# LOW-RANK ADAPTER STRUCTURE SHOULD BE READ OFF, NOT WRITTEN IN

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Low-Rank Adaptation (LoRA) efficiently represents task-specific updates, but an important question remains: *how much structure does a task actually need?* Standard LoRA fixes the number of rank-one components in advance and uses dense factors. We propose SPAR (**SP**arsity **A**nd **R**ank, adaptively read), a two-stage framework that uses task information to determine component counts and factor sparsity. During fine-tuning, Adaptive Rank compares each module's task-loss gradient signal with its historical peak to guide component growth and stopping. After fine-tuning, Adaptive Sparsity refits each learned adapter update with sparse factors to obtain a compact representation. Component counts and coordinate supports are selected through input-aware reconstruction and model-level task-quality checks, without a prescribed coordinate keep ratio. Thus, SPAR uses task information to determine how much capacity to build during learning and how much structure to retain afterward. On eight GLUE tasks with DeBERTaV3-Base, Adaptive Rank achieves a higher average score than existing LoRA baselines with fewer factor parameters. Adaptive Sparsity further reduces the mean number of nonzero factor parameters by 33.4% while maintaining task performance. We also demonstrate that SPAR is effective with Qwen3-4B-Base on the mathematical reasoning benchmarks MATH500 and AIME2024. The two stages also apply to other adapter methods, as illustrated by adaptive capacity allocation in LoRA and LoRA+ and sparse refitting of AdaLoRA and IncreLoRA updates. These results support our central view: adapter structure can be *read off* from task-dependent signals rather than *written in* as a prescribed final rank and coordinate keep ratio. Source code is available at https://anonymous.4open.science/r/SPAR-6F72/.

## 1 INTRODUCTION

Fine-tuning adapts pretrained models to the specific objectives and data distributions of downstream tasks. However, storing and serving a full model for each task becomes increasingly costly as model size and the number of tasks grow. Parameter-efficient fine-tuning addresses this cost with small trainable modules (Houlsby et al., 2019). Low-rank adaptation (LoRA) has become a standard choice (Hu et al., 2022). It freezes the base model and learns an update $\Delta\mathbf{W} = \mathbf{B}\mathbf{A}^\top$ of rank at most $r$ for each adapted weight matrix, where $\mathbf{A}$ and $\mathbf{B}$ are trainable input- and output-factor matrices. Task data determine the values in this update, but the practitioner still writes part of its structure in before learning. At the component level, the rank parameter $r$ sets the number of rank-one terms. At the coordinate level, standard LoRA uses dense factors, whereas sparse variants retain only a subset of factor coordinates, making the number of retained coordinates an additional structural choice. Together, component counts and factor sparsity characterize the adapter's structure.

One approach to determining adapter structure is to search over candidate configurations (Tribes et al., 2023; Baek et al., 2026). Such methods use task data to select a configuration from a predefined search space, requiring the evaluation of multiple candidates. Another approach directly adjusts or estimates adapter capacity using task or model signals. Some methods redistribute rank across modules through masking or explicit pruning and regrowth (Zhang et al., 2023b; Chang et al., 2025), add components incrementally (Zhang et al., 2023a), prune components from initially high-rank adapters (Yu et al., 2025), or deactivate components through learned gates (Ding et al., 2023).



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.

Other methods use the base model's geometry, gradients, or activations to estimate module-wise ranks before fine-tuning (Ed-dib et al., 2025; He et al., 2025; Paischer et al., 2025). At the coordinate level, RoseLoRA selects factor coordinates while gradually reducing the retained fraction toward a prescribed target (Wang et al., 2024). These methods use different external controls to guide or constrain adapter structure (see Appendix H for a broader comparison). A target rank, total budget, or keep ratio specifies how much capacity to retain; a capacity bound instead constrains the range within which the final adapter structure is determined from data.

To guide these structural decisions, many adaptive-rank methods assess component importance using pruning-style scores (Molchanov et al., 2017; Michel et al., 2019; Sanh et al., 2020; Sun et al., 2024; Zhang et al., 2023b) or learned gate magnitudes (Ding et al., 2023). These quantities indicate which components to remove first, while a prescribed budget or cutoff determines where removal stops. Their raw scales can also vary across layers, training stages, and tasks, so a fixed cutoff on unnormalized scores need not have a consistent interpretation (Michel et al., 2019; Dong et al., 2019). AROMA offers an exception: its stopping rule determines component counts from the relative magnitude of the newest fitted update, a measure of parameter movement (Sheng et al., 2025). However, component counts capture only one level of adapter structure. Adaptive-rank methods typically retain dense factors, whereas methods such as RoseLoRA select factor coordinates using a prescribed keep ratio (Wang et al., 2024). This raises a joint question: *how should component counts and factor supports be adapted to the task?*

To address this question, we propose SPAR (**SP**arsity **A**nd **R**ank, adaptively read), a two-stage framework that combines task-guided capacity growth with input-aware sparse refitting. Our central idea is to use task information to determine adapter structure in both stages. During the fine-tuning stage, each module compares its current task-loss gradient signal with its historical values to guide component growth and stopping, allowing its component count to adapt during optimization. During the subsequent refitting stage, we seek a compact sparse representation of each learned adapter update, using input-aware reconstruction and model-level task-quality criteria to determine which structure to retain. In this sense, structure is *read off* from task-dependent signals rather than fixed in advance by a single rank and coordinate keep ratio. Our main contributions are threefold:

1. **Task-guided rank adaptation.** We develop Adaptive Rank, which uses a task-loss gradient signal to quantify the maximum predicted first-order loss decrease from a unit-norm rank-one perturbation. Each module compares this signal with its historical peak to guide component growth and stopping during joint fine-tuning, allowing component counts to adapt to the task without prescribing a final rank.
2. **Input-aware sparse refitting.** We develop Adaptive Sparsity, which refits complete learned updates with new sparse factors, separating the retained representation from the training-time factorization. It selects component counts and factor supports through input-aware reconstruction and cumulative model-level task-quality checks, without prescribing a coordinate keep ratio.
3. **Empirical validation across tasks and adapter methods.** On eight GLUE tasks with DeBERTaV3-Base, Adaptive Rank achieves a higher mean score than the compared baselines with complete eight-task results. Adaptive Sparsity further reduces the mean number of nonzero factor parameters by 33.4% while maintaining mean task performance. We also validate SPAR on mathematical reasoning with Qwen3-4B-Base and demonstrate that both stages can be applied independently to existing adapter methods.

## 2 Related Work

We review how existing methods determine adapter structure at the component and coordinate levels, focusing on which structural choices are data-dependent and which are specified in advance. We then discuss sparse regression and low-rank reconstruction techniques relevant to our refitting stage.

**Adaptive-rank LoRA.** Adaptive-rank methods differ in when ranks are determined and how capacity is allocated across modules. During fine-tuning, AdaLoRA prunes singular triplets according to a prescribed total-rank budget schedule, while ElaLoRA redistributes rank through paired pruning and expansion (Zhang et al., 2023b; Chang et al., 2025). IncreLoRA incrementally adds components to selected modules until a target total rank is reached (Zhang et al., 2023a), whereas PrunedLoRA

prunes components from initially high-rank adapters toward a target rank (Yu et al., 2025). Other approaches adapt component usage or rank allocation subject to fixed per-module ranks, prescribed rank intervals, or maximum-rank parameterizations (Gu et al., 2025; Tripathi et al., 2026; Duan et al., 2026).

Rank selection can also be formulated as hyperparameter optimization (Tribes et al., 2023; Baek et al., 2026) or architecture search (Zhou et al., 2024; Chang et al., 2024; Munoz et al., 2024; Cho et al., 2026), selecting task-dependent configurations from predefined search spaces. Alternatively, pre-fine-tuning estimators derive module-wise ranks from model geometry, gradients, activations, or representational sensitivity and keep the resulting allocation fixed during training (Ed-dib et al., 2025; He et al., 2025; Paischer et al., 2025; Liu et al., 2026). Sequential approaches such as BoostLoRA and XGBLoRA accumulate updates through repeated low-rank training and merging under round or step budgets; BoostLoRA additionally supports early stopping (Anantha et al., 2026; Zhang et al., 2024). InRank, developed for low-rank training rather than adapter tuning, expands a cumulative update using an explained-ratio criterion (Zhao et al., 2023). AROMA is particularly relevant to our work: it sequentially trains and merges rank-one updates, stopping growth according to the magnitude of the newest update relative to the current weight matrix (Sheng et al., 2025). SPAR uses a task-loss gradient signal that measures local first-order descent opportunity along a rank-one direction. Each module compares this signal with its own historical maximum to determine whether growth should continue.

**Sparse LoRA.** Sparsity in low-rank adaptation can operate on entire rank-one components or on individual coordinates within their factors. SoRA uses learned gates to deactivate components while retaining dense factors (Ding et al., 2023). RoseLoRA selects row- and column-wise sparsity patterns in LoRA factors while gradually reducing the retained fraction toward a prescribed target (Wang et al., 2024). GRASP LoRA learns a global pruning ratio from validation-loss feedback for cross-lingual transfer (Hassan & Chen, 2026). These methods differ in both the granularity of sparsity and whether its amount is prescribed or learned.

