$	256	1	24	4096	16
Scaling (α)	-	r	Default	r	r	r	equation 17
LR (Math)	5.0e−6	1.0e−5	2.5e−4	5.0e−5	2.5e−4	2.5e−5	2.5e−5
LR (Code)	2.5e−6	$\{1.0e{-}5, 1.0e{-}4\}$	5.0e−4	1.0e−4	1.0e−4	2.5e−5	5.0e−4

B.4 Complete Commonsense Reasoning Results

The main text reports a compact comparison at representative parameter budgets. Table 10 provides the corresponding per-task results across the evaluated budgets. Small differences should be interpreted together with the reported variation across runs.

C Additional Results and Ablations

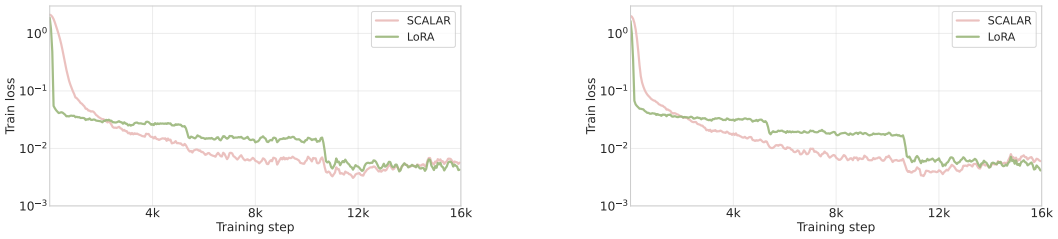
This section provides the diagnostic experiments and ablations referenced in the main text. These results are intended to clarify the behavior of the proposed parameterization rather than to establish universal properties of LoRA or fixed-basis adaptation.

C.1 Learned-Direction Reweighting

This subsection expands the learned-direction reweighting diagnostic introduced in Section 3. For each adapted matrix, we first train a rank-one LoRA adapter, freeze its learned input and output directions, discard the learned magnitude, and initialize a new scalar coefficient. The second stage optimizes only these scalars using the same data and training protocol as the rank-one LoRA run. Because the directions are obtained after LoRA training, this experiment isolates the role of coefficient reweighting but is not itself a practical adaptation method.

Table 10: **Accuracy comparison (3 independent runs) of Qwen2.5-7B and LLaMA3-8B with various PEFT methods on eight commonsense reasoning benchmarks.** This table supplements the main commonsense reasoning table by reporting the performance of each baseline under different trainable parameter budgets.

Model	Method	Params (%)	BoolQ	PIQA	SIQA	HS	WG	ARC-E	ARC-C	OBQA	Avg.
Qwen2.5-7B	FT	100.00	75.3	89.7	82.0	96.1	89.7	95.6	88.4	93.1	$88.7_{\pm 0.1}$
	LoRA ($r = 1$)	0.0331	74.3	90.3	81.9	95.8	88.7	96.1	89.2	91.7	$88.5_{\pm 0.1}$
	LoRA ($r = 8$)	0.2643	74.6	90.4	81.9	96.3	89.3	95.9	89.5	92.3	$88.8_{\pm 0.1}$
	VeRA	0.0208	73.7	90.2	81.5	95.6	87.7	96.3	88.8	91.5	$88.2_{\pm 0.2}$
	VeRA	0.0235	73.6	90.0	81.8	95.8	88.0	96.1	89.5	91.2	$88.2_{\pm 0.1}$
	NOLA	0.0026	73.2	89.3	82.5	95.8	88.0	96.5	89.0	91.2	$88.2_{\pm 0.1}$
	NOLA	0.0052	73.3	89.6	82.1	95.8	88.8	96.6	89.1	91.3	$88.3_{\pm 0.2}$
	NOLA	0.0105	73.6	90.0	82.0	95.8	88.6	96.4	89.2	91.5	$88.4_{\pm 0.0}$
	LoRA-XS	0.0026	72.5	89.2	82.5	95.4	86.1	96.1	89.4	91.5	$87.8_{\pm 0.2}$
	LoRA-XS	0.0052	72.6	89.6	82.7	95.5	86.5	96.5	88.5	91.2	$87.9_{\pm 0.1}$
	LoRA-XS	0.0105	73.4	89.8	82.4	95.6	87.2	96.5	88.5	90.9	$88.0_{\pm 0.2}$
	RandLoRA	0.0175	73.9	89.5	81.2	95.8	88.3	96.1	89.1	90.8	$88.1_{\pm 0.2}$
	RandLoRA	0.0225	73.6	89.6	81.0	95.7	88.3	96.2	89.2	91.8	$88.2_{\pm 0.2}$
	SCALAR ($K = 1024$)	0.0026	73.4	90.3	82.4	96.0	87.5	96.8	89.0	92.3	$\mathbf{88.5}_{\pm 0.1}$
	SCALAR ($K = 2048$)	0.0052	73.7	90.0	82.5	96.1	88.2	96.7	89.2	91.9	$\mathbf{88.6}_{\pm 0.0}$
	SCALAR ($K = 4096$)	0.0105	74.0	90.3	82.2	96.2	89.3	96.5	89.5	93.0	$\mathbf{88.9}_{\pm 0.1}$
LLaMA3-8B	FT	100.00	76.2	90.4	83.0	96.7	89.4	94.1	85.1	88.8	$88.0_{\pm 0.2}$
	LoRA ($r = 1$)	0.0326	74.3	89.9	81.9	96.3	86.5	93.3	83.3	89.0	$87.2_{\pm 0.2}$
	LoRA ($r = 8$)	0.2604	75.9	90.0	83.0	96.6	89.4	93.3	83.1	87.8	$87.4_{\pm 0.3}$
	VeRA	0.0199	74.8	88.9	81.6	95.9	87.1	92.8	82.1	87.5	$86.3_{\pm 0.3}$
	VeRA	0.0228	75.3	89.0	82.4	95.7	88.1	93.3	82.1	85.5	$86.4_{\pm 0.0}$
	NOLA	0.0028	74.0	89.1	81.8	96.2	87.8	93.0	82.7	88.3	$86.6_{\pm 0.3}$
	NOLA	0.0057	74.7	89.5	82.5	96.4	88.4	93.0	83.2	88.2	$87.0_{\pm 0.3}$
	NOLA	0.0114	74.7	89.5	82.8	96.4	88.3	92.9	82.9	87.4	$86.9_{\pm 0.1}$
	LoRA-XS	0.0028	73.0	88.7	81.2	95.5	86.0	91.9	80.7	84.5	$85.1_{\pm 0.0}$
	LoRA-XS	0.0064	73.0	88.6	82.5	95.9	86.9	92.5	81.7	87.1	$86.0_{\pm 0.0}$
	LoRA-XS	0.0114	73.3	88.3	82.7	96.0	88.1	92.4	81.9	87.7	$86.2_{\pm 0.2}$
	RandLoRA	0.0203	75.1	89.5	82.3	95.5	86.7	93.2	81.7	88.4	$86.6_{\pm 0.0}$
	RandLoRA	0.0269	75.4	89.1	82.3	95.6	87.2	92.9	82.5	87.1	$86.5_{\pm 0.2}$
	SCALAR ($K = 1024$)	0.0028	73.5	89.2	82.4	96.2	88.0	92.8	82.3	87.6	$\mathbf{86.5}_{\pm 0.3}$
	SCALAR ($K = 2048$)	0.0057	74.2	89.7	82.5	96.3	87.5	93.3	83.5	88.3	$\mathbf{86.9}_{\pm 0.1}$
	SCALAR ($K = 4096$)	0.0114	74.0	89.6	82.8	96.8	88.8	93.1	83.9	89.2	$\mathbf{87.3}_{\pm 0.1}$



