CovRA: LoRA Rank Allocation via Conditional Marginal Gain

Anonymous authors
Paper under double-blind review

Abstract

Low-rank adaptation (LoRA) is a widely used method for parameter-efficient fine-tuning, and its effectiveness is shaped by rank allocation and weight initialization. Existing pre-fine-tuning rank-allocation methods typically determine module ranks from importance scores of modules or candidate rank-one updates, but these scores are computed once and not updated as candidates are selected. When different candidates within a module exhibit similar first-order loss-sensitivity patterns across calibration samples, independent scores may count this overlap repeatedly, overstating the additional value of candidates similar to those already selected. We propose Coverage-based Rank Allocation (CovRA). Using a small calibration set on a frozen model, CovRA constructs candidate rank-one updates for each module and takes their per-sample first-order loss sensitivities as responses. It then selects candidates sequentially within each module: at each step, it reevaluates all unselected candidates, treats the part of their responses explained by selected candidates as already covered, and defines the residual response energy as their additional value, yielding a conditional marginal gain. These gains are standardized across modules and used to assign integer ranks under a fixed parameter budget. CovRA additionally reuses the input directions retained during allocation to initialize LoRA. Across two base models and three tasks, CovRA outperforms standard LoRA at the same parameter budget in every setting. On Llama-3.1-8B-Base, it improves GSM8K accuracy over standard LoRA by 5.16 percentage points. In a controlled comparison in the same setting with random initialization and all else fixed, replacing independent valuation with conditional marginal valuation alone improves accuracy by 1.31 points, indicating that updating valuation with the selected set can yield more effective cross-module rank configurations.

1 Introduction

As large language models continue to grow (Grattafiori et al., 2024; OpenAI et al., 2024; Gemini Team et al., 2024), the storage and computational costs of full fine-tuning (Full FT) increase accordingly. Parameter-efficient fine-tuning (PEFT) has therefore become a mainstream approach: adapter modules (Houlsby et al., 2019) and prefix tuning (Li & Liang, 2021) train only a small number of additional parameters, and their design choices can be compared from a unified perspective (He et al., 2022). LoRA adapts large models to downstream tasks using few trainable parameters (Hu et al., 2022), consistent with the low intrinsic dimensionality of fine-tuning (Aghajanyan et al., 2021), and can be combined with quantization to further reduce memory use (Dettmers et al., 2023). Standard LoRA assigns the same rank to all target modules, although modules at different depths and functional positions need not have the same adaptation demand. Uniform rank gives some modules more capacity than they need while limiting the task-related updates that others can express. Allocating a limited parameter budget across modules is thus central to improving LoRA's parameter efficiency.

One family of methods determines module ranks once from a small calibration set before fine-tuning, after which training follows the standard LoRA pipeline. Examples include GoRA, EVA, and GeLoRA (He et al., 2025; Paischer et al., 2025; Ed-dib et al., 2025). These methods assign fixed independent values to modules or candidate rank-one updates without updating them with the selected set. Orthogonality in parameter or activation space does not ensure non-overlapping per-sample responses: two orthogonal spectral directions can still produce highly similar responses

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.

on the same samples, causing independent valuation of later rank units to count already-covered responses again.

We therefore define a candidate's value as the additional response it provides given the selected set. CovRA constructs candidate rank-one updates from calibration gradients, forms per-sample responses, and defines conditional marginal gain as projection-residual energy relative to the span of selected responses. After each selection, the remaining candidates are reevaluated so that already-covered responses no longer contribute to additional value. The resulting gain sequence is standardized into candidate-level scores and aggregated into module demand and soft quotas, which jointly guide integer rank allocation under the parameter budget.

Experiments support this design through performance comparisons, mechanism analysis, and response-overlap diagnostics. Under a fixed LoRA parameter budget, CovRA improves over standard LoRA by 5.16 percentage points on Llama-3.1-8B-Base/GSM8K. In controlled comparisons with random initialization, conditional marginal valuation improves over the shared rank-position decay control by 0.99 percentage points. In the default candidate pool, 125 of 128 modules exhibit greater response redundancy than an encoding-matched permutation null. The resulting rank configurations are stable across seeds and differ across tasks, while selected-input-direction initialization further reuses existing calibration information.

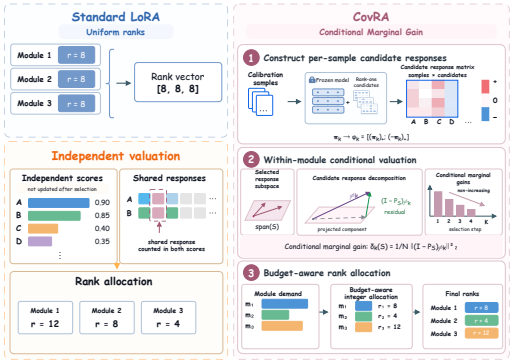


Figure 1: Schematic rank allocation. CovRA values additional responses conditional on the selected set and assigns integer ranks under a fixed budget. Values are illustrative.

Our contributions are threefold.

1. We reexamine pre-fine-tuning LoRA rank allocation through the lens of per-sample response redundancy. Independent scores can repeatedly count responses already covered by earlier candidates, so we redefine a candidate's value as the additional response it provides relative to the selected set.
2. We propose CovRA, a pre-fine-tuning rank-allocation method based on conditional marginal gain. CovRA constructs rank-one candidates from calibration gradients, defines conditional marginal gain as projection-residual energy relative to the span of selected responses, and converts the resulting sequence into an integer rank configuration under a fixed parameter budget; retained input directions are further reused for initialization.
3. We validate CovRA through performance comparisons, controlled comparisons, and mechanism diagnostics. CovRA consistently outperforms standard LoRA at the same parameter budget. On Llama-3.1-8B-Base/GSM8K, it improves by 5.16 percentage points. Conditional marginal valuation improves over independent valuation by 1.31 percentage points and retains a 0.99-point improvement after controlling for shared rank-position decay; in the default candidate pool, 125/128 modules exhibit greater response redundancy than the permutation null.

2 Related Work

2.1 Rank Allocation for LoRA

Adaptive rank-allocation methods can be divided roughly by when allocation occurs. Dynamic methods adjust effective rank or the cross-module budget during optimization, including AdaLoRA, SoRA, DyLoRA, IncreLoRA, HALoRA, and IGU-LoRA (Zhang et al., 2023b; Ding et al., 2023; Valipour et al., 2023; Zhang et al., 2023a; 2026; Cui et al., 2026). They maintain additional scoring, gating, pruning, or scheduling state during training; salience analysis also studies LoRA intrinsic rank from training-time signals (Ke et al., 2024). SubLoRA uses submodular greedy selection of singular values from a trained LoRA and likewise values candidates conditional on the selected set

(Gao & Tan, 2025). CovRA instead measures overlap through per-sample first-order loss responses before fine-tuning.

A second family determines the rank configuration once before fine-tuning using model statistics or a small calibration set. GoRA, EVA, and GeLoRA use calibration-gradient importance, explained variance of activation directions, and intrinsic dimension of hidden representations, respectively (He et al., 2025; Paischer et al., 2025; Ed-dib et al., 2025). Other methods likewise use salience, entropy, or task-related statistics before fine-tuning (Li et al., 2026; Lin et al., 2026; Kim et al., 2024). We focus on this family. Their scores typically do not update with the current selected set once computed and hence do not explicitly measure a candidate's additional response relative to selected candidates. EVA uses an orthogonal decomposition into activation principal components to characterize explained variance, so its directions have a clear non-redundancy property in activation space. However, non-redundancy in activation space does not directly imply non-redundancy in per-sample loss-response space, which is the focus of our conditional marginal valuation.

2.2 Direction Selection and Initialization

A separate line of work improves adapter initialization, update subspaces, or optimization once rank is given. PiSSA, OLoRA, MiLoRA, and LoRA-GA use spectral directions, orthogonal bases, minor singular components, or initial-gradient information (Meng et al., 2024; Büyükakyüz, 2024; Wang et al., 2025a; 2024); HRP and DoRA change the starting point or parameterization through high-rank preheating or weight decomposition, respectively (Chen et al., 2025; Liu et al., 2024). Other methods improve optimization of low-rank parameters through factor learning rates, gradient correction, preconditioning, or scaling (Hayou et al., 2024; Wang et al., 2025b; Zhang & Pilanci, 2024; Kalajdzievski, 2023). These methods typically do not directly determine the rank budget across target modules and are complementary to the rank-allocation problem studied here. CovRA further reuses input directions retained during rank allocation for initialization, making use of existing calibration information. Section 4.3 examines the effect of this reuse.

3 Method

CovRA measures a candidate's additional response relative to the selected set and aggregates each module's sequence of conditional marginal gains into the module demand used for cross-module allocation. Section 3.1 defines the problem; Section 3.2 explains how calibration gradients yield candidates and their per-sample responses; Section 3.3 defines conditional marginal gains that update with the selected set; Section 3.4 standardizes the sequence and converts it into integer ranks under the parameter budget; and Section 3.5 reuses retained input directions for initialization. All steps occur before fine-tuning, after which training follows standard LoRA.

3.1 Problem Setup

Let $\mathcal{M}$ be the set of target modules. For module m, let W_m be its frozen pretrained weight. LoRA introduces low-rank factors $B_m \in \mathbb{R}^{d_{m,\text{out}} \times r_m}$ and $A_m \in \mathbb{R}^{r_m \times d_{m,\text{in}}}$ with integer rank r_m. CovRA uses effective scaling α/r_m (Appendix E). Standard LoRA assigns the same rank to all target modules, whereas CovRA jointly determines $\{r_m\}$ subject to per-module rank bounds and a fixed parameter budget. Because different modules introduce different numbers of parameters per added rank, the unit-rank cost and budget constraint are

$$c_m = d_{m,\text{in}} + d_{m,\text{out}}, \qquad \sum_{m \in \mathcal{M}} c_m r_m = B^\star, \qquad r_{\min} \le r_m \le r_{\max}. \tag{1}$$

Here $B^\star$ is the reference parameter budget, defined as the actual trainable parameter count of standard LoRA at uniform reference rank r_{ref}. The expansion $B_m A_m = \sum_{j=1}^{r_m} b_{m,j} a_{m,j}^\top$ expresses the update as r_m rank-one updates, where $b_{m,j}$ and $a_{m,j}^\top$ are column j of B_m and row j of A_m, respectively. We call the capacity added by increasing a module's rank by one, which accommodates an additional rank-one update, a *rank unit*. Its value is the additional per-sample response that this update can provide on the calibration samples relative to the module's existing selections. To estimate this value, CovRA constructs a finite set of candidate rank-one updates from calibration gradients on the frozen model and computes their responses on each calibration sample; both are defined next.