Related approaches directly sparsify updates to base-model weights. SHiRA fixes a mask before training (Bhardwaj et al., 2024), while random-mask methods optimize randomly selected weights (Xu & Zhang, 2024; Rios et al., 2025). Their sparsity is defined over base-weight entries. SPAR's Adaptive Sparsity refits complete learned adapter updates with new sparse factors, selecting both component counts and factor supports without prescribing a final coordinate keep ratio.

**Sparse regression and low-rank reconstruction.** Reduced-rank regression constrains the rank of a coefficient matrix (Izenman, 1975), while sparse regression selects variables or coefficients through penalties such as the Lasso's $\ell_1$ penalty (Tibshirani, 1996). Subsequent formulations jointly address variable selection and rank control (Chen & Huang, 2012; Bunea et al., 2012). Stagewise methods construct regularization paths through incremental updates (Efron et al., 2004; Zhao & Yu, 2007). In tensor regression, He et al. (2018) sequentially fit sparse rank-one terms to the residuals left by preceding terms. Our sparse refitting procedure draws on this stagewise residual-fitting construction.

Low-rank approximation also supports model initialization and compression, with different choices of approximation target and timing. PiSSA initializes trainable factors from the principal singular components of pretrained weights while freezing the residual (Meng et al., 2024), and ConsNo-TrainLoRA (CNTLoRA) constructs data-driven initializations before adapter training (Das et al., 2025). DeltaZip compresses full fine-tuning deltas (Yao et al., 2025), whereas EoRA fits low-rank compensation for errors introduced by base-weight compression (Liu et al., 2024a). SPAR refits trained adapter updates by reconstructing their action on observed module inputs and checking model-level task quality after replacement. The reconstruction criterion measures local approximation fidelity, while cumulative task-quality checks assess the effect of accepted replacements on the adapted model.

## 3 METHOD

We consider adapting a pretrained model through task-specific updates to selected linear modules while keeping the pretrained weights frozen. For each adapted module $m$, let $\mathbf{W}_m \in \mathbb{R}^{q_m \times p_m}$ denote its pretrained weight matrix, $\Delta\mathbf{W}_m \in \mathbb{R}^{q_m \times p_m}$ its adapter update, and $\mathbf{h}_m \in \mathbb{R}^{p_m}$ its input.

Omitting any unchanged bias, the module output is

$$\mathbf{y}_m = \mathbf{W}_m \mathbf{h}_m + \Delta \mathbf{W}_m \mathbf{h}_m. \tag{1}$$

SPAR determines the structure of these updates in two stages. During fine-tuning, *Adaptive Rank* grows each module's component count using a task-loss gradient signal relative to the module's historical peak. After fine-tuning, *Adaptive Sparsity* refits each complete learned update with sparse factors, selecting component counts and coordinate supports through input-aware reconstruction and cumulative model-level task-quality checks. This design separates the parameterization used to learn an update from the representation retained after refitting. We describe the two stages in turn below and provide the corresponding algorithms and implementation details in Appendices A and B.

### 3.1 Adaptive Rank

Adaptive Rank adjusts each module's component count during fine-tuning. It uses the largest singular value of the update gradient to measure the local first-order opportunity for a rank-one descent step, and compares this signal with the module's historical peak to guide growth and stopping. Each insertion preserves the model's current function.

**Expandable components.** Let $r_m$ be the current number of components in module $m$. We parameterize its update as

$$\Delta \mathbf{W}_m = \alpha \sum_{j=1}^{r_m} e_{m,j} \mathbf{b}_{m,j} \mathbf{a}_{m,j}^\top, \tag{2}$$

where $\mathbf{a}_{m,j} \in \mathbb{R}^{p_m}$ and $\mathbf{b}_{m,j} \in \mathbb{R}^{q_m}$ are trainable input and output factors, and $e_{m,j} \in \mathbb{R}$ is a trainable scalar coefficient. The component count $r_m$ upper-bounds the algebraic rank of $\Delta \mathbf{W}_m$. The scale $\alpha > 0$ remains fixed as components are added, so insertion does not rescale existing terms. Each module starts with $r_{m,0}$ components and $\Delta \mathbf{W}_m = \mathbf{0}$, preserving the pretrained module's output at initialization (Appendices A.1 and C).

**A gradient signal for growth.** Let $\mathcal{L}$ be the task loss on the current effective batch, comprising the examples contributing to one optimizer update. Define the gradient with respect to the complete module update as

$$\mathbf{G}_m := \nabla_{\Delta \mathbf{W}_m} \mathcal{L} \in \mathbb{R}^{q_m \times p_m}. \tag{3}$$

For a step size $\eta > 0$, consider a rank-one perturbation $-\eta \mathbf{b}\mathbf{a}^\top$ to $\Delta \mathbf{W}_m$, where $\mathbf{a} \in \mathbb{R}^{p_m}$ and $\mathbf{b} \in \mathbb{R}^{q_m}$ are unit vectors. This perturbation has Frobenius norm $\eta$ (i.e., $\| -\eta \mathbf{b}\mathbf{a}^\top \|_F = \eta$). Holding all other parameters fixed, the first-order approximation at the current parameters predicts a loss decrease of $\eta \mathbf{b}^\top \mathbf{G}_m \mathbf{a}$. Among such rank-one perturbations, the largest predicted decrease per unit Frobenius norm is

$$\max_{\|\mathbf{a}\|_2 = \|\mathbf{b}\|_2 = 1} \mathbf{b}^\top \mathbf{G}_m \mathbf{a} = \sigma_1(\mathbf{G}_m), \tag{4}$$

where $\sigma_1(\mathbf{G}_m)$ is the largest singular value of $\mathbf{G}_m$. The maximum is attained when $\mathbf{a}$ and $\mathbf{b}$ are the corresponding right and left singular vectors, respectively (Appendix A.2). We use this local first-order signal to guide component growth.

**Growth relative to the module's history.** Let $\widehat{\sigma}_m^{(t)} \geq 0$ denote the estimate of $\sigma_1(\mathbf{G}_m)$ at the $t$-th growth examination of module $m$. At the first examination, the module initializes its historical peak and adds one component. From the second examination onward, growth stops permanently if the current score falls below a fraction $\tau$ of the previous peak:

$$\widehat{\sigma}_m^{(t)} < \tau \max_{1 \leq s < t} \widehat{\sigma}_m^{(s)}, \qquad t \geq 2, \quad 0 < \tau < 1. \tag{5}$$

Otherwise, the module updates its historical peak to include the current score and adds one component. The comparison uses only previous scores in its reference. All modules share the stopping ratio $\tau$ within a run but maintain separate histories. Components remain trainable after growth stops. Modules whose stopping condition is not met continue under this rule until the fine-tuning horizon; Appendix A.3 describes the zero-score boundary case.

**Scheduling growth examinations.** To allow a newly inserted component to optimize before reassessing growth, we schedule examinations using its coefficient sensitivity. Let $u$ count local optimizer updates since initialization or the latest insertion. Before the optimizer step, we compute

$$\zeta_{m,u} := \left| e^{(u)}_{m,r_m} \frac{\partial \mathcal{L}}{\partial e^{(u)}_{m,r_m}} \right|, \tag{6}$$

where $e^{(u)}_{m,r_m}$ is the newest component's coefficient and $\mathcal{L}$ is evaluated on the current effective batch. Before the first insertion, we monitor the last-indexed initial component. This sensitivity measures the magnitude of the first-order loss response per unit relative change in the coefficient. After a minimum interval, an examination is triggered when consecutive sensitivity-window means are sufficiently close and the recent mean exceeds a fixed fraction of its running peak. A maximum waiting interval triggers an examination regardless of these tests. Thus, sensitivity determines *when to examine*, while the spectral history determines *whether to grow*. Insertion resets the sensitivity statistics and local counter but retains the spectral peak. Appendix A.4 specifies the tests.

**Function-preserving insertion.** When growth is accepted, spectral estimation also provides an estimated unit leading right singular vector $\widehat{\mathbf{v}}_{m,1} \in \mathbb{R}^{p_m}$ of the current $\mathbf{G}_m$. We initialize the new component as

$$\mathbf{a}_{m,r_m+1} = c_a \widehat{\mathbf{v}}_{m,1}, \qquad \mathbf{b}_{m,r_m+1} = \mathbf{0}, \qquad e_{m,r_m+1} = e_0, \tag{7}$$

where $c_a > 0$ is the input-factor scale and $e_0 \neq 0$ is the initial scalar coefficient, both fixed within a run. The input factor follows the gradient's leading right singular direction, while the zero output factor makes the inserted contribution exactly zero. Insertion therefore leaves the current model function unchanged. Subsequent optimizer steps train new and existing components jointly. Appendix A.5 describes optimizer initialization and learning-rate schedules for new components; Algorithm 1 in Appendix A.6 specifies the order of gradient computation, insertion, and optimization.