(a) Qwen2.5-7B (b) LLaMA3-8B

Figure 4: **Optimization behavior of rank-one LoRA and learned-direction reweighting.** The reweighting model freezes the directions learned by LoRA and re-optimizes one scalar coefficient for each adapted matrix. It optimizes 224 coefficients for LLaMA3-8B and 196 for Qwen2.5-7B, and follows a similar loss-reduction trajectory in these runs.

Figure 4 shows that the scalar-reweighted model follows a similar training-loss trajectory to rank-one LoRA while optimizing several orders of magnitude fewer parameters. In this diagnostic construction, the learned LoRA directions are sufficient in these runs to recover much of the observed fitting behavior by re-optimizing only their magnitudes. This result does not imply that such directions are available before fine-tuning.

Table 11 reports slightly higher average accuracy for direction reweighting on both evaluated model families. The differences are small and should not be interpreted as evidence of a universal generalization advantage. Since both stages use the same learned directions, the experiment shows that

Table 11: **Average accuracy of rank-one LoRA and learned-direction reweighting on eight commonsense reasoning tasks.** The reweighting model uses fewer than 0.01% as many trainable parameters as rank-one LoRA and attains similar reported averages in these two settings.

Model	Method	Params	Avg. Acc.
LLaMA3-8B	LoRA ($r = 1$)	2,621,440	87.2
	Direction reweighting	224	87.5
Qwen2.5-7B	LoRA ($r = 1$)	2,523,136	88.5
	Direction reweighting	196	88.7

restricting the second stage to scalar magnitudes does not reduce the reported performance in these settings; it does not identify the mechanism behind the small differences.

Together, the loss and accuracy results support a limited but useful observation: after informative rank-one LoRA directions have been learned, re-optimizing their scalar magnitudes can recover similar fitting behavior and downstream accuracy. `SCALAR` addresses the practical part of this observation by constructing an activation-calibrated input subspace before fine-tuning and learning only layer-specific atom weights.

C.2 Does LoRA Learn High-Energy Input Subspaces?

The analysis in Section 3 suggests that learning the input-side factor of LoRA can be interpreted, in part, as learning directions that better align with the activation geometry. We provide a diagnostic experiment to examine whether this behavior appears during practical LoRA fine-tuning.

We train LoRA adapters on a 7B language model. For an adapted linear layer, let $\boldsymbol{A}^t \in \mathbb{R}^{R\times n}$ denote the input-side LoRA factor at training step t. Directly measuring $\|\boldsymbol{A}^t\boldsymbol{x}\|_2^2$ is not reliable because the LoRA factorization is scale-indeterminate:

$$\boldsymbol{B}^t\boldsymbol{A}^t = \left(c^{-1}\boldsymbol{B}^t\right)\left(c\boldsymbol{A}^t\right)$$

for any nonzero scalar c. We therefore remove the scale effect by evaluating the row-normalized representative

$$\overline{\boldsymbol{A}}^t = \left(\boldsymbol{A}^t(\boldsymbol{A}^t)^\top + \epsilon \boldsymbol{I}_R\right)^{-\frac{1}{2}}\boldsymbol{A}^t,$$

where ϵ is a small constant for numerical stability. Left multiplication by the positive-definite factor preserves the row subspace of $\boldsymbol{A}^t$. When $\epsilon = 0$ and $\boldsymbol{A}^t$ has full row rank, the rows of $\overline{\boldsymbol{A}}^t$ are orthonormal and the construction is exactly invariant to positive global rescaling of $\boldsymbol{A}^t$. The small regularizer used in computation makes this invariance approximate near rank-deficient states.

During training, we reuse the input activations from the normal forward pass and compute the captured activation-energy ratio

$$\rho^t = \frac{\mathbb{E}_{\boldsymbol{x}}\left[\left\|\overline{\boldsymbol{A}}^t\boldsymbol{x}\right\|_2^2\right]}{\mathbb{E}_{\boldsymbol{x}}\left[\|\boldsymbol{x}\|_2^2\right]}. \tag{18}$$