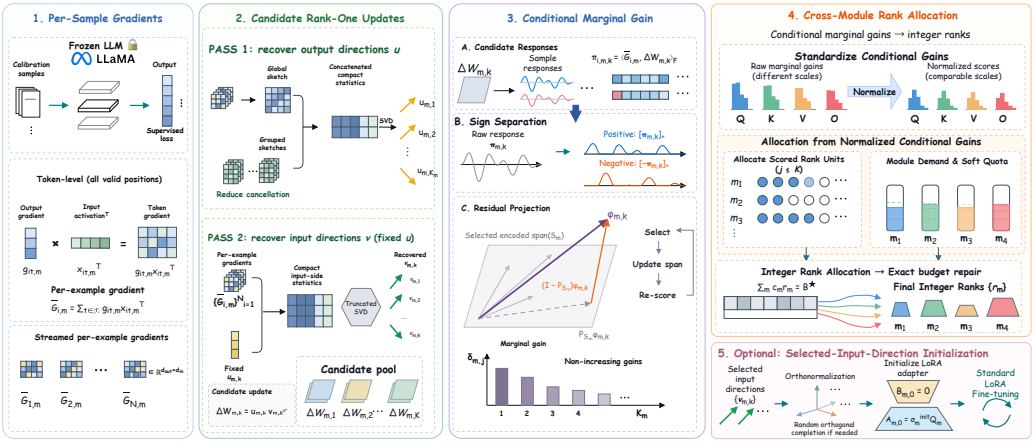


Figure 2: Overview of CovRA. Per-sample gradients define candidate rank-one updates and responses; conditional marginal gains guide budget-constrained integer rank allocation. Retained input directions optionally initialize LoRA.

3.2 Candidate Rank-One Updates and Per-Sample Responses

This section focuses on one target module and omits the module index m. For sample i, let $\mathcal{A}_i$ be the positions at which the supervised loss is computed, and let $\mathcal{T}_i$ contain all valid positions at which the module participates in the forward pass, with $\mathcal{A}_i \subseteq \mathcal{T}_i$. Define the sample loss as $L_i = |\mathcal{A}_i|^{-1} \sum_{a \in \mathcal{A}_i} \ell_{ia}$. At position t, let x_{it} and $h_{it} = W x_{it}$ be the module input and output, and let $g_{it} = \partial L_i / \partial h_{it}$. The per-sample calibration gradient is

$$\bar{G}_i = \nabla_W L_i = \sum_{t \in \mathcal{T}_i} g_{it} x_{it}^\top. \tag{2}$$

The supervised loss is computed only at positions in $\mathcal{A}_i$, but the weight gradient accumulates contributions from all valid module positions, including positions that can influence supervised predictions through later computation without directly carrying a loss. Because L_i is already normalized by $|\mathcal{A}_i|$, Equation 2 is not averaged again by sequence length or supervised-token count. Appendix E details per-sample gradient extraction and normalization. Candidates are constructed in two streaming passes over the calibration set. A seeded permutation divides the samples into C groups $\mathcal{I}_1, \ldots, \mathcal{I}_C$, and a random projection $\Omega \in \mathbb{R}^{d_{\text{in}} \times s}$ compresses the input dimension. The first pass forms global and grouped sketches:

$$\begin{aligned} Y^{(0)} &= \sum_i \bar{G}_i \Omega, \qquad Y^{(c)} = \sum_{i \in \mathcal{I}_c} \bar{G}_i \Omega, \\ \widehat{Y} &= [\lambda Y^{(0)} \mid Y^{(1)} \mid \cdots \mid Y^{(C)}]. \end{aligned} \tag{3}$$

The top K left singular vectors of $\widehat{Y}$ give the output directions u_k. Holding these directions fixed, the second pass forms

$$\begin{aligned} z_k^{(0)} &= \sum_i \bar{G}_i^\top u_k, \qquad z_k^{(c)} = \sum_{i \in \mathcal{I}_c} \bar{G}_i^\top u_k, \\ \widehat{Z}_k &= [\lambda z_k^{(0)} \mid z_k^{(1)} \mid \cdots \mid z_k^{(C)}]. \end{aligned} \tag{4}$$

The leading left singular vector of $\widehat{Z}_k$ gives the unit input direction v_k and candidate $\Delta W_k = u_k v_k^\top$ (Appendix A). Global terms capture directions consistent across samples; grouped terms retain directions strong within subsets but weakened in the global sum. We use $C = 4$ and $\lambda = 1$.

For candidate k and sample i, define the *per-sample response*

$$\pi_{ik} = \langle \bar{G}_i, \Delta W_k \rangle_F = u_k^\top \bar{G}_i v_k = \frac{d}{d\eta} L_i(W + \eta \Delta W_k)\Big|_{\eta=0}. \tag{5}$$

Thus π_{ik} is the directional derivative of sample i's supervised loss along candidate k. Let $\boldsymbol{\pi}_k = (\pi_{1k}, \ldots, \pi_{Nk})^\top \in \mathbb{R}^N$ be its response vector over the calibration set. The candidate count K

limits only explicitly constructed and valued candidates, not the final allocated rank. Each candidate is oriented so that $\sum_i \pi_{ik} \geq 0$ (Appendix A). Under this convention, a perturbation along $-\Delta W_k$ makes the first-order change in mean calibration loss nonpositive; $\pi_{ik} > 0$ indicates a first-order loss decrease for sample i, and $\pi_{ik} < 0$ a first-order increase. Distinct candidates can still induce similar responses on the same samples, so Section 3.3 values them conditionally on the responses already selected.

3.3 Conditional Marginal Gain

Once the candidate pool is fixed, CovRA asks how much new per-sample response a candidate provides given the selections already made within the same module. Orthogonality in parameter space does not guarantee non-overlapping per-sample responses: truncated-SVD output directions and their rank-one updates may be mutually orthogonal while producing similar responses on the same calibration samples. CovRA therefore performs conditional valuation in the response space defined in Section 3.2. CovRA removes the component of each candidate response explained by previously selected responses; the residual energy is its conditional marginal gain. Under the sign convention above, CovRA treats only responses in the same direction on the same sample as overlap. It separates the positive and negative parts of each response vector, where $[\cdot]_+$ is applied elementwise:

$$\phi_k = \begin{bmatrix} [\boldsymbol{\pi}_k]_+ \\ [-\boldsymbol{\pi}_k]_+ \end{bmatrix} \in \mathbb{R}^{2N}. \tag{6}$$

This encoding preserves the energy of each candidate while placing opposing responses on the same sample in different coordinates so that they do not count as overlap. Appendix B gives the relevant properties and an example. Let S be the current selected set and P_S the orthogonal projector onto the span of $\{\phi_{k'} : k' \in S\}$. The conditional marginal gain of candidate k relative to S is the average energy in its encoded response not explained by that subspace:

$$\delta_k(S) = \frac{1}{N} \left\| (I - P_S)\phi_k \right\|_2^2 . \tag{7}$$

Starting from $S = \varnothing$, CovRA repeatedly selects $k_j = \arg\max_{k \notin S} \delta_k(S)$, records $\delta_j = \delta_{k_j}(S)$, and updates $S \leftarrow S \cup \{k_j\}$ until all K candidates have been ordered. Crucially, every step recomputes the gains of all remaining candidates using the updated selected set; using only the empty-set response energy would reduce the procedure to fixed independent valuation. Because the selected response subspace expands, the residual energy of any remaining candidate cannot increase, so $\delta_1 \geq \cdots \geq \delta_K$. Conditional marginal gain measures uncovered first-order response energy on the calibration set, not a guaranteed reduction in downstream training loss. Appendix B gives the related mathematical connections. The testable hypothesis behind this design is that candidates within a module exhibit per-sample response overlap beyond what can be explained by individual response distributions. Section 4.4 tests this hypothesis with a permutation baseline. Each module thus yields a conditional selection order and gain sequence $\delta_{1:K}$, which Section 3.4 standardizes and aggregates into module demand.

3.4 Cross-Module Standardization and Rank Allocation

Section 3.3 yields a conditional marginal gain sequence $\delta_{m,1:K}$ for every module. Turning these sequences into integer ranks satisfying Equation 1 requires addressing two issues: gain scales are not directly comparable across modules or projection types, and modules have different unit-rank parameter costs c_m. Before cross-module comparison, CovRA applies one robust transformation to all gains. It first log-compresses their heavy-tailed scale, then standardizes them using the median and median absolute deviation (MAD) within the same target-module projection type $\tau(m)$, and finally applies softplus to obtain nonnegative order-preserving candidate scores:

$$w_{m,j} = \mathrm{softplus}\left(\frac{\log(\delta_{m,j} + \varepsilon_\delta) - \mu_{\tau(m)}}{\mathrm{MAD}_{\tau(m)} + \varepsilon_{\mathrm{MAD}}} \right). \tag{8}$$

Here μ_τ and MAD_τ are computed separately over valid candidate log-gains of each type τ. Llama-3.1-8B-Base uses $q/k/v/o$; Llama-2-7B-Base additionally uses $up/gate/down$, each as its own standardization group. Appendix C.1 gives candidate-validity and numerical-stability rules.

Summing the available candidate scores gives module demand D_m. Square-root compression produces the budget share s_m, which is converted through the unit-rank cost into the module soft quota $\bar{r}_m$:

$$D_m = \sum_j w_{m,j}, \qquad s_m = \frac{\sqrt{D_m + \varepsilon_D}}{\sum_{n \in \mathcal{M}} \sqrt{D_n + \varepsilon_D}}, \qquad \bar{r}_m = r_{\min} + \frac{s_m B_{\text{rem}}}{c_m}, \tag{9}$$

where the sum uses only available candidates and $B_{\text{rem}} = B^\star - \sum_{m \in \mathcal{M}} c_m r_{\min}$ is the budget remaining after base ranks. Square-root compression reduces the heavy tail of module demand and avoids concentrating capacity in a few modules. We use D_m only as a proxy for relative capacity demand, and $\bar{r}_m$ is a soft reference rather than a hard cap. Every module starts at $r_{\min}$. An increment backed by an unused valid scored candidate k uses dynamic priority $P_{m,k}$; a soft-quota-guided increment uses relative shortfall g_m:

$$\begin{aligned} \psi_m &= \frac{\max(0, r_m - r_{\min})}{\max(\bar{r}_m - r_{\min}, \epsilon)}, \qquad P_{m,k} = \frac{w_{m,k}}{c_m^\beta (1 + \psi_m^2)}, \\ g_m &= \frac{\max(0, \bar{r}_m - r_m)}{\max(\bar{r}_m - r_{\min}, \epsilon)}, \qquad \beta = 0.5. \end{aligned} \tag{10}$$

Here r_m is the current integer rank. Candidate priority declines with quota progress, while quota-guided increments favor larger relative shortfalls. Conditional gains from Equation 7 become candidate scores in Equation 8, then module demand and soft quotas in Equation 9, before these rules yield integer ranks. Gains are measured only for the first K candidates per module; no new gains are extrapolated beyond K. Every increment obeys the rank bounds and the budget cap, and strict budget repair makes the final cost exactly $B^\star$. Appendix C details both increment types and budget repair; valid input directions retained after repair are used for the initialization in Section 3.5.

3.5 Selected-Input-Direction Initialization (SelInit)

After each module's integer rank has been determined, CovRA can apply selected-input-direction initialization (SelInit). SelInit does not participate in valuation or allocation; it only reuses valid input directions retained after budget repair. For module m, these directions are orthogonalized in descending score order and completed with random directions orthogonal to the selected subspace, forming a row-orthogonal $Q_m \in \mathbb{R}^{r_m \times d_{m,\text{in}}}$. With the norm-matching scale σ_m^{init}, initialize

$$A_{m,0} = \sigma_m^{\text{init}} Q_m, \qquad B_{m,0} = 0. \tag{11}$$

The initial low-rank update is therefore zero. Let G_m be the full weight gradient. Under a one-step gradient-descent approximation without adaptive preconditioning or weight decay, only B_m receives a gradient at the first step, giving

$$\Delta W_{m,1} \propto -\gamma G_m A_{m,0}^\top A_{m,0} = -\gamma (\sigma_m^{\text{init}})^2 G_m Q_m^\top Q_m, \tag{12}$$

where γ is the step size in this approximation. Thus the initial update retains the full-gradient component in the row space of Q_m; selected directions make that subspace calibration-dependent. Subsequent training is not restricted to the candidate span. Appendix D gives the construction and derivation, Section 4.3 gives the controlled comparison, and module ranks remain fixed during fine-tuning.