### 3.2 Adaptive Sparsity

The factorization learned during fine-tuning may contain more structure than is needed to represent the resulting update. Adaptive Sparsity refits the complete update with new sparse factors, whether the source adapter was trained by Adaptive Rank or another method. It constructs candidate representations using input-aware reconstruction, then checks their cumulative effect on task quality in the full model. Both component counts and factor-coordinate supports can change, without a prescribed coordinate keep ratio.

**Sparse representation.** Let $\Theta_{\text{source}}$ denote the source model after fine-tuning, and let $\Delta\mathbf{W}_m$ be its complete update for module $m$, including all scaling factors. We approximate this update by

$$\widetilde{\Delta\mathbf{W}}_m = \sum_{k=1}^{\tilde{r}_m} \widetilde{e}_{m,k} \widetilde{\mathbf{b}}_{m,k} \widetilde{\mathbf{a}}^{\top}_{m,k}, \qquad \|\widetilde{\mathbf{a}}_{m,k}\|_1 = \|\widetilde{\mathbf{b}}_{m,k}\|_1 = 1, \quad \widetilde{e}_{m,k} > 0, \tag{8}$$

where $\widetilde{\mathbf{a}}_{m,k} \in \mathbb{R}^{p_m}$ and $\widetilde{\mathbf{b}}_{m,k} \in \mathbb{R}^{q_m}$ are sparse factors that may have either sign, and $\widetilde{e}_{m,k}$ is the term's amplitude. The unit-$\ell_1$ constraints make $\widetilde{e}_{m,k}$ equal to the entrywise $\ell_1$ norm of its rank-one term. Candidate representations satisfy $1 \leq \tilde{r}_m \leq r_m$, where $r_m$ now denotes the source adapter's component count. Compactness is measured by the number of retained components and the total number of nonzero factor entries.

**Input-aware reconstruction.** We measure how well a proposed update $\mathbf{M} \in \mathbb{R}^{q_m \times p_m}$ reproduces the source update's output on $N_m$ module inputs $\mathbf{h}_{m,i}$ collected from $\Theta_{\text{source}}$:

$$\mathcal{E}_m(\mathbf{M}; \mathbf{H}_m) = \frac{1}{N_m} \sum_{i=1}^{N_m} \|(\Delta\mathbf{W}_m - \mathbf{M})\mathbf{h}_{m,i}\|_2^2 = \|\Delta\mathbf{W}_m - \mathbf{M}\|^2_{\mathbf{H}_m}, \tag{9}$$

where $\mathbf{H}_m = \frac{1}{N_m} \sum_i \mathbf{h}_{m,i}\mathbf{h}^{\top}_{m,i}$ is the uncentered input second moment and $\|\mathbf{X}\|^2_{\mathbf{H}_m} := \operatorname{tr}(\mathbf{X}\mathbf{H}_m\mathbf{X}^{\top})$. The quantity $\|\mathbf{X}\|_{\mathbf{H}_m}$ is a seminorm when $\mathbf{H}_m$ is singular and reduces to the Frobenius norm when $\mathbf{H}_m = \mathbf{I}$. It weights reconstruction by the observed input distribution. The moment is computed once from the source model and remains fixed throughout refitting.

**Data partition and usage.** We partition the training examples into three fixed, example-disjoint subsets: $\mathcal{T}_{\text{fit}}$, $\mathcal{T}_{\text{sel}}$, and $\mathcal{A}$. Inputs from the first two subsets yield $\mathbf{H}_m^{\text{fit}}$ for fitting and $\mathbf{H}_m^{\text{sel}}$ for all reconstruction-based selection decisions. The subset $\mathcal{A}$ is used for full-model task-quality checks. External validation and test data are not used for these refitting decisions. Appendix B.1 describes the partition and input collection.

**Stagewise sparse fitting.** We adapt the sequential rank-one construction and coordinate-stagewise path of He et al. (2018) to input-weighted reconstruction. At step $k$, the previous terms remain fixed and a new term is fitted to $\mathbf{R}_{m,k-1} = \Delta\mathbf{W}_m - \widetilde{\Delta\mathbf{W}}_{m,k-1}$, where $\widetilde{\Delta\mathbf{W}}_{m,k-1}$ is their sum and $\widetilde{\Delta\mathbf{W}}_{m,0} = \mathbf{0}$. For $\mathbf{T}_{m,k} = \widetilde{e}_{m,k}\widetilde{\mathbf{b}}_{m,k}\widetilde{\mathbf{a}}_{m,k}^\top$ under the constraints in Eq. (8), the fitting objective is

$$\frac{1}{2}\|\mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}\|_{\mathbf{H}_m^{\text{fit}}}^2 + \lambda_{m,k}\widetilde{e}_{m,k}, \qquad \lambda_{m,k} \geq 0. \tag{10}$$

With $\widetilde{\mathbf{b}}_{m,k}$ fixed, the scaled input factor $\mathbf{w} = \widetilde{e}_{m,k}\widetilde{\mathbf{a}}_{m,k}$ satisfies $\widetilde{e}_{m,k} = \|\mathbf{w}\|_1$, so the objective has a quadratic reconstruction term and an $\ell_1$ penalty $\lambda_{m,k}\|\mathbf{w}\|_1$. Fixing the input factor gives the analogous form for the scaled output factor. This conditional form explains how the amplitude penalty promotes sparse coordinates.

The joint factorization problem is nonconvex. We explore reconstruction–sparsity tradeoffs along a coordinate-stagewise path with nonincreasing $\lambda_{m,k}$ and record a finite set of states. Starting with one nonzero entry per factor, forward moves activate coordinates or increase their magnitudes; backward moves shrink active coordinates and can set them to zero. These states are alternative versions of the same new term. They serve two roles: extending the residual sequence and selecting the last term of a candidate representation. Appendix B.2 specifies the solver and sampling rules.

**Building a shared residual sequence.** To define the next residual, we hold earlier terms fixed and select the recorded version of the current rank-one term that yields the largest reduction in reconstruction error on the selection inputs. This choice prioritizes reconstruction accuracy when extending the shared sequence. For each reconstruction tolerance, we separately select a sparse version of the final term, as described below. For a proposed update $\mathbf{M}$, define its normalized squared reconstruction error as

$$\overline{\mathcal{E}}_m(\mathbf{M}) = \frac{\mathcal{E}_m(\mathbf{M}; \mathbf{H}_m^{\text{sel}})}{Z_m}, \qquad Z_m = \mathcal{E}_m(\mathbf{0}; \mathbf{H}_m^{\text{sel}}) + \epsilon_{\text{den}}, \tag{11}$$

where $\epsilon_{\text{den}} > 0$ is a numerical stabilizer, and $Z_m$ remains fixed throughout construction, expressing errors and gains on the scale of the source update's output energy.

Among the recorded positive-amplitude versions of $\mathbf{T}_{m,k}$, we choose the one minimizing $\overline{\mathcal{E}}_m(\widetilde{\Delta\mathbf{W}}_{m,k-1} + \mathbf{T}_{m,k})$ as the *continuation term* $\mathbf{T}_{m,k}^{\text{cont}}$. Its normalized gain is

$$g_{m,k} = \overline{\mathcal{E}}_m(\widetilde{\Delta\mathbf{W}}_{m,k-1}) - \overline{\mathcal{E}}_m(\widetilde{\Delta\mathbf{W}}_{m,k-1} + \mathbf{T}_{m,k}^{\text{cont}}). \tag{12}$$

We admit the term only if $g_{m,k} \geq g_{\min} > 0$, ensuring a decrease of at least $g_{\min}$ in normalized selection error. We then set $\widetilde{\Delta\mathbf{W}}_{m,k} = \widetilde{\Delta\mathbf{W}}_{m,k-1} + \mathbf{T}_{m,k}^{\text{cont}}$ and continue fitting the new residual. Construction stops when no positive-amplitude state is available, the gain test fails, or $r_m$ terms have been admitted. All reconstruction tolerances reuse this sequence and its recorded alternatives.

**Selecting compact candidates.** Because the reconstruction accuracy needed to preserve task performance is unknown, we use a finite set $\mathcal{R}$ of nonnegative tolerances, fixed before refitting, to construct candidates for full-model checks. For $\rho \in \mathcal{R}$, let $\tilde{r}_m(\rho)$ be the length of the shortest nonempty prefix of admitted continuation terms satisfying $\overline{\mathcal{E}}_m(\widetilde{\Delta\mathbf{W}}_{m,k}) \leq \rho$. If no prefix qualifies, that tolerance produces no candidate.