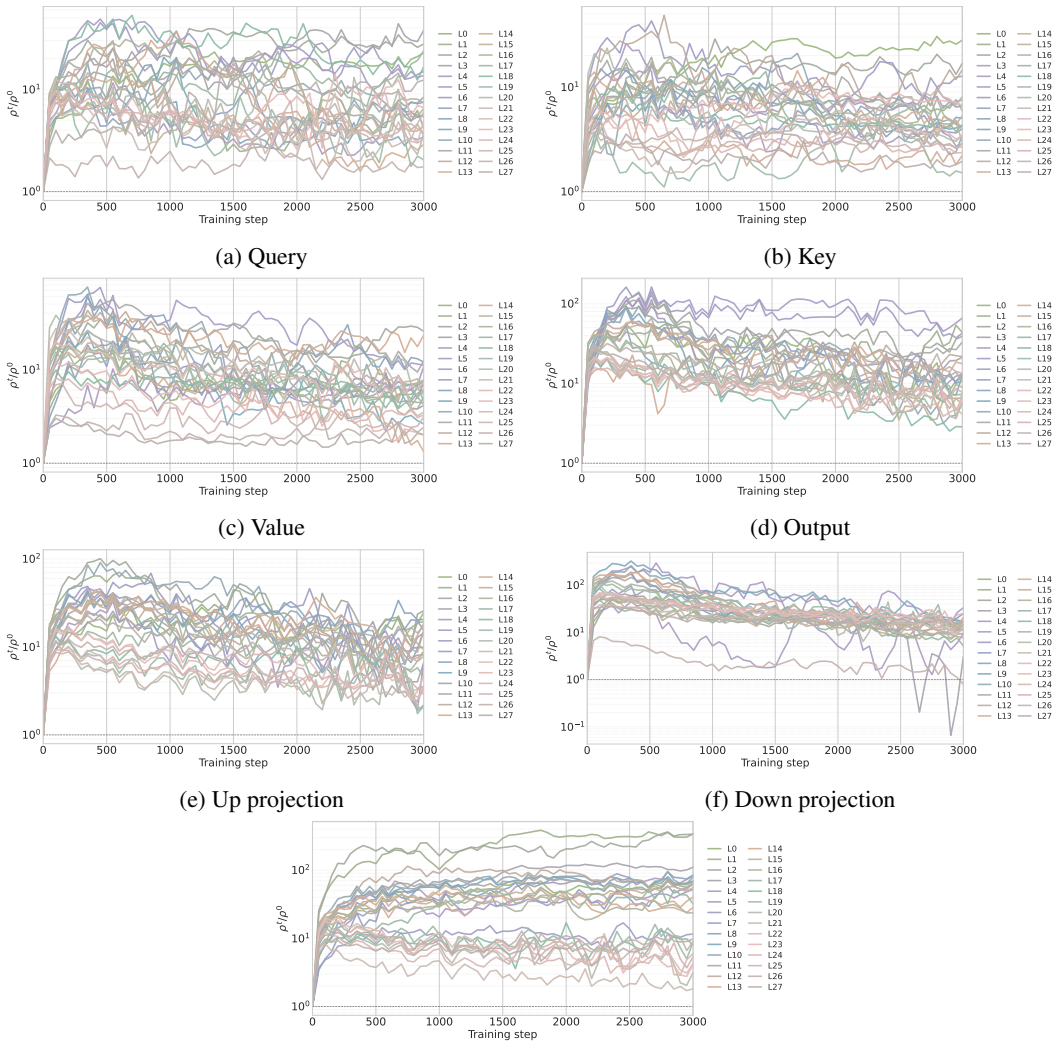
To compare different layers and module types, we report the relative ratio

$$\widehat{\rho}^t = \frac{\rho^t}{\rho^0},$$

where ρ^0 is the value at the random-initialized LoRA state. Thus, $\widehat{\rho}^t > 1$ indicates that the current LoRA input subspace captures more normalized activation energy than its initialization.

Figure 5 reports the diagnostic dynamics of $\widehat{\rho}^t$ across different adapted module types. We do not expect the ratio to increase monotonically in practical training, since LoRA optimizes task loss rather than the activation-coverage objective. Indeed, the module-wise curves contain a small number of outliers; for example, one down-projection layer falls below its random-initialized level during training. Across most modules in this run, the aggregate trend is toward larger normalized activation coverage, although individual layers need not follow that trend. This diagnostic is consistent with

the view that learning $\boldsymbol{A}$ can move the input subspace toward directions that are more active on downstream data; it does not establish that activation coverage is LoRA's optimization objective. SCALAR instead constructs an activation-calibrated input subspace before fine-tuning and allocates its trainable budget to scalar atom weights.



(a) Query (b) Key

(c) Value (d) Output

(e) Up projection (f) Down projection

(g) Gate projection

Figure 5: **Diagnostic training dynamics of LoRA on Qwen2.5-7B.** We report the relative normalized captured activation-energy ratio $\widehat{\rho}^t = \rho^t/\rho^0$, where ρ^t is defined in equation 18. Values above 1 indicate stronger activation alignment than the random-initialized LoRA input subspace. The dynamics are not monotonic and include layer-wise outliers. In this run, most module groups show an aggregate shift toward higher normalized activation coverage, although the behavior is not uniform across layers.

C.3 Comparison with Enhanced LoRA Variants

We further compare SCALAR with DoRA (Liu et al., 2024), PiSSA (Meng et al., 2024), and EVA (Paischer et al., 2025), which represent three complementary improvements over standard LoRA. DoRA introduces a magnitude–direction weight decomposition, PiSSA initializes and optimizes the principal singular components of the pretrained weights, and EVA initializes LoRA from high-variance activation directions with optional adaptive rank allocation. These comparisons place SCALAR alongside methods that improve the LoRA parameterization or its initialization. EVA is

particularly relevant because it also exploits activation geometry; however, unlike EVA, which subsequently trains both LoRA factors, SCALAR freezes the activation-derived input basis and learns only scalar combinations of fixed random output atoms.

All methods are evaluated on Qwen2.5-7B using the same training and evaluation protocol. We use identical adapted modules and LoRA ranks for LoRA, DoRA, PiSSA, and EVA, and independently tune their learning rates using the same search budget.

Table 12: **Comparison with enhanced LoRA variants on Qwen2.5-7B.** Despite using substantially fewer trainable parameters, SCALAR remains competitive with parameterization- and initialization-enhanced LoRA methods.