4 Experiments and Analysis

This section addresses three questions: whether CovRA improves performance under a fixed parameter budget, whether conditional marginal valuation provides a useful signal beyond independent valuation and shared rank-position decay, and whether the default candidate pool exhibits the response overlap that CovRA is designed to address. We then analyze selected-input-direction initialization, rank-configuration stability, and one-time preprocessing cost.

4.1 Experimental Setting and Controlled Protocol

The main experiments fine-tune Llama-3.1-8B-Base (Grattafiori et al., 2024) on WizardLM-52K (Xu et al., 2024), MetaMathQA-100K (Yu et al., 2024), and Code-Feedback-100K (Zheng et al.,

2024), and evaluate with MT-Bench (Zheng et al., 2023), GSM8K (Cobbe et al., 2021), and HumanEval pass@1 (Chen et al., 2021), respectively. Llama-2-7B-Base (Touvron et al., 2023) serves as a second base model. We adapt $q/k/v/o$ on Llama-3.1-8B-Base and additionally $up/gate/down$ on Llama-2-7B-Base. HumanEval pass@1 uses one generated completion per problem with temperature 0.8 and $\mathrm{top_p} = 0.95$. Results are means $\pm$ sample standard deviations over three runs; CovRA and its controlled variants use seeds 42–44. Baselines include LoRA, AdaLoRA, GoRA, DoRA, and LoRA-GA, all reproduced under a unified pipeline; GoRA is the primary pre-fine-tuning rank-allocation baseline. LoRA and default CovRA exactly match the reference budget $B^\star$, while other methods retain their native parameter counts; Full FT is an unconstrained reference. Default CovRA uses $r_{\mathrm{ref}} = 8$, $B^\star = 6{,}815{,}744$, $K = 16$, $N = 256$, and strict budget repair; we also examine $r_{\mathrm{ref}} = 32$ and 128. Calibration samples come from the corresponding training sets and never include evaluation data. Appendix E gives the complete training, implementation, and rank-bound settings, and Appendix H reports sensitivity to K and N.

4.2 Main Results

Tables 1 and 2 summarize the main results on two base models. Because the task metrics have different scales, we do not compute an across-task average.

Table 1: Results on Llama-3.1-8B-Base.

Method	MT-Bench ↑	GSM8K ↑	HumanEval ↑
Full FT	5.89 ± 0.18	73.54 ± 0.42	51.63 ± 1.27
LoRA	6.13 ± 0.03	67.78 ± 1.19	42.89 ± 0.35
AdaLoRA†	6.18 ± 0.13	70.48 ± 0.82	42.07 ± 3.23
GoRA†	$\mathbf{6.34 \pm 0.04}$	$\underline{72.71 \pm 0.68}$	$\underline{48.78 \pm 1.83}$
DoRA†	6.24 ± 0.10	69.35 ± 0.94	44.11 ± 1.53
LoRA-GA	6.03 ± 0.06	71.22 ± 0.84	43.70 ± 0.70
CovRA (Ours)	$\underline{6.33 \pm 0.04}$	$\mathbf{72.93 \pm 0.50}$	$\mathbf{48.98 \pm 1.96}$
CovRA$_{r_{\mathrm{ref}}=32}$	6.28 ± 0.06	75.66 ± 0.38	50.20 ± 1.76
CovRA$_{r_{\mathrm{ref}}=128}$	6.30 ± 0.21	75.82 ± 0.38	51.02 ± 1.76

Table 2: Results on Llama-2-7B-Base.

Method	MT-Bench ↑	GSM8K ↑	HumanEval ↑
Full FT	5.30 ± 0.11	59.36 ± 0.85	35.37 ± 2.11
LoRA	5.59 ± 0.09	42.08 ± 0.08	15.04 ± 0.35
AdaLoRA	5.61 ± 0.07	50.54 ± 1.28	17.89 ± 0.70
GoRA	5.66 ± 0.10	$\underline{54.03 \pm 0.34}$	$\underline{24.39 \pm 1.22}$
DoRA	$\mathbf{5.93 \pm 0.04}$	53.37 ± 0.75	19.51 ± 0.61
LoRA-GA	$\underline{5.90 \pm 0.12}$	53.58 ± 0.42	20.33 ± 0.93
CovRA (Ours)	5.83 ± 0.10	$\mathbf{54.21 \pm 0.65}$	$\mathbf{24.80 \pm 1.27}$
CovRA$_{r_{\mathrm{ref}}=32}$	5.79 ± 0.08	56.91 ± 0.55	26.02 ± 1.41
CovRA$_{r_{\mathrm{ref}}=128}$	5.81 ± 0.10	57.09 ± 0.60	26.63 ± 1.53

Mean $\pm$ sample SD over three runs; GSM8K and HumanEval are percentages. **Bold**/underlined values are the best/second-best mean scores among default-setting PEFT methods, excluding Full FT and higher-budget rows; they do not indicate statistical significance. Shading identifies CovRA. In Table 1, † marks baselines whose budgets differ from $B^\star$; see Appendix Table F1.

Results. On Llama-3.1-8B-Base, CovRA reaches 72.93 ± 0.50 GSM8K accuracy and 48.98 ± 1.96 HumanEval pass@1 with the same 6.82M trainable-parameter budget as standard LoRA, improving over LoRA by 5.16 and 6.10 percentage points, respectively. The 5.16-point gain is calculated from unrounded results; the table means are displayed to two decimal places. Its MT-Bench score is 6.33 ± 0.04, close to GoRA's 6.34 ± 0.04. GoRA uses 7.00M trainable parameters (102.70% of the reference budget), whereas CovRA exactly matches the reference budget. Overall, CovRA obtains larger gains on mathematical reasoning and code generation, while differences among methods are smaller on MT-Bench. On Llama-2-7B-Base, CovRA reaches 54.21 ± 0.65 on GSM8K and 24.80 ± 1.27 on HumanEval, with slightly higher mean scores than GoRA on both metrics in Table 2. Its MT-Bench score is 5.83 ± 0.10, below DoRA and LoRA-GA. These results show that CovRA's GSM8K and HumanEval gains are not limited to one base model, while also indicating that the magnitude of its advantage varies across task metrics. Higher-reference-budget results appear in Appendix Table F4. At $r_{\mathrm{ref}} = 32$ and 128, CovRA reaches 75.66 ± 0.38 and 75.82 ± 0.38 GSM8K accuracy, respectively. Increasing the reference budget improves performance beyond the default setting, but the increase from $r_{\mathrm{ref}} = 32$ to 128 is only 0.15 percentage points, indicating diminishing returns from further budget increases.

4.3 Controlled Comparison of Valuation and Initialization

Valuation. To isolate the effect of valuation, we fix candidate construction, standardization, the soft-quota rule, rank allocation, and budget repair and use random initialization throughout; only candidate valuation changes.

CovRA-I uses each candidate's own response energy for independent valuation, CovRA-SD additionally applies shared rank-position decay across modules, and CovRA dynamically computes conditional marginal gains from selected responses. Appendix E gives complete definitions. Because CovRA-I and CovRA select the same first candidate, their difference mainly arises from conditional reevaluation and reordering of later candidates. Table 3 shows that CovRA-RI improves

over CovRA-I-RI and CovRA-SD-RI by 1.31 and 0.99 percentage points on GSM8K, respectively, and by 1.22 and 0.81 points on HumanEval. On GSM8K, CovRA-RI outperforms both CovRA-I-RI and CovRA-SD-RI in all three paired runs (Appendix Table F3). Because these configurations share the remaining pipeline and all use random initialization, updating valuation with the selected set provides useful rank-allocation information on both tasks beyond independent scores and shared rank-position decay.

Initialization. Table 3 also compares selected-input-direction initialization (SelInit). At uniform rank, SelInit raises GSM8K from 67.78 to 70.81 and HumanEval from 42.89 to 45.53; under CovRA allocation, the corresponding values rise from 72.48 to 72.93 and from 46.95 to 48.98. Holding initialization fixed, CovRA allocation improves over uniform rank by $+4.70$ and $+4.07$ points under random initialization on GSM8K and HumanEval, respectively, and by $+2.12$ and $+3.46$ points under SelInit. Rank allocation and direction initialization both contribute, but their benefits are not simply additive.

Table 3: Controlled results on Llama-3.1-8B-Base at the default budget.

Configuration	GSM8K (%) ↑	HumanEval (%) ↑
Uniform	67.78 ± 1.19	42.89 ± 0.35
Uniform-SelInit	70.81 ± 0.38	45.53 ± 0.93
CovRA-I-RI	71.17 ± 0.44	45.73 ± 1.06
CovRA-SD-RI	71.49 ± 0.13	46.14 ± 1.53
CovRA-RI	72.48 ± 0.40	46.95 ± 1.83
CovRA	72.93 ± 0.50	48.98 ± 1.96

Mean $\pm$ sample SD over seeds 42–44.

4.4 Response-Overlap Diagnostic

As discussed in Section 3.3, candidates that are orthogonal in parameter space can still have overlapping per-sample responses, causing independent valuation to count already-covered responses repeatedly. To test for this overlap, we sort candidates in descending order of their own response energy and use cumulative response redundancy $\rho(k)$ to measure the fraction of energy in the first k candidates explained by the span of preceding candidate responses. We also construct an encoding-matched permutation null by independently permuting each candidate's calibration sample indices before applying the same sign-separated encoding. This preserves each candidate's response distribution, energy, and sign proportion while disrupting sample-wise correspondence across candidates. Appendix G.1 gives the definitions and statistical aggregation. Table 4 uses the default Llama-3.1-8B-Base/GSM8K pool with $N = 256$ and $K = 16$. Values are medians $[Q_1, Q_3]$ over 128 modules; brackets indicate interquartile ranges across modules, not variation across seeds or confidence intervals. Paired differences are computed within each module before aggregation. At all three diagnostic depths, actual candidates have higher cumulative response redundancy than the permutation null. For the full $K = 16$ pool, the paired difference is positive for 125/128 modules, with consistent trends at $\rho(4)$ and $\rho(8)$. These results show widespread per-sample response overlap in the default candidate pool and provide empirical support for conditional marginal valuation.

Table 4: Response overlap in the default pool: module medians $[Q_1, Q_3]$.

Diagnostic	Actual candidates	Matched permutation null	Paired difference (actual − null)
$\rho(4)$	0.205 $[0.151, 0.259]$	0.108 $[0.075, 0.143]$	0.093 $[0.064, 0.120]$
$\rho(8)$	0.241 $[0.179, 0.292]$	0.132 $[0.091, 0.169]$	0.104 $[0.074, 0.130]$
$\rho(16)$	0.254 $[0.188, 0.311]$	0.143 $[0.101, 0.181]$	0.108 $[0.078, 0.134]$

4.5 Analysis of Rank Allocations

To examine whether CovRA's final ranks vary by model and task, we compare three model–task settings. Figure 3 shows the mean final ranks of $q/k/v/o$ projection modules across Transformer layers, with each cell averaged over seeds 42–44. The final rank configuration is not a fixed layer-depth template: it varies with the model–task setting.