For each qualifying prefix, we keep its first $\tilde{r}_m(\rho) - 1$ continuation terms fixed and revisit the recorded versions of its last term. Among versions satisfying both the full-prefix error tolerance $\rho$ and a normalized gain of at least $g_{\min}$ over the preceding prefix, we choose the one with the fewest nonzero entries across its two factors. Denoting this *terminal term* by $\mathbf{T}_m^{\text{term}}(\rho)$, the candidate is

$$\widetilde{\Delta\mathbf{W}}_m^{(\rho)} = \sum_{k=1}^{\tilde{r}_m(\rho)-1} \mathbf{T}_{m,k}^{\text{cont}} + \mathbf{T}_m^{\text{term}}(\rho). \tag{13}$$

The continuation term is itself feasible, so the terminal selection has at least one feasible state. The substitution changes only the candidate; the shared sequence and later residuals remain unchanged. Thus, component counts are selected through prefix length, and factor supports through the sparse path and terminal selection. Appendix B.3 gives the feasibility checks and tie-breaking rules.

**Cumulative task-quality checks.** Accepted replacements can change downstream inputs, whereas reconstruction moments remain fixed at the source model. We therefore check each candidate in the assembled model. Let $\mathbf{q}_{\mathcal{A}}(\Theta)$ collect the declared task metrics on $\mathcal{A}$ under a fixed evaluation protocol, with all metrics oriented so that larger values are better. Starting from $\Theta_{\text{source}}$, we visit modules in a fixed order and test their valid candidates in increasing $\rho$, from stricter to looser reconstruction tolerances. Each trial includes all previously accepted replacements. We accept the first candidate satisfying

$$\mathbf{q}_{\mathcal{A}}(\Theta_{\text{cand}}) \succeq \mathbf{q}_{\mathcal{A}}(\Theta_{\text{source}}), \tag{14}$$

where $\succeq$ denotes componentwise comparison against the fixed source-model reference. Rejected trials are rolled back; if no candidate passes, the module retains its source representation. Accepted factors receive no further optimization. Algorithm 2 summarizes the procedure, and Appendix B.4 describes the acceptance rules.

# 4 Experiments

We evaluate SPAR in terms of task performance, adapter structure, and reconstruction fidelity. We compare alternative reconstructions of the same learned updates and evaluate Adaptive Rank and Adaptive Sparsity separately with existing adapter methods.

## 4.1 Experimental Setup

**Models and tasks.** We evaluate DeBERTaV3-Base (He et al., 2023) on eight GLUE tasks (Wang et al., 2019), following the task-specific setups used by AdaLoRA (Zhang et al., 2023b). We also fine-tune Qwen3-4B-Base (Qwen Team, 2025) on the 20,480 examples in Math-CoT-20k (Ren et al., 2026) and evaluate on MATH500 (Lightman et al., 2024) and AIME2024 (Zhang & Math-AI Team, 2024), following Ren et al. (2026).

**Comparisons.** For GLUE, we compare SPAR with fixed-structure, adaptive-rank, and other adaptive baselines, retaining results only when their evaluation metrics are compatible with ours (see Table 1). These baselines retain the training and reporting protocols of their original studies. For Qwen3-4B-Base, we compare SPAR with fixed-rank LoRA (Hu et al., 2022) and adaptive-rank AROMA (Sheng et al., 2025) using the same training data and evaluation protocol. Adapter ranks $r = 8$ and $r = 16$ follow Gong et al. (2026) and Ju et al. (2025), respectively. We include the pretrained reference from Ren et al. (2026). SPAR w/o AS denotes Adaptive Rank alone. Appendix C details training and evaluation.

**Reporting.** All experiments conducted in this work use three random seeds, and we report their mean scores. GLUE averages include all eight tasks and are reported only for complete rows. For mathematical reasoning, avg@3 on MATH500 and avg@10 on AIME2024 average correctness over sampled responses and problems, then over training seeds. Appendices C.1 and C.2 define the task metrics. Refitting uses separate subsets of training data for fitting, reconstruction selection, and model-level task-quality checks; external validation and test data are excluded from these refitting decisions.

## 4.2 Task Performance and Learned Capacity

**Language understanding.** As shown in Table 1, Adaptive Rank (SPAR w/o AS) achieves an eight-task average of 89.99, outperforming all baselines with complete eight-task results. It also achieves the highest RTE accuracy, 90.25. Adaptive Sparsity subsequently reduces the mean number of nonzero factor parameters from 0.916M to 0.610M, a reduction of 33.4%, while maintaining the mean task score (89.99 before refitting and 90.01 afterward). After refitting, the reported mean score is unchanged or slightly higher on every task. Figure 1 in Appendix D shows how the returned component counts and allocated factor parameters vary across tasks, and how refitting changes each

Table 1: GLUE results with DeBERTaV3-Base. SPAR shows three-seed means with sample standard deviations below. Full FT, HAdapter, PAdapter, BitFit, and LoRA scores follow Zhang et al. (2023b); other baselines follow their cited sources. Avg. is the unweighted eight-task mean for complete rows. Bold/underlining mark best/second-best reported scores, including ties; dashes indicate unavailable results under the stated metrics.

| Method | MNLI Acc. ↑ | SST-2 Acc. ↑ | CoLA MCC ↑ | QQP Acc. ↑ | QNLI Acc. ↑ | RTE Acc. ↑ | MRPC Acc. ↑ | STS-B Corr. ↑ | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| Full FT | 89.90 | 95.63 | 69.19 | **92.40** | 94.03 | 83.75 | 89.46 | 91.60 | 88.25 |
| HAdapter (Houlsby et al., 2019) | 90.13 | 95.53 | 68.64 | 91.91 | 94.11 | 84.48 | 89.95 | 91.48 | 88.28 |
| PAdapter (Pfeiffer et al., 2021) | 90.33 | 95.61 | 68.77 | 92.04 | 94.29 | 85.20 | 89.46 | 91.54 | 88.41 |
| BitFit (Ben Zaken et al., 2022) | 89.37 | 94.84 | 66.96 | 88.41 | 92.24 | 78.70 | 87.75 | 91.35 | 86.20 |
| LoRA (Hu et al., 2022) | 90.65 | 94.95 | 69.82 | 91.99 | 93.87 | 85.20 | 89.95 | 91.60 | 88.50 |
| AdaLoRA (Zhang et al., 2023b) | 90.76 | 96.10 | 71.45 | 92.23 | 94.55 | 88.09 | 90.69 | 91.84 | 89.46 |
| IncreLoRA (Zhang et al., 2023a) | **90.93** | 96.21 | 71.82 | 92.25 | 94.45 | 88.21 | 91.01 | 91.93 | 89.60 |
| SoRA (Ding et al., 2023) | 90.35 | 95.64 | 71.48 | 92.39 | 94.28 | 87.77 | 91.98 | — | — |
| AFLoRA (Liu et al., 2024b) | 89.88 | 96.22 | 72.01 | 90.81 | 94.42 | 88.09 | 91.91 | — | — |
| DCFT (Zhang et al., 2025) | 89.40 | 95.53 | 71.07 | 89.20 | 93.59 | 88.45 | 90.93 | — | — |
| LoRA-Dash (Si et al., 2025) | 90.55 | **96.99** | 70.78 | 92.39 | 94.22 | 88.09 | 91.18 | — | — |
| DisLoRA (She et al., 2025) | 90.12 | 95.65 | 68.33 | 91.17 | 94.60 | 85.92 | — | — | — |
| GeLoRA (Ed-dib et al., 2025) | 89.84 | 95.05 | 70.96 | — | 93.87 | 81.29 | 89.90 | 91.66 | — |
| ElaLoRA (Chang et al., 2025) | 89.51 | 95.87 | 70.19 | 91.42 | 94.36 | 88.81 | 88.98 | — | — |
| ID3 (Agarwal et al., 2025) | 89.58 | 95.90 | 71.46 | 90.31 | 94.03 | 87.19 | 91.12 | **91.97** | 88.94 |
| ScaLoRA (Zhang et al., 2026) | 90.36 | 95.83 | 69.86 | 92.10 | 94.34 | 87.61 | 90.28 | — | — |
| AdaPaD (Su et al., 2026) | 90.37 | 96.33 | **72.26** | 91.55 | 94.36 | 86.28 | 91.67 | — | — |
| SPAR w/o AS | 90.60 | 96.41 | 72.18 | 92.03 | 94.60 | **90.25** | 91.91 | 91.96 | 89.99 |
| SD | ±0.04 | ±0.07 | ±0.41 | ±0.08 | ±0.10 | ±0.36 | ±0.25 | ±0.07 | — |
| SPAR | 90.62 | 96.41 | 72.18 | 92.03 | **94.64** | **90.25** | **91.99** | **91.97** | **90.01** |
| SD | ±0.06 | ±0.07 | ±0.41 | ±0.10 | ±0.11 | ±0.36 | ±0.14 | ±0.07 | — |

Table 2: Mathematical reasoning with Qwen3-4B-Base. For each trained model, avg@3/avg@10 average correctness over three/ten responses per MATH500/AIME2024 problem, then over problems. The pretrained reference is from Ren et al. (2026). Bold marks the best results.

| Method | MATH500 avg@3 (%) ↑ | AIME2024 avg@10 (%) ↑ |
|---|---|---|
| Pretrained (w/o SFT) | 70.80 | 11.00 |
| LoRA$_{r=8}$ | 70.13 | 13.67 |
| LoRA$_{r=16}$ | 70.60 | 15.67 |
| AROMA | 70.80 | 14.44 |
| SPAR | **73.00** | **17.00** |

learned representation. Appendix G further examines allocations across modules at the end of fine-tuning.