Method	**Params** (%)	**Avg. Acc.**
LoRA	0.0331	88.5
DoRA	0.0513	88.7
PiSSA	0.0331	88.5
EVA	0.0331	88.7
SCALAR	0.0105	88.9

As shown in Table 12, the enhanced LoRA variants provide only modest improvements over standard LoRA under this setting. DoRA and EVA increase the average accuracy from 88.5 to 88.7, while PiSSA performs comparably to the standard LoRA initialization. In comparison, SCALAR reaches an average accuracy of 88.9 using approximately one-third of the trainable parameters of LoRA, PiSSA, and EVA, and about one-fifth of those required by DoRA.

The comparison with EVA is particularly informative. Although both methods use activation-derived input directions, EVA retains fully trainable low-rank factors, whereas SCALAR fixes the input basis and optimizes only a compact set of scalar coefficients. In this setting, retaining two fully trainable activation-initialized LoRA factors does not yield a higher reported average than SCALAR. The result is consistent with the proposed design: an informative fixed input subspace and a random output dictionary can provide competitive adaptation at a smaller trainable-parameter budget. We do not infer from this single setting that one parameterization uniformly dominates the others.

C.4 General Instruction Fine-tuning

To further evaluate the applicability of SCALAR beyond task-specific fine-tuning, we conduct general-domain instruction tuning on three model families spanning different parameter scales: Gemma3-1B (Team, 2025), LLaMA3.2-3B (Grattafiori et al., 2024), and Qwen2.5-7B (Yang et al., 2024a). For each model, we fine-tune on the 52K-example Alpaca-GPT4 dataset (Peng et al., 2023) for 3 epochs and evaluate the resulting models on MMLU (Hendrycks et al., 2021a) (5-shot). We report accuracy over the four MMLU subject categories, including STEM, humanities, social sciences, and other domains, together with the overall accuracy. Full-parameter fine-tuning, LoRA ($r = 8$), and two parameter configurations of SCALAR use the same training data, number of epochs, target modules, and optimization budget for a fair comparison.

As shown in Table 13, SCALAR remains competitive across heterogeneous model families and parameter scales. On Gemma3-1B, SCALAR achieves an overall MMLU accuracy of 27.13, outperforming LoRA by 0.59 points while using approximately 17$\times$ fewer trainable parameters. On LLaMA3.2-3B, the larger SCALAR configuration reaches 56.79, improving upon LoRA by 0.22 points and remaining within 0.10 points of full-parameter fine-tuning while training only 0.0250% of the model parameters. On Qwen2.5-7B, the smaller SCALAR configuration obtains the best overall accuracy of 74.58, surpassing both LoRA and full-parameter fine-tuning with only 0.0052% trainable parameters.

The preferred parameter configuration varies across model families: the larger configuration performs better on Gemma3-1B and LLaMA3.2-3B, whereas the smaller configuration performs best on Qwen2.5-7B. Nevertheless, both configurations generally remain close to LoRA and full-parameter fine-tuning. These results demonstrate that the fixed-direction scalar adaptation used by

Table 13: **General-domain instruction-tuning results on MMLU.** Params denotes the percentage of trainable parameters. The best parameter-efficient result for each model is highlighted in bold.

Model	Method	Params (%)	STEM	Humanities	Social Sciences	Other	Overall
Gemma3-1B	FT	100.00	27.37	27.04	24.54	25.97	26.33
	LoRA	0.6481	27.85	26.82	24.67	26.65	26.54
	SCALAR ($K = 1024$)	0.0186	27.66	26.84	26.06	26.62	26.81
	SCALAR ($K = 2048$)	0.0373	27.78	27.10	25.67	27.97	**27.13**
LLaMA3.2-3B	FT	100.00	48.33	51.94	65.68	64.34	56.89
	LoRA	0.3770	48.02	51.82	64.97	64.15	56.57
	SCALAR ($K = 2048$)	0.0125	47.70	51.92	64.93	63.76	56.44
	SCALAR ($K = 4096$)	0.0250	47.80	52.18	65.42	64.37	**56.79**
Qwen2.5-7B	FT	100.00	71.77	68.06	84.24	77.21	74.46
	LoRA	0.2643	71.39	68.33	84.11	77.05	74.41
	SCALAR ($K = 2048$)	0.0052	71.55	68.12	84.69	77.41	**74.58**
	SCALAR ($K = 4096$)	0.0105	71.49	68.20	84.30	77.05	74.43

SCALAR is not restricted to a particular architecture or model size, and can support general-domain instruction tuning from 1B to 7B parameters with substantially fewer trainable parameters.

C.5 Scaling to Larger Models

A key property of SCALAR is that the number of trainable parameters introduced for each adapted matrix is decoupled from its input and output dimensions. For a weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, a rank-$r$ LoRA adapter introduces $r(m + n)$ trainable parameters. In contrast, SCALAR optimizes only K scalar coefficients for the corresponding fixed atoms, independent of m and n. Its total trainable parameter count may still depend on the number of adapted modules and Transformer layers, but it does not grow with the width of each linear transformation.

To examine whether this dimension-decoupled parameterization remains effective on larger backbones, we extend the commonsense reasoning experiments to Qwen2.5-14B and Qwen2.5-32B. All methods are trained on Commonsense170K and evaluated on the same eight benchmarks used in the main experiments. We use the same target modules and training protocol, and compare two configurations of LoRA and SCALAR.

Table 14: **Commonsense reasoning results on larger Qwen2.5 models.** Params denotes the percentage of trainable parameters. LoRA introduces $r(m + n)$ trainable parameters for an $m \times n$ weight matrix, whereas SCALAR assigns K trainable scalar coefficients to each adapted module, independent of m and n.