The different GSM8K and HumanEval configurations on the same Llama-3.1-8B-Base indicate that each task's calibration data affect capacity allocation across modules; cross-model differences also reflect architecture and per-rank parameter costs. To distinguish cross-task differences from allocation-stage randomness, Table 5 compares same-task configurations across seeds and cross-task configurations on the same architecture. Spearman correlation is computed over all $q/k/v/o$ modules and may include systematic differences between projection types. We use it only to compare overall agreement, without interpreting it as a within-projection-type relationship; Appendix H gives the aggregation details. Across seeds, configurations are highly correlated and their mean absolute rank difference is below one; across tasks on the same architecture, correlation is lower and mean absolute rank difference is larger. Both metrics show that cross-task differences exceed same-task cross-seed variation: allocation is relatively stable across repeated runs while changing with the calibration data. In the default setting, allocation and initialization together take 1011 ± 1 s of preprocessing, about 10.57% of training time (9565 ± 74 s); Appendix H reports stage timings and the run environment.

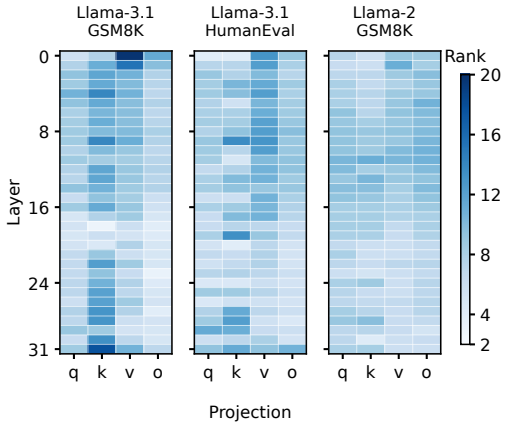


Figure 3: Mean attention-projection ranks over seeds 42–44. Full labels and values appear in Figure H2.

Table 5: Rank stability on Llama-3.1-8B-Base.

Comparison	Spearman ρ	Mean absolute rank difference
Across seeds (same task)	0.911 ± 0.008	0.818 ± 0.027
GSM8K vs. HumanEval (same model)	0.55	1.84

Across seeds: mean $\pm$ sample SD over three seed pairs. Across tasks: comparison of seed-mean rank vectors.

5 Discussion and Conclusion

CovRA performs one-time valuation and integer allocation from a small calibration set before fine-tuning, after which ranks remain fixed. Training-time dynamic allocation can use the full optimization trajectory but must maintain allocation state throughout optimization; the two approaches therefore make different trade-offs between available information and implementation cost. This work has three main limitations. First, conditional marginal valuation operates within individual modules, over a finite candidate pool, and with a sign-separated response representation; it does not model response dependence across modules. Joint cross-module valuation would substantially expand the combinatorial selection space and is beyond the scope of this work. Second, using module demand D_m as a proxy for relative capacity demand is a modeling choice whose relationship to actual capacity requirements is not directly tested here. Third, the experiments cover only two Llama base models and a limited set of target-module configurations, while retaining standard LoRA α/r_m scaling. Examining calibration-distribution shifts, other model families, additional target-module configurations, and the separate effects of capacity and rank-dependent scaling would help establish the method's range of applicability.

We presented CovRA, which allocates LoRA ranks before fine-tuning by scoring each candidate rank-one update by the per-sample response it adds beyond the candidates already selected in its module. The resulting conditional marginal gains are aggregated into module demand and converted into integer ranks under a fixed parameter budget. Across two base models and three tasks, CovRA outperforms standard LoRA at the same budget, by 5.16 percentage points on Llama-3.1-8B-Base/GSM8K. In the same setting, with other components fixed and random initialization, conditional valuation improves over independent valuation by 1.31 points and retains its advantage over shared rank-position decay; the default candidate pool also exhibits widespread per-sample response overlap. These results indicate that measuring a candidate's additional value relative to the selected set is an effective criterion for LoRA rank allocation before fine-tuning.

AI USE STATEMENT

LLM assistants were used for language polishing and translation, searching related work, and assisting with research ideation. All cited works and AI-assisted research outputs were independently verified by the authors, who take full responsibility for this paper.

REPRODUCIBILITY STATEMENT

Section 3 and Appendices A–H describe the method, experimental protocols, and complete results. Code will be released on GitHub upon publication.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.568.

Kerim Büyükakyüz. OLoRA: Orthonormal low-rank adaptation of large language models. arXiv:2406.01775, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. arXiv:2107.03374, 2021.

Yuzhu Chen, Yingjie Wang, Shi Fu, Li Shen, Yongcheng Jing, Xinmei Tian, and Dacheng Tao. HRP: High-rank preheating for superior LoRA initialization. arXiv:2502.07739, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training verifiers to solve math word problems. arXiv:2110.14168, 2021.

Xuan Cui, Huiyue Li, Run Zeng, Yunfei Zhao, Jinrui Qian, Wei Duan, Bo Liu, and Zhanpeng Zhou. IGU-LoRA: Adaptive rank allocation via integrated gradients and uncertainty-aware scoring. arXiv:2603.13792, 2026.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, 2023.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Abdessalam Ed-dib, Zhanibek Datbayev, and Amine M. Aboussalah. GeLoRA: Geometric adaptive ranks for efficient LoRA fine-tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 25174–25196. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.1372.

Yihang Gao and Vincent Y. F. Tan. Automatic rank determination for low-rank adaptation via submodular function maximization. arXiv:2507.01841, 2025.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv:2403.05530, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. arXiv:2407.21783, 2024.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Haonan He, Peng Ye, Yuchen Ren, Yuan Yuan, Luyang Zhou, Shucun Ju, and Lei Chen. GoRA: Gradient-driven adaptive low rank adaptation. In *Advances in Neural Information Processing Systems*, 2025.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

Neil Houlsby, Andrei Giurgiu, Stanisław Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 2790–2799. PMLR, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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

Wenjun Ke, Jiahao Wang, Peng Wang, Jiajun Liu, Dong Nie, Guozheng Li, and Yining Li. Unveiling LoRA intrinsic ranks via salience analysis. In *Advances in Neural Information Processing Systems*, 2024.

Minsoo Kim, Sihwa Lee, Wonyong Sung, and Jungwook Choi. RA-LoRA: Rank-adaptive parameter-efficient fine-tuning for accurate 2-bit quantized large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Minglei Li, Peng Ye, Jingqi Ye, Haonan He, and Tao Chen. E2LoRA: Efficient and effective low-rank adaptation with entropy-guided adaptive sharing. In *International Conference on Learning Representations*, 2026.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021.

Weicheng Lin, Yi Zhang, Jiawei Dang, and Liang-Jie Zhang. TLoRA: Task-aware low rank adaptation of large language models. arXiv:2604.18124, 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 *Proceedings of the 41st 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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3846.

OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, et al. GPT-4o system card. arXiv:2410.21276, 2024.

Fabian Paischer, Lukas Hauzenberger, Thomas Schmied, Benedikt Alkin, Marc Peter Deisenroth, and Sepp Hochreiter. Parameter efficient fine-tuning via explained variance adaptation. In *Advances in Neural Information Processing Systems*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, et al. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288, 2023.

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

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*, 2025a.

Shaowen Wang, Linxi Yu, and Jian Li. LoRA-GA: Low-rank adaptation with gradient approximation. In *Advances in Neural Information Processing Systems*, 2024.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-Pro: Are low-rank adapters properly optimized? In *International Conference on Learning Representations*, 2025b.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, 2024.

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. In *International Conference on Learning Representations*, 2024.

Fangzhao Zhang and Mert Pilanci. Riemannian preconditioned LoRA for fine-tuning foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. IncreLoRA: Incremental parameter allocation method for parameter-efficient fine-tuning. arXiv:2308.12043, 2023a.

Letian Zhang, Guanghao Meng, Xudong Ren, and Jinpeng Wang. HALoRA: Low-rank adaptation with hierarchical budget allocation for efficient vision-language alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 28283–28291, 2026. doi: 10.1609/aaai.v40i33.40056.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023b.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, et al. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

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. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.762.

A Algorithm and Candidate-Construction Details

This appendix gives the complete pre-fine-tuning CovRA procedure and the candidate-construction details.

Candidate-construction computation. Use the per-sample gradients $\bar{G}_{i,m} \in \mathbb{R}^{d_{m,\text{out}} \times d_{m,\text{in}}}$ from Equation 2. Let $\Omega_m \in \mathbb{R}^{d_{m,\text{in}} \times s}$ be the input-side random projection, $\mathcal{I}_c$ the cth sample group, and C the number of groups. Each run uses its sketch seed to permute sample indices with Python's `random.Random`, then assigns samples cyclically by their permuted positions modulo C. All target modules share these groups within the run. For module m at zero-based position j in the ordered target-module list, Ω_m is sampled from $\mathcal{N}(0, I/s)$ using a CPU PyTorch generator seeded with the sketch seed plus j. The sketch accumulators and SVD inputs use float32. The first pass accumulates

$$Y_m^{(0)} = \sum_i \bar{G}_{i,m}\Omega_m, \qquad Y_m^{(c)} = \sum_{i \in \mathcal{I}_c} \bar{G}_{i,m}\Omega_m,$$
$$\widehat{Y}_m = [\lambda Y_m^{(0)} \mid Y_m^{(1)} \mid \cdots \mid Y_m^{(C)}] \in \mathbb{R}^{d_{m,\text{out}} \times (C+1)s}. \tag{13}$$

Compute the exact thin SVD with `torch.linalg.svd` and `full_matrices=False`, $\widehat{Y}_m = U_m \Sigma_m V_m^\top$, and take $u_{m,k} = U_m[:,k] \in \mathbb{R}^{d_{m,\text{out}}}$ for $k = 1, \ldots, K$. Holding each output direction fixed, the second pass accumulates

$$z_{m,k}^{(0)} = \sum_i \bar{G}_{i,m}^\top u_{m,k}, \qquad z_{m,k}^{(c)} = \sum_{i \in \mathcal{I}_c} \bar{G}_{i,m}^\top u_{m,k} \in \mathbb{R}^{d_{m,\text{in}}}. \tag{14}$$

Algorithm 1 CovRA

Require: Frozen model, calibration set $\mathcal{D}_{\text{cal}}$, target modules $\mathcal{M}$, reference budget $B^\star$, $r_{\min}$, K, $r_{\max}$
Ensure: Integer ranks $\{r_m\}$ and initialized LoRA adapters

```
for each module m ∈ M do
    Accumulate global and grouped sketches with a shared random projection
    Compute a truncated SVD and retain K candidate output directions
    Recover paired input directions in a second streaming pass
    Construct candidate rank-one updates and apply the sign-alignment convention
    Compute per-sample responses
    S_m ← ∅
    for j = 1, ..., K do
        Evaluate each unselected candidate's residual response energy relative to span(S_m)
        Select the candidate with the largest conditional marginal gain
        Update S_m and record its conditional marginal gain
    end for
end for
Apply within-type log compression, robust standardization, and softplus to gains
Allocate base rank to every module and associate available high-scoring input directions
Aggregate standardized scores into module demand and construct soft quotas
Allocate feasible rank units with a deterministic greedy rule: use candidate-level dynamic priority for increments directly supported by scored candidates with numerically valid input directions, and soft-quota guidance for remaining capacity
Enforce rank bounds and the budget cap B* for every increment; extrapolate no new conditional marginal gains beyond K
if strict budget repair is enabled then
    Repair discrete-cost reachability to exact B* without reading scores or soft quotas
    Fail the run if exact B* is unreachable within rank bounds
end if
Obtain final ranks within [r_min, r_max]; final r_m may exceed K
Collect retained, numerically valid input directions after budget repair
Orthogonalize by residual projection in descending standardized-score order
Fill missing directions with Kaiming random directions orthogonal to the selected subspace
Apply QR orthogonalization and norm matching to initialize A; set B = 0
return {r_m} and initialized LoRA adapters
```

Concatenate these vectors as

$$\widehat{Z}_{m,k} = [\lambda z_{m,k}^{(0)} \mid z_{m,k}^{(1)} \mid \cdots \mid z_{m,k}^{(C)}] \in \mathbb{R}^{d_{m,\text{in}} \times (C+1)}. \tag{15}$$

The first left singular vector of $\widehat{Z}_{m,k}$ from the same exact thin-SVD routine, normalized to unit length, is $v_{m,k}$, giving

$$\|v_{m,k}\|_2 = 1, \qquad \Delta W_{m,k} = u_{m,k} v_{m,k}^\top \in \mathbb{R}^{d_{m,\text{out}} \times d_{m,\text{in}}}. \tag{16}$$

Before response computation, if $\langle v_{m,k}, z_{m,k}^{(0)} \rangle < 0$, set $v_{m,k} \leftarrow -v_{m,k}$. For a zero inner product, retain the SVD sign from the fixed backend and seed. All candidate alignment values in the default Seed 42–44 runs were strictly positive after alignment. This matches the convention in Section 3.2.