**Mathematical reasoning.** Table 2 reports results with Qwen3-4B-Base. SPAR achieves 73.00 on MATH500 and 17.00 on AIME2024, exceeding the reported means of both fixed-rank LoRA settings and AROMA. These results extend SPAR's evaluation to mathematical reasoning with a larger backbone.

### 4.3 Compactness and Reconstruction Fidelity

To isolate the effect of reconstruction after fine-tuning, we apply three alternative compression methods to the same SPAR w/o AS updates. Post-Squeeze (the post-tuning LoRA-Squeeze variant) reduces update rank (Vulić et al., 2026); PARA allocates ranks across modules using a global singular-value threshold (Kumaravelu et al., 2026); and ComPEFT combines sparsification with ternary quantization (Yadav et al., 2025). We report task score, allocated and nonzero A/B entries,

Table 3: Post-training reconstruction of the same updates learned by SPAR w/o AS. Results are averaged over eight GLUE tasks. $e_X$ denotes the relative update-action error. The best values among reconstructed models are shown in **bold**.

| | | A/B parameters (M) | | | |
|---|---|---|---|---|---|
| Method | Score ↑ | Allocated ↓ | Nonzero ↓ | $r_{\text{total}}$ ↓ | $e_X$ ↓ |
| SPAR w/o AS | 89.99 | 0.916 | 0.916 | 397.5 | — |
| SPAR w/o AS + Post-Squeeze | 88.42 | 0.372 | 0.372 | 161.6 | 0.2585 |
| SPAR w/o AS + PARA | 86.77 | **0.197** | **0.197** | **98.9** | 0.2390 |
| SPAR w/o AS + ComPEFT | 88.11 | 0.916 | 0.458 | 397.5 | 0.8404 |
| SPAR | **90.01** | 0.658 | 0.610 | 289.8 | **0.0423** |

total stored component count $r_{\text{total}}$, and relative update-action error $e_X$. The error $e_X$ measures the relative discrepancy between the source and reconstructed updates' outputs on the same cached source-model inputs; lower values indicate greater reconstruction fidelity. Appendix E.1 defines the metrics and reports task-level results.

Table 3 shows that SPAR reduces the mean allocated factor count from 0.916M to 0.658M and the mean stored component count from 397.5 to 289.8, while retaining a mean task score of 90.01. It also achieves the lowest mean update-action error, $e_X = 0.0423$, among the evaluated reconstructions. The alternative methods retain fewer nonzero factor parameters (0.197M–0.458M) but yield lower mean task scores (86.77–88.42) and larger mean reconstruction errors (0.2390–0.8404). These results compare the reported operating points, which have different retained parameter counts. At these settings, Adaptive Sparsity reduces adapter structure while preserving mean task performance and closely reproducing the learned updates' effects on task inputs.

### 4.4 Structural Adaptation Across Adapter Methods

We further evaluate whether SPAR's two stages can be applied independently to existing adapter methods. We apply Adaptive Rank (AR) to LoRA (Hu et al., 2022) and LoRA+ (Hayou et al., 2024) for task-dependent capacity allocation. For example, adding AR to LoRA+ improves QNLI accuracy from 93.88 to 94.09 while reducing allocated factor parameters from 2.654M to 2.177M. We also use Adaptive Sparsity (AS) to refit the complete updates learned by AdaLoRA (Zhang et al., 2023b) and IncreLoRA (Zhang et al., 2023a), allowing both component counts and factor supports to change after fine-tuning. For source adapters trained in our experiments, AS reduces the mean number of nonzero factor parameters from 0.353M to 0.295M for AdaLoRA and from 1.995M to 1.537M for IncreLoRA, while maintaining their mean task scores. Detailed comparisons, task-wise results, and reconstruction statistics are provided in Appendix F (Tables 6–7).

## 5 Conclusion

We introduced SPAR, a two-stage framework that adapts component counts and factor supports using task-dependent signals. Adaptive Rank compares each module's task-loss gradient signal with its historical peak to guide growth and stopping during fine-tuning. After fine-tuning, Adaptive Sparsity refits complete learned updates with sparse factors through input-aware reconstruction and cumulative model-level task-quality checks, separating the retained representation from the training-time factorization. On eight GLUE tasks with DeBERTaV3-Base, Adaptive Rank achieves a higher mean score than the compared baselines with complete results, while Adaptive Sparsity further reduces the mean number of nonzero factor parameters by 33.4% while maintaining mean task performance. We also demonstrate SPAR's effectiveness on mathematical reasoning with Qwen3-4B-Base and the independent applicability of both stages to existing adapter methods. These findings support our central view: adapter structure can be *read off* from task-dependent signals under explicit training and selection criteria, without prescribing a final rank or coordinate keep ratio. Future work will explore extending SPAR to tensor-structured adapters, using task-dependent signals to guide tensor-rank selection and factor sparsity in higher-order representations.

## AI USE STATEMENT

We used Codex and Claude Code to assist with literature review, manuscript preparation and revision, LaTeX formatting, and code implementation and debugging. We also used these tools to obtain feedback on conceptual framing, mathematical exposition, methodological choices, experimental design, and result interpretation. The authors reviewed and revised the AI-assisted material and take full responsibility for the final manuscript, methods, code, references, experimental results, and conclusions.

## ETHICS STATEMENT

This work studies parameter-efficient adaptation and compression of pretrained language models using publicly available models and benchmark datasets. It does not involve human subjects or the collection of personal or sensitive information. The adapted models may inherit biases and other limitations from the underlying pretrained models and training data.

## REPRODUCIBILITY STATEMENT

Section 3 defines the two stages of SPAR. Appendix A details Adaptive Rank's initialization, examination schedule, component insertion, and optimizer-state handling. Appendix A.2 derives the first-order rank-one signal underlying Adaptive Rank. Appendix B describes Adaptive Sparsity, including input-aware reconstruction, sparse candidate construction and selection, and model-level acceptance. Appendix B.1 specifies the training-data partitions and input-moment estimation. Our implementation, experimental configurations, and example checkpoints are available at https://anonymous.4open.science/r/SPAR-6F72/.

## REFERENCES


Aradhye Agarwal, Suhas Kamasetty Ramesh, Ayan Sengupta, and Tanmoy Chakraborty. Step-by-step unmasking for parameter-efficient fine-tuning of large language models. *Transactions of the Association for Computational Linguistics*, 13:1767–1788, 2025. doi: 10.1162/tacl.a.59. URL https://aclanthology.org/2025.tacl-1.79/.

Raviteja Anantha, Nick Levato, and Layne C. Price. BoostLoRA: Growing effective rank by boosting adapters. *arXiv preprint arXiv:2604.27308*, 2026. URL https://arxiv.org/abs/2604.27308.

Seong-Eun Baek, Jung-Mok Lee, Sung-Bin Kim, and Tae-Hyun Oh. A language-guided Bayesian optimization for efficient LoRA hyperparameter search. In *Proceedings of the 43rd International Conference on Machine Learning*, Seoul, South Korea, July 2026. PMLR. URL https://icml.cc/virtual/2026/poster/64828.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 1–9, 2022. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1/.

Kartikeya Bhardwaj, Nilesh Prasad Pandey, Sweta Priyadarshi, Viswanath Ganapathy, Shreya Kadambi, Rafael Esteves, Shubhankar Borse, Paul Whatmough, Risheek Garrepalli, Mart van Baalen, Harris Teague, and Markus Nagel. Sparse high rank adapters. In *Advances in Neural Information Processing Systems*, volume 37, pp. 13685–13715, 2024. doi: 10.52202/079017-0438. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/18c0102cb7f1a02c14f0929089b2e576-Abstract-Conference.html.

Florentina Bunea, Yiyuan She, and Marten H. Wegkamp. Joint variable and rank selection for parsimonious estimation of high-dimensional matrices. *The Annals of Statistics*, 40(5):2359–2388, 2012. doi: 10.1214/12-AOS1039. URL https://arxiv.org/abs/1110.3556.

Aofei Chang, Jiaqi Wang, Han Liu, Parminder Bhatia, Cao Xiao, Ting Wang, and Fenglong Ma. BIPEFT: Budget-guided iterative search for parameter efficient fine-tuning of large pretrained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7429–7440, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.437. URL https://aclanthology.org/2024.findings-emnlp.437/.

Huandong Chang, Zicheng Ma, Mingyuan Ma, Zhenting Qi, Andrew Sabot, Hong Jiang, and H. T. Kung. ElaLoRA: Elastic & learnable low-rank adaptation for efficient model fine-tuning. *arXiv preprint arXiv:2504.00254*, 2025. URL https://arxiv.org/abs/2504.00254.