Model	Method	Params (%)	BoolQ	PIQA	SIQA	HS	WG	ARC-E	ARC-C	OBQA	Avg.
Qwen2.5-14B	LoRA ($r = 1$)	0.0291	78.6	93.1	84.9	97.4	92.3	97.7	93.4	95.8	91.7
	LoRA ($r = 8$)	0.2324	77.8	93.7	83.8	97.6	92.3	97.9	93.4	96.8	91.7
	SCALAR ($K = 4096$)	0.0093	77.8	94.0	84.9	97.4	91.9	97.9	93.8	96.6	91.8
	SCALAR ($K = 8192$)	0.0186	78.0	93.4	84.3	97.6	92.7	97.9	93.4	97.8	**91.9**
Qwen2.5-32B	LoRA ($r = 1$)	0.0255	80.0	94.7	84.0	97.7	92.7	98.7	95.6	97.4	92.6
	LoRA ($r = 8$)	0.2044	80.6	94.3	84.5	98.0	93.8	98.5	96.0	97.2	**92.9**
	SCALAR ($K = 4096$)	0.0056	79.2	94.6	85.5	97.8	93.4	98.7	95.8	97.0	92.8
	SCALAR ($K = 8192$)	0.0112	79.7	94.5	85.7	97.7	94.1	98.6	95.5	97.2	**92.9**

As shown in Table 14, SCALAR remains competitive as the backbone size increases. On Qwen2.5-14B, SCALAR with $K = 8192$ achieves the best average accuracy of 91.9 while training only 0.0186% of the model parameters. The $K = 4096$ configuration also slightly outperforms both LoRA settings, despite using approximately $3.1\times$ fewer trainable parameters than rank-one LoRA.

On Qwen2.5-32B, SCALAR with $K = 8192$ matches rank-eight LoRA at an average accuracy of 92.9, while reducing the trainable parameter percentage from 0.2044% to 0.0112%, corresponding to an approximately $18.3\times$ reduction. The $K = 4096$ configuration remains within 0.1 points of the best result while training only 0.0056% of the model parameters.

For a fixed value of K, the trainable parameter percentage of SCALAR decreases as the backbone grows from 14B to 32B. This behavior follows from the fact that the scalar budget of each adapted

module is independent of the matrix dimensions m and n. By contrast, LoRA requires $r(m+n)$ parameters per adapted matrix and therefore allocates more trainable parameters as the model width increases. These results show that SCALAR preserves competitive adaptation performance on larger models while becoming proportionally more parameter-efficient.

C.6 Ablation Study

We conduct ablation studies to examine two key design choices of SCALAR: the grouped update parameterization and the cross-layer sharing of the output-side bases. Unless otherwise specified, all variants are evaluated on the commonsense reasoning tasks using Qwen2.5-7B and follow the same training configuration as the main experiments.

C.6.1 Grouped Parameterization and Cross-Layer Sharing

Grouped parameterization. As introduced in Section 4, SCALAR parameterizes the weight update as

$$\Delta \boldsymbol{W} = s \sum_{r=1}^{R} \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h} \boldsymbol{a}_r^\top, \qquad K = RH, \tag{19}$$

where the H components in each group share the same input-side direction $\boldsymbol{a}_r$, while retaining distinct output-side directions $\boldsymbol{b}_{r,h}$. The update can therefore be equivalently expressed as

$$\Delta \boldsymbol{W} = s \sum_{r=1}^{R} \widetilde{\boldsymbol{b}}_r \boldsymbol{a}_r^\top, \qquad \widetilde{\boldsymbol{b}}_r = \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h}. \tag{20}$$

Although the formulation contains $K = RH$ scalar-weighted atoms, the H output atoms within each group can be merged into a single effective output vector $\widetilde{\boldsymbol{b}}_r$ before being applied to the input activations. Consequently, the resulting update has rank at most R, rather than K.

This grouped structure also reduces the cost of applying the update. For a batch of activations $\boldsymbol{X} \in \mathbb{R}^{n \times L}$, where L denotes the total number of tokens, the grouped update can be evaluated as

$$\Delta \boldsymbol{W} \boldsymbol{X} = s \widetilde{\boldsymbol{B}} \boldsymbol{A} \boldsymbol{X},$$

where $\boldsymbol{A} \in \mathbb{R}^{R \times n}$ and $\widetilde{\boldsymbol{B}} \in \mathbb{R}^{m \times R}$. Constructing $\widetilde{\boldsymbol{B}}$ requires weighted accumulation of the K fixed output atoms, with cost $\mathcal{O}(mK)$. The subsequent matrix multiplications cost $\mathcal{O}(LR(n+m))$, giving a total complexity of

$$\mathcal{O}\big(mK + TR(n+m)\big).$$

Since $K = RH$, the token-dependent matrix-multiplication cost scales with R rather than K. The merging cost is independent of T and can be amortized over the batch and sequence dimensions. The grouped formulation also reduces the intermediate activation size from $\mathcal{O}(LK)$ to $\mathcal{O}(LR)$.

To isolate the effect of grouping, we consider an ungrouped variant:

$$\Delta \boldsymbol{W} = s \sum_{k=1}^{K} \gamma_k \boldsymbol{b}_k \boldsymbol{a}_k^\top, \tag{21}$$

where each scalar coefficient is associated with an independent pair of input- and output-side directions. For $\boldsymbol{A} \in \mathbb{R}^{K \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m \times K}$, this update is evaluated as

$$\Delta \boldsymbol{W} \boldsymbol{X} = s \boldsymbol{B} \, \mathrm{diag}(\boldsymbol{\gamma}) \boldsymbol{A} \boldsymbol{X},$$

with dominant complexity

$$\mathcal{O}\big(LK(n+m)\big).$$

The ungrouped variant can produce an update with rank as high as K, but it requires all K input and output directions to be evaluated for every token. In contrast, grouping retains K candidate output atoms while reducing the token-dependent computation by approximately a factor of H.

As shown in Table 15, the grouped formulation increases training throughput from 0.58 to 0.75 iterations per second, corresponding to an improvement of approximately 29%, without increasing peak memory. This result is consistent with the complexity analysis: the grouped implementation

Table 15: **Ablation on the grouped update parameterization.** Results are reported on the commonsense reasoning tasks using Qwen2.5-7B. Grouping improves training throughput while also yielding slightly higher average accuracy.

Method	Iter./s	Mem. (GB)	Avg. Acc.
Grouped, equation 20 ($K = 1024, R = 16$)	0.75	49.1	88.5
Ungrouped, equation 21 ($K = 1024$)	0.58	49.1	88.3

first merges the H output atoms associated with each input direction through weighted vector accumulation, and then performs matrix multiplications over only R effective directions. By contrast, the ungrouped variant evaluates K independent input and output directions for every token.