Global accumulation emphasizes gradient directions that are consistent across samples. Random grouped accumulation retains variation that is strong within subsets of samples and may be weakened by global aggregation. We fix $C = 4$ and $\lambda = 1$ and do not tune either value. Sample grouping affects only candidate-pool construction; it does not alter the subsequent per-sample response computation or conditional marginal valuation.

B Conditional Marginal Gain Details

This appendix gives three basic properties of the sign-separated responses and conditional marginal gains in Section 3.3. As a supplementary connection, if the empirical Fisher matrix is defined as the

mean of outer products of per-sample gradients, a candidate's own response energy is the empirical Fisher quadratic form along that candidate update direction. This interpretation explains the statistical meaning of independent response energy but does not change the conditional marginal valuation used in the main text. For a raw response vector π, define

$$\phi(\pi) = [(\pi^+)^\top, (\pi^-)^\top]^\top, \qquad \pi^+ = \max(\pi, 0), \quad \pi^- = \max(-\pi, 0). \tag{17}$$

The properties below concern candidate responses after the sign-alignment convention in Section 3.2.

Energy preservation. The positive and negative parts have disjoint support, so

$$\|\phi(\pi)\|_2^2 = \|\pi^+\|_2^2 + \|\pi^-\|_2^2 = \|\pi\|_2^2. \tag{18}$$

Sign separation preserves individual candidate response energy, but changes inter-candidate inner products and conditional projections.

Nonnegative response inner products. For any two candidate responses π_a, π_b,

$$\phi(\pi_a)^\top \phi(\pi_b) = (\pi_a^+)^\top \pi_b^+ + (\pi_a^-)^\top \pi_b^- \geq 0. \tag{19}$$

The Gram matrix of sign-separated responses is therefore elementwise nonnegative. This representation preserves same-direction response overlap on the same sample and places opposing responses in different coordinates.

Non-increasing conditional marginal gains. As candidates are selected, their response subspace only expands, so every remaining candidate's projection-residual energy cannot increase. Selecting the largest current residual energy gives

$$\delta_{m,1} \geq \delta_{m,2} \geq \cdots \geq \delta_{m,K}. \tag{20}$$

This monotonicity follows from conditional projection, not an imposed shared rank-position decay.

Illustrative example. For

$$\pi_a = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad \pi_b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \tag{21}$$

the raw response inner product is zero. Raw signed responses characterize candidate relationships using signed inner products and linear spans. After sign separation, the inner product is one and counts only same-direction overlap on the same sample. Taking absolute values instead makes both vectors $(1, 1)^\top$. Thus raw signed responses, sign separation, and absolute values define different notions of overlap; CovRA uses sign separation as its coverage measure.

Connection to orthogonalization residuals and Schur complements. Within a module's sign-separated response matrix, let ϕ_j denote a candidate column, S the selected set, P_S the orthogonal projector onto the span of selected response columns, and G the Gram matrix of response columns. Let $(\cdot)^+$ denote the Moore–Penrose pseudoinverse. Conditional marginal gain can be written as

$$\delta_j(S) = \frac{1}{N} \phi_j^\top (I - P_S) \phi_j = \frac{1}{N} \left[G_{jj} - G_{jS} G_{SS}^+ G_{Sj} \right]. \tag{22}$$

This is the Schur complement of the candidate's own Gram term conditional on selected responses, divided by N. If the selected response columns are linearly independent and G_{SS} is invertible, then

$$\det G_{S \cup \{j\}} = \det G_{SS} \left(G_{jj} - G_{jS} G_{SS}^{-1} G_{Sj} \right). \tag{23}$$

The sequential residual calculation has the same form as residual norms in column-pivoted Gram–Schmidt / QR, and is a conditional marginal gain given selected responses from the Gram-matrix perspective. This connection explains valuation over the finite candidate pool. Cross-module rank configuration is determined by the subsequent standardization, module-demand aggregation, and integer budget allocation.

For a nonsingular Gram submatrix, the product of sequential Schur complements equals its determinant. The accumulated logarithms of raw conditional marginal gains thus admit a log-determinant interpretation. This interpretation applies only to gains before standardization; median/MAD calibration, softplus, demand aggregation, and square-root compression serve cross-module allocation and do not retain the same exact determinant interpretation.

C Discrete Rank Allocation and Budget Repair Details

This appendix details integer budget allocation in Algorithm 1. Given rank bounds $r_{\min} \leq r_m \leq r_{\max}$, unit rank costs c_m, scored candidates, and reference budget $B^\star$, allocation starts from base ranks. Candidate-level dynamic priority and module soft quotas guide additional capacity, followed by budget repair. The candidate count K limits only direct valuation within each module, not final rank. Every increment must keep actual cost at or below $B^\star$.

C.1 Score- and Quota-Guided Capacity Allocation

Conditional marginal gains map to nonnegative standardized scores by Equation 8. A candidate participates in integer allocation as a scored candidate only if its response, gain, and score are finite and its input direction is numerically valid. Module demand, budget shares, and soft quotas follow Section 3.4.

Every module first receives base rank $r_{\min}$, whose parameter cost counts toward the budget from the start. If candidates meet the above conditions, base rank units are preferentially associated with the input directions having the highest standardized scores, and these candidates are marked as used. A direction is numerically valid only if all its elements are finite and $\|v_{m,k}\|_2 > \epsilon_v$; directions containing NaN or Inf, or whose norm does not exceed the threshold, are numerically invalid.

For remaining scored candidates with numerically valid input directions, dynamic priority incorporates the standardized score, unit rank cost, and module progress relative to its soft quota:

$$\psi_m = \frac{\max(0, r_m - r_{\min})}{\max(\bar{r}_m - r_{\min}, \epsilon)}, \qquad P_{m,k} = \frac{w_{m,k}}{c_m^\beta (1 + \psi_m^2)}. \tag{24}$$

Here $w_{m,k}$ is the standardized score associated with candidate k; the default cost-sensitive exponent is $\beta = 0.5$ (Appendix E). For modules with unused scored candidates, additional rank units are preferentially selected by this dynamic priority. Scores use only the first K measured positions; no new conditional marginal gains are extrapolated for capacity beyond K. Module shortfall relative to the soft quota guides the other increments. Both types obey the rank upper bound and $B^\star$ budget constraint; soft quotas are capacity references, not hard upper bounds.

For soft-quota-guided capacity, define relative quota shortfall as

$$g_m = \frac{\max(0, \bar{r}_m - r_m)}{\max(\bar{r}_m - r_{\min}, \epsilon)}. \tag{25}$$

At every step, among modules satisfying the rank upper bound and whose next rank unit fits within the budget cap, choose the module with the largest relative shortfall. Resolve ties by a fixed module order and recompute shortfalls after each increment. If all feasible modules have reached or exceeded their soft quotas and the budget remains incomplete, prefer the feasible module with the smallest relative excess.

If the selected module still has an unused numerically valid input direction, preferentially associate it with the additional rank unit; otherwise add capacity only. These increments do not extrapolate new conditional marginal gains, and their module assignment is guided, rather than constrained, by soft quotas.

C.2 Budget Repair

Because modules may have different unit rank costs, sequential allocation need not reach the reference budget exactly. When enabled, budget repair finds the largest reachable actual parameter cost not exceeding $B^\star$ within the rank bounds. Among configurations with the same reachable cost, it first minimizes module-wise absolute rank changes from the pre-repair configuration, then resolves ties by a fixed module order. Repair may simultaneously decrease ranks of high-cost modules and increase ranks of several lower-cost modules.

Budget repair reads neither standardized scores nor module soft quotas. It addresses only discrete-cost budget reachability while preserving the pre-repair allocation structure as closely as possible. All reported experiments for CovRA and its controlled variants enable strict budget repair and require the final actual cost to equal $B^\star$ exactly. A run fails if exact equality is unreachable within the rank bounds.

C.3 Complexity

Let P be the total number of scored candidates, R the number of rank units added under soft-quota guidance, and $|\mathcal{M}|$ the number of target modules. Candidate standardization and sorting cost at most $O(P \log P)$. If dynamic priorities of all remaining candidates are recomputed every round, candidate-level greedy selection has worst-case complexity $O(P^2)$. Soft-quota-guided capacity allocation requires approximately $O(R|\mathcal{M}|)$ module comparisons. Budget repair by bounded cost-state search adds pseudopolynomial overhead.

D Initialization Derivation

D.1 Initialization Construction and the First Update

Selected-input-direction initialization does not change CovRA's rank-allocation objective; it reuses the input directions that are finally retained and numerically valid after budget repair. For module m, these directions are processed in descending standardized-score order and orthonormalized by modified Gram–Schmidt, projecting out the span of already-retained directions before normalizing. If fewer numerically valid directions are available than the final rank r_m, they are completed with Kaiming-uniform random directions, which are likewise projected out of the retained span.

Let the resulting r_m directions form the row-orthogonal matrix Q_m. After a shared norm-matching step, the LoRA input factor is

$$A_{m,0} = \sigma_m^{\text{init}} Q_m, \qquad B_{m,0} = 0, \tag{26}$$

where the initialization norm scale σ_m^{init} is independent of the budget share s_m (Section 3.4) and of the effective LoRA scaling coefficient. Because $B_{m,0} = 0$, the low-rank update is zero at initialization, so the model's initial function is unchanged.

To explain the effect of the initialized input subspace on the earliest update direction, consider a one-step gradient-descent approximation that ignores adaptive preconditioning and weight decay. Let the learning rate be γ and the full weight gradient be G_m. Under this approximation, the gradient with respect to A_m is zero, while B_m updates first along the input subspace fixed by $A_{m,0}$, so the first low-rank weight change is approximately

$$\Delta W_{m,1} \propto -\gamma G_m A_{m,0}^\top A_{m,0}. \tag{27}$$

Since $A_{m,0}^\top A_{m,0} = (\sigma_m^{\text{init}})^2 Q_m^\top Q_m$ and $Q_m^\top Q_m$ is the orthogonal projector onto the initialized input subspace, the first low-rank update preferentially retains the component of the full gradient that lies within this input subspace. Random completion directions follow the same orthogonalization and norm-matching rules. This property holds for any given orthogonal input subspace. Selected directions make that subspace depend on calibration information; this does not imply that the initialization objective is equivalent to the conditional response-coverage objective in Section 3.3.