Lisha Chen and Jianhua Z. Huang. Sparse reduced-rank regression for simultaneous dimension reduction and variable selection. *Journal of the American Statistical Association*, 107(500):1533–1545, 2012. doi: 10.1080/01621459.2012.734178. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.2012.734178.

Minkyoung Cho, Insu Jang, Shuowei Jin, Zesen Zhao, Adityan Jothi, Ethem F. Can, Min-Hung Chen, and Z. Morley Mao. MARS: Harmonizing multimodal convergence via adaptive rank search. *arXiv preprint arXiv:2603.00720*, 2026. URL https://arxiv.org/abs/2603.00720.

Debasmit Das, Hyoungwoo Park, Munawar Hayat, Seokeon Choi, Sungrack Yun, and Fatih Porikli. ConsNoTrainLoRA: Data-driven weight initialization of low-rank adapters using constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 498–507, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Das_ConsNoTrainLoRA_Data-driven_Weight_Initialization_of_Low-rank_Adapters_using_Constraints_ICCV_2025_paper.html.

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*, pp. 4133–4145, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.252. URL https://aclanthology.org/2023.emnlp-main.252/.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/html/Dong_HAWQ_Hessian_AWare_Quantization_of_Neural_Networks_With_Mixed-Precision_ICCV_2019_paper.html.

Zhibin Duan, Yuhong Wang, Jiahong Fu, Zongsheng Yue, Bo Chen, and Zongben Xu. BaRA: Bayesian adaptive rank allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2606.29184*, 2026. URL https://arxiv.org/abs/2606.29184.

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, 2025. doi: 10.18653/v1/2025.findings-emnlp.1372. URL https://aclanthology.org/2025.findings-emnlp.1372/.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–451, 2004. doi: 10.1214/009053604000000067. URL https://arxiv.org/abs/math/0406456.

Eman Elrefai, Aml Hassan Esmail, and Mohamed Lotfy Elrefai. Gumball at QIAS 2025 shared task: Arabic LLM automated reasoning in Islamic inheritance. In *Proceedings of The Third Arabic Natural Language Processing Conference: Shared Tasks*, pp. 953–959. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.arabicnlp-sharedtasks.132. URL https://aclanthology.org/2025.arabicnlp-sharedtasks.132/.

Xuan Gong, Senmiao Wang, Hanbo Huang, Ruoyu Sun, and Shiyu Liang. VCORE: Variance-controlled optimization-based reweighting for chain-of-thought supervision. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28161–28180. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1298. URL https://aclanthology.org/2026.acl-long.1298/.

Naibin Gu, Zhenyu Zhang, Xiyu Liu, Peng Fu, Zheng Lin, Shuohuan Wang, Yu Sun, Hua Wu, Weiping Wang, and Haifeng Wang. BeamLoRA: Beam-constraint low-rank adaptation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11871–11883, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.582. URL https://aclanthology.org/2025.acl-long.582/.

Besher Hassan and Xiuying Chen. GRASP LoRA: GRPO guided adapter sparsity policy for cross lingual transfer. *arXiv preprint arXiv:2601.06702*, 2026. URL https://arxiv.org/abs/2601.06702.

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*, volume 235 of *Proceedings of Machine Learning Research*, pp. 17783–17806. PMLR, 2024. URL https://proceedings.mlr.press/v235/hayou24a.html.

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*, volume 38, pp. 114074–114102, 2025. doi: 10.52202/085713-3805. URL https://papers.nips.cc/paper_files/paper/2025/hash/a5e4907a40c0dcb8433a35c714ba9d79-Abstract-Conference.html.

Lifang He, Kun Chen, Wanwan Xu, Jiayu Zhou, and Fei Wang. Boosted sparse and low-rank tensor regression. In *Advances in Neural Information Processing Systems*, volume 31,

2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/8d34201a5b85900908db6cae92723617-Abstract.html.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2111.09543.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

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.

Alan Julian Izenman. Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5(2):248–264, 1975. doi: 10.1016/0047-259X(75)90042-1. URL https://www.sciencedirect.com/science/article/pii/0047259X75900421.

Feng Ju, Zeyu Qin, Rui Min, Zhitao He, Lingpeng Kong, and Yi R. Fung. Reasoning path divergence: A new metric and curation strategy to unlock LLM diverse thinking. *arXiv preprint arXiv:2510.26122*, 2025. URL https://arxiv.org/abs/2510.26122.

Vishnuprasadh Kumaravelu, Sunil Gupta, and P. K. Srijith. Post-optimization adaptive rank allocation for LoRA. *arXiv preprint arXiv:2604.27796*, 2026. URL https://arxiv.org/abs/2604.27796.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/aca97732e30bcf1303bc22ac3924fd16-Abstract-Conference.html.

Jiaqi Liu, Haidong Kang, Qihui Zhao, Guo Yu, and Jingchao Wang. RSRA: Training-free probing of representation sensitivity for efficient LoRA rank allocation. *arXiv preprint arXiv:2607.09757*, 2026. URL https://arxiv.org/abs/2607.09757.

Shih-Yang Liu, Maksim Khadkevich, Nai Chit Fung, Charbel Sakr, Chao-Han Huck Yang, Chien-Yi Wang, Saurav Muralidharan, Hongxu Yin, Kwang-Ting Cheng, Jan Kautz, Yu-Chiang Frank Wang, Pavlo Molchanov, and Min-Hung Chen. EoRA: Fine-tuning-free compensation for compressed LLM with eigenspace low-rank approximation. *arXiv preprint arXiv:2410.21271*, 2024a. URL https://arxiv.org/abs/2410.21271.

Zeyu Liu, Souvik Kundu, Anni Li, Junrui Wan, Lianghao Jiang, and Peter Beerel. AFLoRA: Adaptive freezing of low rank adaptation in parameter efficient fine-tuning of large models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 161–167, 2024b. doi: 10.18653/v1/2024.acl-short.16. URL https://aclanthology.org/2024.acl-short.16/.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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, pp. 121038–121072, 2024. doi: 10.52202/079017-3846. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/db36f4d603cc9e3a2a5e10b93e6428f2-Abstract-Conference.html.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/2c601ad9d2ff9bc8b282670cdd54f69f-Abstract.html.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJGCiw5gl.

Juan Pablo Munoz, Jinjie Yuan, Yi Zheng, and Nilesh Jain. LoNAS: Elastic low-rank adapters for efficient large language models. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 10760–10776, Torino, Italia, May 2024. ELRA and ICCL. URL https://aclanthology.org/2024.lrec-main.940/.

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*, volume 38, pp. 46215–46267, 2025. doi: 10.52202/085713-1539. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/41d33bd41fd44bd9dba0e092047cf213-Abstract-Conference.html.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, 2021. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39/.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

Qihan Ren, Peng Wang, Ruikun Cai, Shuai Shao, Dadi Guo, Yuejin Xie, Yafu Li, Quanshi Zhang, Xia Hu, Jing Shao, and Dongrui Liu. Rethinking generalization in reasoning SFT: A conditional analysis on optimization, data, and model capability. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2604.06628.

Jesus Rios, Pierre Dognin, Ronny Luss, and Karthikeyan Natesan Ramamurthy. Sparsity may be all you need: Sparse random parameter adaptation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 18650–18666, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1013. URL https://aclanthology.org/2025.findings-emnlp.1013/.

Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 20378–20389, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/eae15aabaa768ae4a5993a8a4f4fa6e4-Abstract.html.

Yifei She, Xinhao Wei, and Yulong Wang. DisLoRA: Task-specific low-rank adaptation via orthogonal basis from singular value decomposition. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13740–13755, 2025. doi: 10.18653/v1/2025.emnlp-main.694. URL https://aclanthology.org/2025.emnlp-main.694/.

Hao Nan Sheng, Zhi-Yong Wang, Hing Cheung So, and Mingrui Yang. AROMA: Autonomous rank-one matrix adaptation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3443–3459, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.170. URL https://aclanthology.org/2025.emnlp-main.170/.

Chongjie Si, Zhiyi Shi, Shifan Zhang, Xiaokang Yang, Hanspeter Pfister, and Wei Shen. Unleashing the power of task-specific directions in parameter efficient fine-tuning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/c3d101702194a5a345d81cdee4fe966f-Abstract-Conference.html.

Barbara Su, Fangshuo Liao, and Anastasios Kyrillidis. AdaPaD: Adaptive parallel deflation for PEFT with self-correcting rank discovery. *arXiv preprint arXiv:2605.10741*, 2026. URL https://arxiv.org/abs/2605.10741.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/14c856c7a41297804de4c4890e846b25-Abstract-Conference.html.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. doi: 10.1111/j.2517-6161.1996.tb02080.x. URL https://academic.oup.com/jrsssb/article/58/1/267/7027929.