The grouped formulation also achieves slightly higher average accuracy despite having a substantially lower maximum effective rank. In this configuration, the additional rank capacity of the ungrouped variant does not improve the reported average, while its token-dependent computation scales with all K directions. Grouping therefore provides a better measured accuracy–throughput trade-off for the selected parameter budget. The exact speedup remains dependent on the implementation, batch shape, and hardware.

Cross-layer sharing of $\boldsymbol{B}$. We next examine the cross-layer sharing strategy for the output-side bases. In the default implementation, layers corresponding to the same projection module share the same collection of fixed output directions $\{\boldsymbol{b}_{r,h}\}$. For example, all query-projection layers reuse one output basis, while different module types retain separate bases. Layer-specific adaptation is still preserved because each layer learns its own coefficients $\gamma_{r,h}$.

Table 16: **Ablation on cross-layer sharing of $\boldsymbol{B}$.** Sharing the output-side bases maintains nearly identical accuracy and throughput while substantially reducing memory consumption, especially as K increases.

Method	Iter./s	Mem. (GB)	Avg. Acc.
Shared $\boldsymbol{B}$ ($K = 1024$)	0.75	49.1	88.5
Unshared $\boldsymbol{B}$ ($K = 1024$)	0.75	51.6	88.5
Shared $\boldsymbol{B}$ ($K = 2048$)	0.75	49.3	88.6
Unshared $\boldsymbol{B}$ ($K = 2048$)	0.75	54.2	88.6
Shared $\boldsymbol{B}$ ($K = 4096$)	0.74	49.7	88.9
Unshared $\boldsymbol{B}$ ($K = 4096$)	0.74	59.4	88.8

Table 16 shows that removing cross-layer sharing provides no meaningful performance benefit. At all three values of K, the shared and unshared variants achieve nearly identical throughput and average accuracy. In contrast, the memory difference grows substantially with K. At $K = 1024$, sharing reduces peak memory from 51.6 GB to 49.1 GB, while at $K = 4096$, the reduction increases from 59.4 GB to 49.7 GB. Thus, the additional storage required by independent layer-wise bases grows rapidly as more directions are introduced, whereas the shared formulation keeps this overhead relatively small.

For the tested module dimensions and values of K, reusing the random output dictionary does not change the reported accuracy beyond 0.1 point. Each layer still forms a distinct update through its own calibrated input basis and scalar coefficients. We therefore use sharing as an implementation option that limits frozen-buffer memory; it is not required for the core `SCALAR` formulation. Grouping, by contrast, directly determines the efficient rank-R computation described in Section 4.

C.6.2 Effect of the Update Rank

We investigate the effect of the maximum update rank R while keeping the total number of trainable coefficients $K = RH$ fixed. For each $K \in \{1024, 2048, 4096\}$, we vary $R \in \{8, 16, 32, 64\}$ and

set $H = K/R$. Increasing R expands the activation-derived input subspace and raises the maximum update rank, whereas decreasing R allocates more output atoms to each input direction.

As shown in Table 17, $R = 16$ achieves the highest or tied-highest accuracy under all three parameter budgets. Nevertheless, the overall variation across $R = 8$ to 64 is small, with a maximum difference of only 0.2 points within each value of K. This suggests that SCALAR is relatively insensitive to the precise rank allocation within this range. In particular, neither substantially increasing the update rank nor assigning many more output atoms to each input direction produces a clear advantage. We therefore use $R = 16$ as the default setting, as it provides a stable balance between input-subspace coverage and output-side expressiveness.

C.6.3 Effect of the Calibration Size

SCALAR estimates the activation-derived input subspace from a calibration set collected before fine-tuning. We therefore study how the number of calibration examples affects both downstream performance and the one-time cost of subspace construction. Table 18 reports the average accuracy over the eight commonsense reasoning benchmarks together with the calibration time.

The downstream accuracy remains within a narrow range of 88.3–88.5 across all calibration sizes. Even 32 examples are sufficient to match the best reported average accuracy, while increasing the calibration size beyond 128 does not produce a consistent improvement. This result indicates that, for the model and task considered here, the dominant activation subspace can be estimated from a relatively small calibration set.

In contrast, the end-to-end calibration cost increases substantially with the number of examples, from 73.4 seconds at a calibration size of 16 to 1886.6 seconds at 1024. This timing includes activation collection through forward passes, distributed reduction of the activation Gram matrices, and the subsequent SVD computation and basis application. We use 128 examples by default as a moderate operating point: it attains the best observed average accuracy while avoiding the substantially higher cost of larger calibration sets. We do not interpret 128 as a uniquely optimal choice, since several smaller and larger calibration sizes achieve effectively identical accuracy. Calibration is performed once before fine-tuning and is excluded from the per-step training throughput reported in the main experiments.

C.6.4 Effect of the Output-Atom Initialization

In SCALAR, the input basis $\boldsymbol{A}$ is obtained from the leading right singular vectors of downstream activations and remains fixed during fine-tuning. The output atoms $\boldsymbol{B}$ are randomly initialized and are also kept frozen, while only the scalar coefficients are optimized. Under the default Gaussian initialization, the entries of $\boldsymbol{B}$ are sampled independently as

$$B_{ij} \sim \mathcal{N}(0, \sigma^2), \tag{22}$$

where σ denotes the element-wise standard deviation.

We examine whether performance is sensitive to the initialization distribution or scale of the fixed output atoms. The upper block of Table 19 compares Gaussian, Rademacher, and Kaiming initialization for $\boldsymbol{B}$. The lower block retains Gaussian initialization and varies σ, while keeping the activation-derived input basis and all other training hyperparameters unchanged.

Table 17: **Effect of the update rank R under fixed trainable-parameter budgets.** We report the average accuracy on the commonsense reasoning benchmarks using Qwen2.5-7B. Performance remains stable across a broad range of ranks, with $R = 16$ giving the strongest overall results.