E Complete Experimental Protocol

E.1 General Experimental Setting

Instruction following, mathematical reasoning, and code generation on Llama-3.1-8B-Base use WizardLM-52K, MetaMathQA-100K, and Code-Feedback-100K, respectively; Llama-2-7B-Base serves as a second base model. On Llama-3.1-8B-Base, the target modules are the attention `q_proj`, `k_proj`, `v_proj`, and `o_proj` projections. On Llama-2-7B-Base, the target modules additionally include `up_proj`, `gate_proj`, and `down_proj`. CovRA and our controlled variants use seeds 42–44. Baselines report the mean $\pm$ sample standard deviation over three runs; same-seed pairing is used only for our internal controlled comparisons. Methods requiring calibration draw samples from the corresponding training set; evaluation data do not participate in calibration.

E.2 Controlled Variants

Full CovRA uses conditional marginal allocation and selected-input-direction initialization. CovRA-RI, CovRA-I-RI, CovRA-SD-RI, and CovRA-M-RI use random initialization. Valuation compar-

isons share the candidate pool, rank bounds, score transformation, reference budget, and integer-allocation pipeline except for the value construction under study. Uniform-SelInit is used for the initialization comparison.

Table E1: Definitions of CovRA and five controlled configurations.

Method	Valuation scheme	Revalues after selection	Initialization	Comparison purpose
CovRA	Conditional marginal gain	Yes	Selected input directions	Full method
Uniform-SelInit	Uniform rank $r = 8$; no conditional marginal rank allocation	—	Selected input directions	Effect of selected-input-direction initialization at uniform rank
CovRA-RI	Conditional marginal gain	Yes	Random	Conditional marginal allocation with random initialization
CovRA-I-RI	Fixed independent candidate energy	No	Random	Fixed independent versus conditional marginal valuation
CovRA-SD-RI	Fixed independent energy $\times$ shared rank-position decay	No	Random	Shared rank-position decay versus module-specific conditional structure
CovRA-M-RI	Module-level scalar valuation with shared rank-position decay	No	Random	Module-level versus candidate-level valuation

Valuation definitions. Let $e_{m,k} = N^{-1}\|\phi_{m,k}\|_2^2$ be the candidate's own response energy, ordered within module as $e_{m,(1)} \geq \cdots \geq e_{m,(K)}$. Define the module total $a_m = \sum_{j=1}^{K} e_{m,(j)}$ and the shared rank-position decay by

$$t_j^{\mathrm{raw}} = \mathrm{median}_m \frac{e_{m,(j)}}{a_m + \epsilon}, \qquad t_j = \frac{t_j^{\mathrm{raw}}}{\sum_{\ell=1}^{K} t_\ell^{\mathrm{raw}}}, \qquad d_j = \frac{t_j}{t_1}. \tag{28}$$

The four valuation schemes are

$$\eta_{m,j}^{\mathrm{M}} = a_m t_j, \quad \eta_{m,j}^{\mathrm{I}} = e_{m,(j)}, \quad \eta_{m,j}^{\mathrm{SD}} = e_{m,(j)} d_j, \quad \eta_{m,j}^{\mathrm{RI}} = \delta_{m,j}. \tag{29}$$

M-RI retains overall module importance with shared rank-position decay; I-RI uses fixed own response energy; SD-RI applies shared rank-position decay to module-specific own response energy; RI updates conditional marginal gains with the selected set. All four then use the same type-wise log–median/MAD–softplus transformation, parameter costs, module soft quotas, candidate-level dynamic priority, soft-quota-guided capacity allocation, and budget repair, with random initialization.

Initialization comparison. Random initialization uses $A_0 \sim \mathrm{KaimingUniform}$ and $B_0 = 0$ and does not perform Gram–Schmidt on selected input directions, projection onto the selected input subspace, or direction norm matching. Selected-input-direction initialization follows Appendix D. In the default setting, $K = \min(2r_{\mathrm{ref}}, 64) = 16$. Uniform-SelInit keeps rank r_{ref} for every module without cross-module allocation, uses at most r_{ref} numerically valid directions constructed for that module, processes them in descending standardized-score order, and fills any remaining rows by the same random-orthogonal rule. It therefore isolates selected-input-direction initialization under uniform rank.

E.3 CovRA Calibration and Allocation Configuration

We focus on default reference rank $r_{\mathrm{ref}} = 8$, with $(r_{\min}, K, r_{\max}) = (2, 16, 32)$ and a standard-LoRA reference budget of 6,815,744 trainable parameters. Score transformation follows Section 3.4. The dynamic-priority cost exponent is $\beta = 0.5$; module soft quotas and soft-quota-guided capacity allocation are specified in Appendix C. Table E2 gives the configuration.

Table E2: CovRA calibration and allocation configuration.

(a) Default Llama-3.1-8B-Base/GSM8K settings

Item	Setting
Calibration data	256 examples; batch size 4; fixed example order; run seeds 42, 43, and 44
Candidates and sketches	$K = 16$ candidates directly scored per module; sketch dimension 32; float32 computation; exact thin SVD
Grouping and weights	$C = 4$ groups; equal global and grouped sketch weights ($\lambda = 1$); the run seed controls sketch grouping
Supervision and per-sample gradients	Loss is computed only at prediction positions corresponding to answer labels; per-sample weight gradients accumulate contributions from all valid sequence positions of the target module. Calibration batch size is 4. Gradients are extracted separately for each sample within the batch and rescaled to the single-sample token-averaged loss in Equation 2.
Rank bounds and repair	$r_{\min} = 2$, $r_{\max} = 32$; budget repair enabled
Dynamic priority and completion	Cost exponent $\beta = 0.5$; soft-quota-guided capacity allocation
Budget and target modules	6,815,744 trainable parameters; `q_proj`, `k_proj`, `v_proj`, `o_proj`
Initialization	Reuse finally retained numerically valid input directions after budget repair; initialize $B = 0$; randomly orthogonally fill missing directions

(b) Reference ranks, rank bounds, candidates, and budgets

r_{ref}	$r_{\min}$	K	$r_{\max}$	Sketch dim.	N	β	Budget
8	2	16	32	32	256	0.5	6,815,744
32	8	64	128	128	256	0.5	27,262,976
128	32	64	512	128	256	0.5	109,051,904

All three configurations use four example groups and global weight 1. The sketch dimension s is listed for each reference rank; each configuration retains the leading K left singular vectors from the exact thin SVD of the float32 hybrid sketch, with no separate randomized-SVD oversampling parameter. Candidate depth follows $K = \min(2r_{\text{ref}}, 64)$ and rank bounds follow $r_{\min} = r_{\text{ref}}/4$ and $r_{\max} = 4r_{\text{ref}}$; apart from rank bounds, candidate depth, and sketch dimension, the higher-budget settings retain the default calibration and allocation protocol. The run seeds 42, 43, and 44 permute examples and assign their permuted positions cyclically to four groups; the group assignment is shared across target modules within each run.

E.4 Training and Evaluation Configuration

Default Llama-3.1-8B-Base training uses one epoch of AdamW; Table E3 gives the full training, generation, and GSM8K evaluation settings. Baselines use AdamW with $\text{betas} = (0.9, 0.999)$, weight decay 5×10^{-4}, effective batch size 64, a cosine learning-rate schedule, warmup ratio 0.03, decay ratio 0.1, and BF16 mixed precision. The peak learning rate is 5×10^{-5} for all methods except AdaLoRA, which uses 5×10^{-4}.

Llama-2-7B-Base uses the corresponding task training data, generation protocol, and evaluation metric, targeting the $q/k/v/o$ and $up/gate/down$ projections. Training uses AdamW with peak learning rate 2×10^{-5}, $\text{betas} = (0.9, 0.999)$, weight decay 0, and effective batch size 32 (per-device batch size 4 with 8 gradient-accumulation steps). The effective batch sizes in Table E4 describe the Llama-3.1-8B-Base task settings. The Llama-2 parameter budget is recomputed for its base-model architecture.

Method-specific settings. LoRA+ uses a learning-rate ratio of 16 between matrices B and A. LoRA-GA uses stable scaling with $\gamma = 16$ and handles pretrained weights during initialization as prescribed by the original method. AdaLoRA uses initial rank 12, target rank 8, $t_i = 150$, and $t_f = 900$. PiSSA uses fast SVD with 64 iterations. The GoRA comparison follows its original settings (He et al., 2025): $r_{\min} = r_{\text{ref}}/2$, $r_{\max} = 4r_{\text{ref}}$, 64 gradient-accumulation steps for initialization, and a learning-rate ratio of 16 between B and A. Its initialization scaling γ is 5×10^{-2} for non-MetaMathQA tasks at $r_{\text{ref}} = 8$, 8×10^{-2} for MetaMathQA at $r_{\text{ref}} = 8$, 10^{-2} at $r_{\text{ref}} = 32$, and 5×10^{-3} at $r_{\text{ref}} = 128$.

Table E3: Default training and GSM8K evaluation configuration.

Item	Setting
Training data and length	MetaMathQA-100K; maximum sequence length 512
Optimizer	AdamW; learning rate 5×10^{-5}; $\mathrm{betas} = (0.9, 0.999)$; $\mathrm{eps} = 10^{-8}$; weight decay 5×10^{-4}
Steps and batch	At most 1,563 steps; batch size 4; gradient accumulation 16; effective batch size 64
Schedule and stability	47 warmup steps; decay ratio 0.1; gradient clipping 1.0
LoRA and precision	$\alpha = 16$; effective scaling α/r; dropout 0; adapter parameters float32; base model bfloat16 (BF16)
GSM8K evaluation	1,319 test questions; fixed zero-shot chain-of-thought template; strict then relaxed answer matching
Generation and checkpoint	Greedy decoding; temperature 0; $\mathrm{top_p} = 1$; at most 256 generated tokens; final checkpoint evaluated once
Stopping condition	`Question:` after a newline

Table E4: Task-specific training and evaluation differences.

Item	MT-Bench	GSM8K	HumanEval
Training data	WizardLM-52K	MetaMathQA-100K	Code-Feedback-100K
Training scale	1 epoch; effective batch size 64	One pass; effective batch size 64	1 epoch; effective batch size 64
Evaluation	Mean score from GPT-4o, Gemini-1.5-Pro, and Llama-3.1-70B-Instruct judgments, following GoRA (He et al., 2025)	Zero-shot chain of thought and the answer matching of Table E3	Official HumanEval pass@1; one completion per problem
Generation	Two-turn MT-Bench prompts following GoRA (He et al., 2025)	Greedy; max new tokens 256; temperature 0; $\mathrm{top_p} = 1$	Max new tokens 128; temperature 0.8; $\mathrm{top_p} = 0.95$

E.5 Default-run Verification and Environment

Accuracy is computed over all 1,319 GSM8K test questions. For seeds 42–44, the correct counts are 968/1319, 963/1319, and 955/1319. Final rank sums differ because unit-rank parameter costs are not identical across modules; after budget repair, all three runs exactly reach the 6,815,744-parameter reference budget.