Christophe Tribes, Sacha Benarroch-Lelong, Peng Lu, and Ivan Kobyzev. Hyperparameter optimization for large language model instruction-tuning. *arXiv preprint arXiv:2312.00949*, 2023. URL https://arxiv.org/abs/2312.00949.

Ashutosh Tripathi, Surya Deep Singh, Pranab Sahoo, and Sriparna Saha. LAARA: Layer-aware adaptive rank allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2607.19391*, 2026. URL https://arxiv.org/abs/2607.19391.

Ivan Vulić, Adam Grycner, Quentin de Laroussilhe, and Jonas Pfeiffer. LoRA-Squeeze: Simple and effective post-tuning and in-tuning compression of LoRA modules. *arXiv preprint arXiv:2602.10993*, 2026. URL https://arxiv.org/abs/2602.10993.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1804.07461.

Haoyu Wang, Tianci Liu, Ruirui Li, Monica Xiao Cheng, Tuo Zhao, and Jing Gao. RoseLoRA: Row and column-wise sparse low-rank adaptation of pre-trained language model for knowledge editing and fine-tuning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 996–1008, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.57. URL https://aclanthology.org/2024.emnlp-main.57/.

Jing Xu and Jingzhao Zhang. Random masking finds winning tickets for parameter efficient fine-tuning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 55501–55524. PMLR, 2024. URL https://proceedings.mlr.press/v235/xu24ag.html.

Prateek Yadav, Leshem Choshen, Colin Raffel, and Mohit Bansal. ComPEFT: Compression for communicating parameter efficient updates via sparsification and quantization. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=CovLQwu611.

Xiaozhe Yao, Qinghao Hu, and Ana Klimovic. DeltaZip: Efficient serving of multiple full-model-tuned LLMs. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pp. 110–127. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3717468. URL https://doi.org/10.1145/3689031.3717468.

Xin Yu, Cong Xie, Xunmei Liu, Ziyu Zhao, Tiantian Fan, Lingzhou Xue, and Zhi Zhang. Train large, deploy compact: Structured compression for compact low-rank adaptation. *arXiv preprint arXiv:2510.00192*, 2025. URL https://arxiv.org/abs/2510.00192v3.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. IncreLoRA: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*, 2023a. URL https://arxiv.org/abs/2308.12043.

Jia-Chen Zhang, Yu-Jie Xiong, Chun-Ming Xia, Dong-Hai Zhu, and Xi-He Qiu. Parameter-efficient fine-tuning of large language models via deconvolution in subspace. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3924–3935, 2025. URL https://aclanthology.org/2025.coling-main.265/.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=lq62uWRJjiY.

Y. Zhang and Math-AI Team. American invitational mathematics examination (AIME) 2024. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/math-ai/aime24.

Yifei Zhang, Hao Zhu, Aiwei Liu, Han Yu, Piotr Koniusz, and Irwin King. Less is more: Extreme gradient boost rank-1 adaption for efficient finetuning of LLMs. *arXiv preprint arXiv:2410.19694*, 2024. URL https://arxiv.org/abs/2410.19694.

Yilang Zhang, Xiaodong Yang, Yiwei Cai, and Georgios B. Giannakis. ScaLoRA: Optimally scaled low-rank adaptation for efficient high-rank fine-tuning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/63892.

Jiawei Zhao, Yifei Zhang, Beidi Chen, Florian Schäfer, and Anima Anandkumar. InRank: Incremental low-rank learning. *arXiv preprint arXiv:2306.11250*, 2023. URL https://arxiv.org/abs/2306.11250.

Peng Zhao and Bin Yu. Stagewise Lasso. *Journal of Machine Learning Research*, 8(89):2701–2726, 2007. URL https://www.jmlr.org/papers/v8/zhao07a.html.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. AutoPEFT: Automatic configuration search for parameter-efficient fine-tuning. *Transactions of the Association for Computational Linguistics*, 12:525–542, 2024. doi: 10.1162/tacl_a_00662. URL https://aclanthology.org/2024.tacl-1.29/.

# A Adaptive Rank Details

This appendix gives the spectral-score derivation and implementation details of Adaptive Rank (Section 3.1). It covers initialization and scaling, examination and growth rules, and function-preserving insertion, including optimizer-state and learning-rate handling. Algorithm 1 combines these steps. We retain the notation of Section 3.1: $m$ indexes adapted modules, $p_m$ and $q_m$ are their input and output dimensions, and $r_m$ is the current component count of module $m$.

## A.1 Initialization and Scaling

Each module starts with $r_{m,0}$ components, with $r_{m,0} = 1$ by default; experiment-specific initial counts are specified in Appendix C. All runs begin with $\Delta\mathbf{W}_m = \mathbf{0}$, preserving the pretrained model's function. The scaling $\alpha > 0$ in Eq. (2) remains fixed throughout training, so growth does not rescale existing components.

For the initial component used on GLUE, we set

$$\mathbf{a}_{m,1} \sim \mathcal{N}(\mathbf{0}, \sigma_{\text{init}}^2 \mathbf{I}), \qquad \mathbf{b}_{m,1} = \mathbf{0}, \qquad e_{m,1} = e_0, \tag{15}$$

where $\sigma_{\text{init}} > 0$ is the input-factor initialization standard deviation, $\mathbf{I}$ is the $p_m \times p_m$ identity matrix, and $e_0 \neq 0$ is the initial scalar coefficient. At initialization, only the output factor can have a nonzero task-loss gradient, $\partial\mathcal{L}/\partial\mathbf{b}_{m,1} = \alpha e_0 \mathbf{G}_m \mathbf{a}_{m,1}$. Newly inserted components use the same $e_0$ and the gradient-based input initialization in Eq. (7). Scalar coefficients remain trainable and may change sign.

## A.2 First-Order Rank-One Signal Derivation

Fix all other model parameters and consider the rank-one perturbation $\delta\Delta\mathbf{W}_m = -\eta\mathbf{b}\mathbf{a}^\top$, where $\eta > 0$ and $\mathbf{a} \in \mathbb{R}^{p_m}$ and $\mathbf{b} \in \mathbb{R}^{q_m}$ are unit-$\ell_2$ vectors. Since $\|\mathbf{b}\mathbf{a}^\top\|_F = 1$, every such perturbation has Frobenius length $\eta$. Using the gradient $\mathbf{G}_m$ defined in Eq. (3), the first-order expansion is

$$\begin{aligned} \mathcal{L}(\Delta\mathbf{W}_m + \delta\Delta\mathbf{W}_m) &\approx \mathcal{L}(\Delta\mathbf{W}_m) + \langle \mathbf{G}_m, \delta\Delta\mathbf{W}_m \rangle \\ &= \mathcal{L}(\Delta\mathbf{W}_m) - \eta\, \mathbf{b}^\top \mathbf{G}_m \mathbf{a}, \end{aligned} \tag{16}$$

where $\langle \mathbf{X}, \mathbf{Y} \rangle = \operatorname{tr}(\mathbf{X}^\top \mathbf{Y})$ is the Frobenius inner product. By the Cauchy–Schwarz inequality and the definition of the spectral norm,

$$\mathbf{b}^\top \mathbf{G}_m \mathbf{a} \leq \|\mathbf{b}\|_2 \|\mathbf{G}_m \mathbf{a}\|_2 \leq \|\mathbf{G}_m\|_2 = \sigma_1(\mathbf{G}_m). \tag{17}$$

Equality holds for a corresponding pair of unit leading left and right singular vectors $\mathbf{u}_{m,1}$ and $\mathbf{v}_{m,1}$: the relation $\mathbf{G}_m \mathbf{v}_{m,1} = \sigma_1(\mathbf{G}_m)\mathbf{u}_{m,1}$ gives $\mathbf{u}_{m,1}^\top \mathbf{G}_m \mathbf{v}_{m,1} = \sigma_1(\mathbf{G}_m)$. Hence $-\mathbf{u}_{m,1}\mathbf{v}_{m,1}^\top$ maximizes the first-order loss decrease per unit Frobenius step length, establishing Eq. (4). This maximization ranges over all rank-one perturbations of the complete update, independently of its current factorization.

## A.3 Historical-Peak State

At the $t$-th growth examination of module $m$, $\widehat{\sigma}_m^{(t)} \geq 0$ denotes the estimate of $\sigma_1(\mathbf{G}_m)$. Each module maintains a scalar historical peak $P_m$, initially unset. At the first examination, we set $P_m \leftarrow \widehat{\sigma}_m^{(1)}$ and add a component. At examination $t \geq 2$, the reference contains only earlier values,

$$P_m^{(t)} = \max_{1 \leq s < t} \widehat{\sigma}_m^{(s)}. \tag{18}$$

Here $s$ indexes earlier examinations, and $P_m^{(t)}$ is the value of the stored variable $P_m$ immediately before examination $t$. With the run's stopping ratio $0 < \tau < 1$, we compare the current estimate with $\tau P_m$ before updating the reference. If $\widehat{\sigma}_m^{(t)} < \tau P_m$, growth stops permanently. Otherwise, we set $P_m \leftarrow \max(P_m, \widehat{\sigma}_m^{(t)})$ and add a component. Insertion resets the sensitivity windows, their running maximum $S_{m,u}$, and the local update count; it retains the spectral peak $P_m$. Existing components remain trainable after growth stops.