	$K = 1024$	$K = 2048$	$K = 4096$
$R = 8$	88.4	**88.6**	88.7
$R = 16$	**88.5**	**88.6**	**88.9**
$R = 32$	**88.5**	**88.6**	88.8
$R = 64$	88.4	88.5	88.7

Table 18: **Effect of the calibration-set size on downstream performance and calibration cost on the 7B model.** Average accuracy is computed over the eight commonsense reasoning benchmarks. Calibration time denotes the one-time cost of collecting activation statistics and computing the input subspaces.

Calibration size	Average accuracy	Calibration time (s)
16	88.3	73.483
32	88.5	108.18
64	88.4	147.67
128 (default)	88.5	237.44
256	88.5	373.28
512	88.4	927.95
1024	88.5	1886.6

Table 19: **Effect of the initialization distribution and element-wise scale of the fixed output atoms $\boldsymbol{B}$.** The upper block compares different initialization rules, while the lower block varies the Gaussian standard deviation σ.

Output-atom initialization	$K = 1024$	$K = 2048$
Gaussian (default)	85.1	**85.2**
Rademacher	85.1	**85.2**
Kaiming	85.1	85.1
Gaussian, $\sigma = 0.05$	85.0	**85.2**
Gaussian, $\sigma = 0.10$	85.1	**85.2**
Gaussian, $\sigma = 0.20$	85.0	85.0

The results show that SCALAR is largely insensitive to the initialization of the fixed output atoms. Gaussian, Rademacher, and Kaiming initialization differ by at most 0.1 percentage points across the two values of K. Varying the Gaussian standard deviation from 0.05 to 0.2 changes accuracy by at most 0.2 points. We therefore use Gaussian initialization with $\sigma = 0.1$ by default. This setting provides consistently strong performance and permits a scalar learning-rate range comparable to that used for LoRA.

D Adapter Storage Analysis

Table 1 compares trainable parameters, but trainable parameter count is not always the same as adapter storage. This distinction is important for methods that use frozen bases, random dictionaries, or post-training merging. We therefore distinguish three quantities: trainable parameters, compact training checkpoint storage, and deployable adapter storage.

For a single adapted matrix $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, standard rank-$R$ LoRA stores

$$\Delta \boldsymbol{W} = \boldsymbol{B}\boldsymbol{A}, \qquad \boldsymbol{B} \in \mathbb{R}^{m\times R}, \qquad \boldsymbol{A} \in \mathbb{R}^{R\times n}.$$

Thus, both its trainable parameter count and deployable adapter storage are $R(m+n)$.

SCALAR uses a different training-time parameterization:

$$\Delta \boldsymbol{W} = s \sum_{r=1}^{R} \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h} \boldsymbol{a}_r^\top,$$

where s is the fixed atom-scaling factor defined in Appendix B.1.3. Only the RH scalar coefficients are trainable. The input directions $\boldsymbol{A} \in \mathbb{R}^{R\times n}$ are fixed task-specific buffers obtained from activation statistics, while the random output atoms can be shared across compatible layers or regenerated from random seeds. After training, the output atoms associated with the same input direction can be

merged:

$$\widetilde{\boldsymbol{b}}_r = s \sum_{h=1}^{H} \gamma_{r,h} \boldsymbol{b}_{r,h} = s \boldsymbol{B}_r \boldsymbol{\gamma}_r.$$

Therefore,

$$\Delta \boldsymbol{W} = \sum_{r=1}^{R} \widetilde{\boldsymbol{b}}_r \boldsymbol{a}_r^\top = \widetilde{\boldsymbol{B}} \boldsymbol{A}, \qquad \widetilde{\boldsymbol{B}} = \begin{bmatrix} \widetilde{\boldsymbol{b}}_1 & \cdots & \widetilde{\boldsymbol{b}}_R \end{bmatrix}.$$

The deployable SCALAR adapter is thus a standard rank-R adapter and requires $R(m+n)$ values, the same as rank-R LoRA. The key distinction is at training time: SCALAR optimizes only RH scalars. A self-contained compact checkpoint stores the calibrated $\boldsymbol{A}$, the learned $\boldsymbol{\Gamma}$, and the seeds needed to regenerate the random dictionary.

Table 20: Storage comparison for a single adapted matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$. Here, R is the adapter rank, H is the number of SCALAR output atoms per input direction, $K = RH$, and N_{scale} denotes the method-dependent number of stored scaling parameters for RandLoRA. Fixed random bases are assumed to be recoverable from seeds when applicable.

Method	LoRA	LoRA-XS	VeRA	NOLA	RandLoRA	SCALAR
Compact training checkpoint	$R(m+n)$	$R^2 + R(m+n)$	$m + R +$ seeds	$k_A + k_B +$ seeds	$N_{\text{scale}} +$ seeds	$Rn + RH +$ seeds

For direct loading, the LoRA-XS checkpoint includes the trainable $R \times R$ matrix and the frozen SVD-derived factors. Although these frozen factors are not trainable, they are needed for direct adapter loading unless one recomputes the truncated SVD from the pretrained weight. For NOLA, random bases may be regenerated from seeds, so its compact checkpoint need only retain the learned combination coefficients and the corresponding seeds. SCALAR has a similar seed-based variant: instead of saving the random output dictionary, we save only the activation-derived $\boldsymbol{A}$, the trained scalar matrix $\boldsymbol{\Gamma}$, and the seeds required to regenerate $\{\boldsymbol{B}_r\}_{r=1}^{R}$. When a deployable adapter is required, $\boldsymbol{\Gamma}$ and the regenerated random dictionary are merged into $\widetilde{\boldsymbol{B}}$, resulting in the standard rank-R storage $R(m+n)$.

We next instantiate these quantities on Qwen2.5-3B and Qwen2.5-7B. We adapt all linear projection modules: q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, and down_proj. Storage is reported under BF16 precision, where each value occupies two bytes.

The results show that SCALAR does not reduce the storage of the merged rank-R deployable adapter below LoRA; after merging, both require the same $R(m+n)$ values. Instead, SCALAR reduces the number of trainable parameters and enables a compact seed-based checkpoint. On Qwen2.5-7B, rank-8 and rank-16 LoRA require 40.37 MB and 80.74 MB, respectively, whereas the corresponding compact SCALAR checkpoints require only 18.92 MB and 37.04 MB. The trainable scalar coefficients occupy only 0.80 MB in both cases because $K = 2048$ is fixed. When materialized for deployment, SCALAR becomes a standard rank-R adapter and therefore matches the corresponding LoRA storage.