The software environment is Python 3.10.20, PyTorch 2.5.1+cu121, Transformers 5.12.1, Accelerate 1.14.0, Datasets 5.0.0, NumPy 2.2.6, and SciPy 1.15.3. The experiment server has four NVIDIA A800 80GB GPUs. Runs use single-process CUDA and scaled dot-product attention (SDPA), with FlashAttention 2 disabled. Settings and results were compiled from resolved runtime configurations, metric records, and final rank files.

F Complete Results and Budget Notes

This appendix gives complete main-result comparisons, parameter-budget information, and per-seed data. Table F1 gives extended Llama-3.1-8B-Base results and parameter budgets, Table F2 gives complete supplementary Llama-2-7B-Base results, Table F3 gives per-seed GSM8K accuracy for the controlled comparisons and higher-reference-budget Llama-3.1-8B-Base settings, and Table F4 summarizes higher-reference-budget CovRA results on both base models.

All comparison results are reproduced by us; Appendix E gives the training and evaluation configuration. Full FT is an unconstrained reference and is excluded from PEFT ranking.

Table F1: Extended Llama-3.1-8B-Base results and parameter budgets.

Method	Trainable params.	Relative budget (%)	MT-Bench ↑	GSM8K ↑	HumanEval ↑
Full FT	≈8.03B	—	5.89 ± 0.18	73.54 ± 0.42	51.63 ± 1.27
LoRA	6.82M	100.00	6.13 ± 0.03	67.78 ± 1.19	42.89 ± 0.35
AdaLoRA†	10.24M	150.24	6.18 ± 0.13	70.48 ± 0.82	42.07 ± 3.23
GoRA†	7.00M	102.70	6.34 ± 0.04	72.71 ± 0.68	48.78 ± 1.83
DoRA†	≈7.14M	≈104.8	6.24 ± 0.10	69.35 ± 0.94	44.11 ± 1.53
PiSSA	6.82M	100.00	6.10 ± 0.08	68.56 ± 1.03	44.31 ± 1.53
LoRA-GA	≈6.82M	≈100.0	6.03 ± 0.06	71.22 ± 0.84	43.70 ± 0.70
CovRA-RI	6.82M	100.00	6.19 ± 0.07	72.48 ± 0.40	46.95 ± 1.83
CovRA	6.82M	100.00	6.33 ± 0.04	72.93 ± 0.50	48.98 ± 1.96
CovRA ($r_{\text{ref}} = 32$)	27.26M	400.00	6.28 ± 0.06	75.66 ± 0.38	50.20 ± 1.76
CovRA ($r_{\text{ref}} = 128$)	109.05M	1600.00	6.30 ± 0.21	75.82 ± 0.38	51.02 ± 1.76

† marks default-setting PEFT baselines whose trainable parameter counts differ from the reference budget. The ≈ symbol identifies approximate parameter counts and relative budgets; it does not establish exact budget equality.

Non-approximate trainable parameter counts are measured from the instantiated models, with implementation settings in Appendix E. The reference parameter budget B^* is the actual LoRA parameter count at uniform reference rank $r_{\text{ref}} = 8$: 6,815,744. Relative budget is only a parameter-count check, not an efficiency metric.

Table F2: Complete supplementary Llama-2-7B-Base results.

Method	MT-Bench ↑	GSM8K ↑	HumanEval ↑
Full FT	5.30 ± 0.11	59.36 ± 0.85	35.37 ± 2.11
LoRA	5.59 ± 0.09	42.08 ± 0.08	15.04 ± 0.35
rsLoRA	5.27 ± 0.04	45.49 ± 0.15	16.26 ± 0.70
DoRA	5.93 ± 0.04	53.37 ± 0.75	19.51 ± 0.61
LoRA+	5.73 ± 0.07	51.96 ± 0.68	18.50 ± 0.70
OLoRA	5.32 ± 0.05	43.47 ± 0.79	17.07 ± 0.00
PiSSA	5.28 ± 0.03	44.38 ± 0.34	16.26 ± 0.35
LoRA-GA	5.90 ± 0.12	53.58 ± 0.42	20.33 ± 0.93
AdaLoRA	5.61 ± 0.07	50.54 ± 1.28	17.89 ± 0.70
GoRA	5.66 ± 0.10	54.03 ± 0.34	24.39 ± 1.22
CovRA	5.83 ± 0.10	54.21 ± 0.65	24.80 ± 1.27

Table F3: Per-seed GSM8K accuracy (%) for controlled comparisons and higher-reference-budget Llama-3.1-8B-Base settings.

Method	Seed 42	Seed 43	Seed 44	Mean ± SD
Uniform	68.01	66.49	68.84	67.78 ± 1.19
Uniform-SelInit	71.19	70.43	70.81	70.81 ± 0.38
CovRA-M-RI	69.75	70.28	70.66	70.23 ± 0.46
CovRA-I-RI	70.66	71.42	71.42	71.17 ± 0.44
CovRA-SD-RI	71.42	71.65	71.42	71.49 ± 0.13
CovRA-RI	72.63	72.78	72.02	72.48 ± 0.40
CovRA	73.39	73.01	72.40	72.93 ± 0.50
CovRA ($r_{\text{ref}} = 32$)	76.04	75.66	75.28	75.66 ± 0.38
CovRA ($r_{\text{ref}} = 128$)	76.19	75.82	75.44	75.82 ± 0.38

Uniform is the uniform-rank baseline in the controlled comparison: standard LoRA with the default parameter budget and random initialization at seeds 42–44. It is the same group of runs as the LoRA row in main-text Table 1. The $r_{\text{ref}} = 32$ and 128 CovRA rows correspond to the higher-reference-budget Llama-3.1-8B-Base settings in Table F4. Mean ± standard deviation is computed from unrounded accuracy values with $\text{ddof} = 1$; differences in main-text Table 3 are aggregated after pairing the internal controlled variants by seed.

Table F4: Higher-reference-budget CovRA results and trainable parameter counts on two base models.

Base model	r_{ref}	Trainable parameters	MT-Bench ↑	GSM8K ↑	HumanEval ↑
Llama-3.1-8B-Base	32	27,262,976	6.28 ± 0.06	75.66 ± 0.38	50.20 ± 1.76
Llama-3.1-8B-Base	128	109,051,904	6.30 ± 0.21	75.82 ± 0.38	51.02 ± 1.76
Llama-2-7B-Base	32	79,953,920	5.79 ± 0.08	56.91 ± 0.55	26.02 ± 1.41
Llama-2-7B-Base	128	319,815,680	5.81 ± 0.10	57.09 ± 0.60	26.63 ± 1.53

MT-Bench, GSM8K, and HumanEval report the mean ± sample standard deviation over seeds 42–44. Per-seed GSM8K results for the higher-reference-budget Llama-3.1-8B-Base settings appear in Table F3. Llama-3.1-8B-Base parameter counts are measured from the instantiated models; the Llama-2-7B-Base higher-budget counts are computed for LoRA targeting the $q/k/v/o$ and $up/gate/down$ projections.

G Supplementary Diagnostics

This appendix supplements Section 4.4, giving full definitions and statistical aggregation for the default-candidate-pool response-overlap diagnostic and its matched permutation null.

G.1 Default Candidate-Pool Response-Overlap Diagnostic

Diagnostic setting. The diagnostic directly uses the default candidate pool created for the default CovRA allocation: $N = 256$ calibration samples, $K = 16$ candidates per module, and 128 $q/k/v/o$ projection modules. Seeds 42–44 reuse the same calibration example set and reflect repeated random candidate construction rather than independent data splits.

Fixed independent ordering. Let $\phi_{m,k}$ denote a sign-separated response in module m. We obtain a fixed order $(a_{m,1}, \ldots, a_{m,K})$ by descending independent energy $\|\phi_{m,k}\|_2^2$. This ordering quantifies repeated counting of already covered responses when independent valuation selects candidates in its own order; conditional ordering would mix the phenomenon with the conditional-marginal correction.

The common $1/N$ factor is omitted from response energies in this diagnostic; this does not affect ordering or energy ratios at a fixed calibration size.

Residual response-energy ratio. Let $\Pi_{m,j-1}$ be the orthogonal projector onto the span of the first $j-1$ ordered candidate responses. We also define the residual response-energy ratio at position j:

$$\gamma_{m,j} = \frac{\|(I - \Pi_{m,j-1})\phi_{m,a_{m,j}}\|_2^2}{\|\phi_{m,a_{m,j}}\|_2^2}, \qquad \gamma_{m,1} = 1. \tag{30}$$

Cumulative response redundancy. For the first k positions, cumulative response redundancy is

$$\rho_m(k) = 1 - \frac{\sum_{j=1}^{k} \left\|(I - \Pi_{m,j-1})\phi_{m,a_{m,j}}\right\|_2^2}{\sum_{j=1}^{k} \left\|\phi_{m,a_{m,j}}\right\|_2^2}. \tag{31}$$

The $\rho(4)$ value uses the first four positions from the same fixed default $K = 16$ candidate pool rather than a separately reconstructed depth-four pool.

Matched permutation null. The null begins with raw signed responses. It independently permutes each candidate's 256 example indices, then applies the same sign-separated encoding and diagnostic. It preserves the response multiset, energy, and sign proportion of each candidate while disrupting sample-level correspondence and covariance. Independent energy and therefore the fixed ordering are unchanged. For every seed–module pair, we average 1,000 permutations.

Statistical aggregation. For every seed–module pair, we compute actual and null statistics. We first average values from seeds 42–44 within each module, then report the median and interquartile interval over the 128 modules. Paired differences are actual minus null within each module before this aggregation; their median need not equal the difference between marginal medians. We do not regard three seeds times 128 modules as 384 independent observations.

The paired module-wise difference is positive for 124/128 modules at $\rho(4)$, 126/128 at $\rho(8)$, and 125/128 at $\rho(16)$.

H Supplementary Experiments and Run Analysis

This appendix reports single-factor sensitivity to candidate depth and calibration size, cross-seed stability of final ranks in the default model–task setting, and absolute timing verifiable from run logs. Tables H1 and H2 report the two sensitivity studies; Figure H1 and Table H3 compare the Seed 42–44 rank configurations visually and quantitatively; and Table H4 reports measured timing for the three default runs.

H.1 Candidate-Depth Sensitivity

With default Llama-3.1-8B-Base/GSM8K, $N = 256$ and $r_{\text{ref}} = 8$, we vary only the number K of candidate rank-one updates directly constructed and conditionally valued per module. All settings use seeds 42–44, retain the 6,815,744 parameter budget, and share the remaining training and allocation protocol; $K = 16$ is the default.

Table H1: Sensitivity to candidate depth K (default: 16). Values are means $\pm$ sample SD over seeds 42–44; GSM8K accuracy is in percent.

K	GSM8K ↑	Allocation time (s) ↓	Initialization time (s) ↓
4	72.55 ± 0.40	710 ± 1	25 ± 2
8	72.91 ± 0.42	781 ± 1	26 ± 2
16	72.93 ± 0.50	985 ± 1	26 ± 2
32	72.71 ± 0.40	1636 ± 4	28 ± 2

Sections 3.2–3.4 define K and allocation beyond the scored prefix. Performance is similar at $K = 16$ and 8, while allocation time is lower than at $K = 32$; this indicates the default $K = 16$ has a reasonable performance–cost profile within the range considered.

H.2 Calibration-Size Sensitivity

With the same default model, task, and training setting, we fix $K = 16$ and $r_{\text{ref}} = 8$ and vary calibration size $N \in \{64, 128, 256, 512\}$ over seeds 42–44. The remaining allocation settings and the 6,815,744 parameter budget are held fixed.

Table H2: Sensitivity to calibration size N (default: 256). Values are means $\pm$ sample SD over seeds 42–44; GSM8K accuracy is in percent.

N	GSM8K ↑	Allocation time (s) ↓	Initialization time (s) ↓
64	72.30 ± 0.39	481 ± 1	25 ± 2
128	72.61 ± 0.39	649 ± 2	26 ± 2
256	72.93 ± 0.50	985 ± 1	26 ± 2
512	72.78 ± 0.40	1656 ± 4	27 ± 2

Mean accuracy increases as N rises from 64 to 256, but does not improve further at $N = 512$, while allocation time grows with N. The default $N = 256$ therefore has a reasonable performance–cost profile within the range examined.

H.3 Cross-Seed Stability of Final Rank Configurations

Seeds 42–44 drive calibration and candidate-construction randomness and are reused in subsequent training. Because final ranks are fixed before training, this section isolates the combined allocation-stage randomness and does not include later training stochasticity.

Table H3: Stability of default CovRA rank configurations.

(a) Per-run rank statistics

Seed	Rank range	Mean rank	Std rank	Total rank
42	2–19	8.328	2.773	1066
43	2–20	8.305	2.830	1063
44	2–20	8.352	2.987	1069

(b) Cross-seed consistency

Metric	Mean $\pm$ Std
Spearman ρ	0.911 ± 0.008
Per-module mean absolute rank difference	0.818 ± 0.027

Final rank sums are 1,066, 1,063, and 1,069. All ranks lie within the preset bounds $[2, 32]$, with observed maxima 19–20. The three pairwise comparisons of complete rank vectors have mean Spearman correlation 0.911 ± 0.008 and per-module mean absolute rank difference below one, indicating

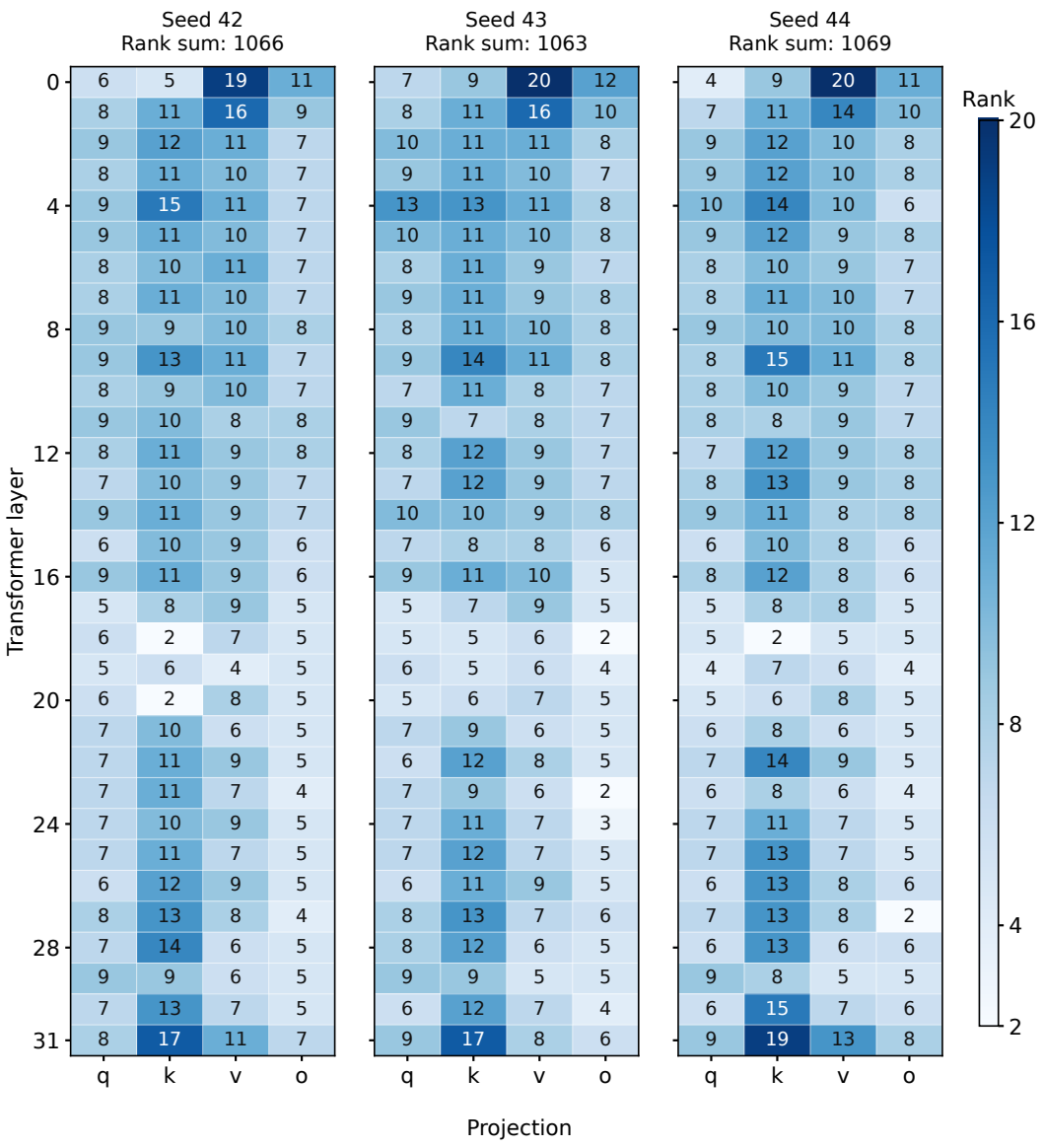


Figure H1: Final rank configurations for the default Llama-3.1-8B-Base/GSM8K setting at seeds 42–44. Colors denote rank rather than parameter count; modules at the lower bound are constrained by the preset bound.

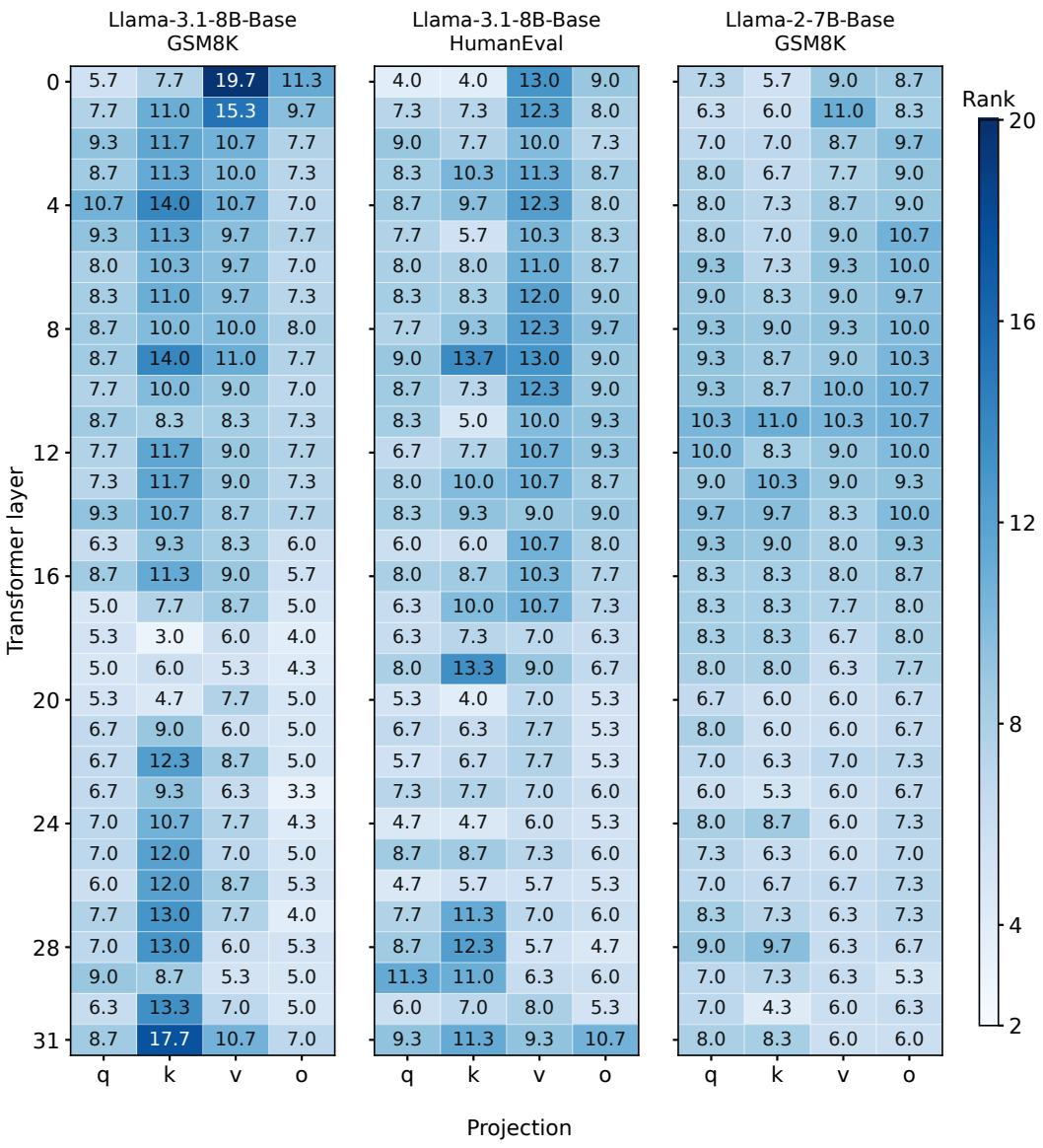


Figure H2: Fully annotated version of Figure 3. Each cell is the mean allocated rank over seeds 42–44 for one layer and attention projection. All panels use the same color scale. Only $q/k/v/o$ projections are shown; the Llama-2 experiments also adapt $up/gate/down$ modules.

high but not perfect allocation-stage stability. Figure 3 compares mean configurations across models and tasks, whereas Figure H1 shows the individual seeds in the default model–task setting.

For the across-seed row in Table 5, the mean and sample standard deviation summarize the three pairwise comparisons among seeds 42–44. The GSM8K versus HumanEval row compares one seed-averaged rank vector per task, yielding a single comparison rather than a sampling distribution to summarize with a standard deviation.

H.4 Measured Absolute Timing

Table H4 reports absolute timing for the default Llama-3.1-8B-Base/GSM8K runs on a server with four NVIDIA A800 80GB GPUs. The implementation uses single-process CUDA. Preprocessing is allocation plus initialization; wall-clock time additionally includes evaluation and other stages and need not equal the listed stage sum exactly.

Table H4: Measured absolute stage times (seconds) for default Llama-3.1-8B-Base/GSM8K runs.

Seed	Allocation	Initialization	Preprocessing	Training	Evaluation	Wall clock
42	984	28	1012	9479	2260	12949
43	984	26	1010	9612	2313	13028
44	986	24	1010	9603	2404	13103
Mean $\pm$ Std	985 ± 1	26 ± 2	1011 ± 1	9565 ± 74	2326 ± 73	13027 ± 77

These absolute times describe CovRA's default implementation. Because comparison methods were not remeasured under one common timing protocol, we do not infer cross-method speed ratios.