E Benchmark Details

We evaluate the adaptation methods on three representative domains: commonsense reasoning, mathematical reasoning, and code generation. For each domain, we fine-tune the pretrained models on a corresponding instruction dataset and evaluate the resulting checkpoints on held-out benchmarks. The descriptions below summarize the data used in our experiments; evaluation details and hyperparameters are given in Appendix B.2.

E.1 Fine-Tuning Datasets

- **Commonsense170K.** Commonsense170K (Hu et al., 2023) is a collection of approximately 170K instruction-formatted examples constructed from the training sets of eight commonsense reasoning benchmarks: BoolQ, PIQA, SIQA, HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge, and OpenBookQA. The original tasks are converted into a

Table 21: **Practical adapter storage on Qwen2.5-3B and Qwen2.5-7B under BF16 precision.** We report LoRA and SCALAR with $R \in \{8, 16\}$, while fixing $K = RH = 2048$ for SCALAR. Storage is reported in MB with $1\text{MB} = 10^6$ bytes.

Storage type	Qwen2.5-3B	Qwen2.5-7B
Rank-8 LoRA deployable adapter	29.93 MB	40.37 MB
SCALAR compact checkpoint ($R = 8$)	14.45 MB	18.92 MB
SCALAR deployable adapter ($R = 8$)	29.93 MB	40.37 MB
Rank-16 LoRA deployable adapter	59.86 MB	80.74 MB
SCALAR compact checkpoint ($R = 16$)	27.87 MB	37.04 MB
SCALAR deployable adapter ($R = 16$)	59.86 MB	80.74 MB

unified instruction–response format and combined for multi-task fine-tuning. Following prior PEFT studies, we train on the combined dataset and evaluate separately on the test set of each constituent benchmark.

- **MetaMathQA.** MetaMathQA (Yu et al., 2023) is a mathematical instruction-tuning dataset containing approximately 395K examples. It is constructed by rewriting existing mathematical questions from multiple perspectives, including forward, backward, and answer-conditioned transformations, while preserving the underlying mathematical knowledge. The dataset contains step-by-step solutions and is designed to improve multi-step mathematical reasoning. We fine-tune the models on MetaMathQA and evaluate them on GSM8K and MATH.
- **Code-Feedback.** Code-Feedback (Zheng et al., 2024) contains approximately 66K code-related interactions involving code generation, execution feedback, and iterative refinement. Its examples cover both initial solutions and subsequent corrections based on execution results, providing supervision for generating functionally correct programs. We use Code-Feedback for code instruction tuning and evaluate the resulting models on HumanEval and MBPP.

E.2 Evaluation Benchmarks

Commonsense Reasoning. We evaluate commonsense reasoning using the eight benchmarks included in Commonsense170K:

- **BoolQ** (Clark et al., 2019) consists of naturally occurring yes-or-no questions paired with passages and evaluates reading comprehension and implicit reasoning.
- **PIQA** (Bisk et al., 2020) evaluates physical commonsense reasoning by asking models to select the more plausible solution to an everyday physical problem.
- **SIQA** (Sap et al., 2019) contains multiple-choice questions about social interactions and evaluates reasoning about people's intentions, reactions, and social consequences.
- **HellaSwag** (Zellers et al., 2019) evaluates commonsense natural-language inference by requiring models to select the most plausible continuation of a given context.
- **WinoGrande** (Sakaguchi et al., 2021) is a large-scale Winograd-style benchmark that tests commonsense reasoning and coreference resolution through binary fill-in-the-blank questions.
- **ARC-Easy and ARC-Challenge** (Clark et al., 2018) are the easy and challenging subsets of the AI2 Reasoning Challenge, respectively. Both contain grade-school-level multiple-choice science questions, with ARC-Challenge including questions that are more difficult for retrieval- and word-matching-based systems.
- **OpenBookQA** (Mihaylov et al., 2018) contains multiple-choice science questions that require combining elementary scientific facts with broader commonsense knowledge.

We report accuracy on each benchmark and use their unweighted mean as the overall commonsense reasoning score.

Mathematical Reasoning. We evaluate mathematical reasoning on GSM8K and MATH:

- **GSM8K** (Cobbe et al., 2021) contains 8.5K linguistically diverse grade-school mathematics word problems. Solving these problems generally requires several arithmetic reasoning steps before producing a final numerical answer.
- **MATH** (Hendrycks et al., 2021b) consists of competition-level mathematics problems spanning algebra, counting and probability, geometry, intermediate algebra, number theory, pre-algebra, and pre-calculus. Compared with GSM8K, it requires more advanced mathematical knowledge and longer reasoning chains.

We extract the final answer from each model response and report exact-match accuracy following the standard evaluation protocol.

Code Generation. We evaluate code generation on HumanEval and MBPP:

- **HumanEval** (Chen et al., 2021) contains 164 hand-written Python programming problems. Each problem provides a function signature, a natural-language specification, and hidden unit tests used to assess functional correctness.
- **MBPP** (Austin et al., 2021) contains short Python programming problems described in natural language, together with reference solutions and test cases. The tasks primarily assess the ability to synthesize functionally correct programs for basic programming problems.

For both benchmarks, generated programs are executed against the provided unit tests, and we report pass@1 as the primary metric.

General-Domain Knowledge Evaluation. We evaluate broad knowledge retention after general instruction fine-tuning using Massive Multitask Language Understanding (MMLU) (Hendrycks et al., 2021a). MMLU consists of multiple-choice questions from 57 academic and professional subjects, covering STEM, humanities, social sciences, and other domains. The benchmark evaluates the breadth of a model's pretrained knowledge and its ability to apply that knowledge across heterogeneous subject areas.

We report accuracy for the four aggregated subject categories and the overall accuracy across all MMLU tasks. This evaluation is used as a complementary measure of general-domain knowledge preservation after fine-tuning on Alpaca-GPT4 (Peng et al., 2023), rather than as a comprehensive assessment of open-ended instruction-following ability.