The spectral identity in Eq. (4) concerns an exact score; this decision uses its numerical estimate. The strict inequality cannot stop a nonnegative estimate when $P_m = 0$. The maximum waiting interval can therefore trigger further insertions even if all scores remain zero. The stated procedure has no additional zero-signal or noise-floor stopping test. The equations assume finite scores and a finite unit insertion vector; numerical failures are not growth evidence.

### A.4 Sensitivity-Based Examination

The examination schedule uses the coefficient sensitivity $\zeta_{m,u}$ in Eq. (6), which measures the magnitude of the first-order loss response per unit relative change in the newest coefficient. Here $u$ counts local optimizer updates since initialization or the latest insertion, and $\zeta_{m,u}$ is computed from the gradient of update $u$ before its optimizer step. Before the first insertion, the monitored coefficient is that of the last-indexed initial component $r_{m,0}$.

Let $w_{\text{sens}}$ be a positive window length, and let $u_{\min} \le u_{\max}$ be the minimum and maximum waiting intervals, with $2w_{\text{sens}} \le u_{\min}$ so that two full windows are available at the first eligible update. For $u \ge w_{\text{sens}}$, let $\bar{\zeta}^{\text{recent}}_{m,u}$ be the mean of the latest $w_{\text{sens}}$ sensitivities. For $u \ge 2w_{\text{sens}}$, let $\bar{\zeta}^{\text{old}}_{m,u}$ be the mean of the $w_{\text{sens}}$ sensitivities immediately preceding that window. The stability and activity tests are evaluated only for $u \ge u_{\min}$:

$$\frac{|\bar{\zeta}^{\text{recent}}_{m,u} - \bar{\zeta}^{\text{old}}_{m,u}|}{\max(\bar{\zeta}^{\text{recent}}_{m,u}, \epsilon)} < \delta_{\text{stab}}, \qquad \bar{\zeta}^{\text{recent}}_{m,u} > \kappa_{\text{act}} S_{m,u}, \tag{19}$$

where $\delta_{\text{stab}} > 0$ is the relative stability threshold, $\epsilon = 10^{-12}$ prevents division by zero, $0 \le \kappa_{\text{act}} < 1$ is the activity-floor ratio, and $S_{m,u}$ is the running maximum of recent-window means from the first full window after initialization or the latest insertion through update $u$, including means computed before $u_{\min}$. A module is examined when $u \ge u_{\min}$ and both tests pass, or when $u = u_{\max}$ regardless of the tests; the maximum waiting interval thus guarantees an examination. The historical-peak comparison in Appendix A.3 then decides whether to add a component.

### A.5 Function-Preserving Insertion and Optimizer State

The insertion in Eq. (7) uses an input-factor scale $c_a > 0$ and coefficient $e_0 \ne 0$, both fixed within a run. We register the new factor vectors and coefficient as separate parameter objects with fresh optimizer state. Their parameter groups inherit the reference adapter group's hyperparameters and initial learning rate, but each new group starts its own schedule.

Let $b$ be the nonnegative integer scheduler counter at registration, $T$ the total training horizon in scheduler steps, and $w \ge 0$ the original warmup length in scheduler steps. A group registered at $b$ has remaining horizon $H_b = T - b$. For a nonnegative integer local scheduler age $a$, its learning rate is

$$\eta_b(a) = \eta_{\text{base}} \begin{cases} a/\max(1, w), & a < w, \\ \max\{0, (H_b - a)/\max(1, H_b - w)\}, & a \ge w. \end{cases} \tag{20}$$

Here $\eta_b(a)$ is the group's learning rate at local age $a$, and $\eta_{\text{base}}$ is the inherited initial learning rate. The scalars $a$ and $b$ in this schedule are counters. Thus a new group restarts warmup for the original length $w$ and decays over its remaining horizon, while existing groups retain their schedules. Its local counter starts at $-1$ and reaches zero at the first scheduler step after registration.

Insertion occurs after gradient computation (Appendix A.6), so new parameters have no gradient on the insertion step. With positive warmup, their first subsequent optimizer update uses the age-zero learning rate, which is zero. If $H_b < w$, the group remains in warmup until training ends; the warmup interval is not shortened to fit the remaining horizon.

### A.6 Overall Adaptive Rank Procedure

Algorithm 1 combines examination, stopping, insertion, and optimization. A module is eligible for examination only while its growth is active and its schedule in Appendix A.4 triggers a check. The algorithm uses $\widehat{\sigma}_m$ for the current spectral estimate and $P_m$ for the stored historical peak; the examination superscript is suppressed only in this algorithm.

**Algorithm 1** Adaptive Rank

**Require:** Pretrained model, training batches, initial counts $\{r_{m,0}\}$, stopping ratio $0 < \tau < 1$, spectral-estimation procedure, examination and insertion settings from Appendices A.4 and A.5
**Ensure:** Trained adapters with module-wise component counts

```
Initialize r_{m,0} components in each module m and set r_m ← r_{m,0}; activate growth, reset the local examination statistics, and leave historical peaks unset
for each effective training batch do
    Compute task-loss gradients and update sensitivity statistics
    for each module m eligible for examination do
        Estimate σ̂_m and the unit right singular vector v̂_{m,1} from G_m
        if P_m is set and σ̂_m < τ P_m then
            Permanently stop growth of module m
        else
            Set P_m ← σ̂_m if unset; otherwise set P_m ← max(P_m, σ̂_m)
            Append a component using Eq. (7)
            Set r_m ← r_m + 1 and reset the module's examination statistics
        end if
    end for
    Take an optimizer step and advance the parameter-group learning-rate schedules
end for
return The trained model and its component counts at the training horizon
```

# B Adaptive Sparsity Details

This appendix supplements Section 3.2 with input-moment construction, coordinate-solver details, and precise candidate-selection rules. We retain the notation of the main text. Algorithm 2 gives the complete procedure.

## B.1 Data and Reconstruction Moments

We use the fixed training-data partition described in Section 3.2, shared across all modules and tolerances. For GLUE, $\mathcal{A}$ contains approximately $0.25\%$–$5\%$ of the training examples, depending on the task; the remaining examples are partitioned approximately $3:1$ between $\mathcal{T}_{\text{fit}}$ and $\mathcal{T}_{\text{sel}}$.

We collect module inputs with forward pre-hooks while running $\mathcal{T}_{\text{fit}}$ and $\mathcal{T}_{\text{sel}}$ through the fixed source model $\Theta_{\text{source}}$. Each non-padding token identified by the attention mask contributes one input vector. Let $N_{\text{fit},m} > 0$ and $N_{\text{sel},m} > 0$ count the vectors collected at module $m$, denoted by $\mathbf{h}_{m,i}^{(\text{fit})}$ and $\mathbf{h}_{m,i}^{(\text{sel})} \in \mathbb{R}^{p_m}$, respectively. The empirical uncentered second moments are

$$\mathbf{H}_m^{\text{fit}} = \frac{1}{N_{\text{fit},m}} \sum_{i=1}^{N_{\text{fit},m}} \mathbf{h}_{m,i}^{(\text{fit})} (\mathbf{h}_{m,i}^{(\text{fit})})^\top \tag{21}$$

and

$$\mathbf{H}_m^{\text{sel}} = \frac{1}{N_{\text{sel},m}} \sum_{i=1}^{N_{\text{sel},m}} \mathbf{h}_{m,i}^{(\text{sel})} (\mathbf{h}_{m,i}^{(\text{sel})})^\top. \tag{22}$$

Both moments are symmetrized and held fixed throughout refitting and model assembly.

## B.2 Coordinate-Stagewise Sparse Path Solver

For module $m$ and residual step $k$, the solver fits one sparse rank-one term to the fixed residual $\mathbf{R} = \mathbf{R}_{m,k-1}$ using $\mathbf{H} = \mathbf{H}_m^{\text{fit}}$. We write $p = p_m$, $q = q_m$ and suppress $(m,k)$ on $\widetilde{e}$, $\lambda$, $\widetilde{\mathbf{a}} \in \mathbb{R}^p$, and $\widetilde{\mathbf{b}} \in \mathbb{R}^q$, with $\|\widetilde{\mathbf{a}}\|_1 = \|\widetilde{\mathbf{b}}\|_1 = 1$.

**Normalized objective.** Define

$$\mathbf{C}^{\text{raw}} = \mathbf{H}\mathbf{R}^\top, \qquad \gamma = \max_{i,j} |C_{ij}^{\text{raw}}|. \tag{23}$$

If $\gamma = 0$, return no positive-amplitude state. Otherwise, set

$$\mathbf{C} = \mathbf{C}^{\mathrm{raw}}/\gamma, \qquad \widetilde{e} = \gamma\bar{e}, \qquad \lambda = \gamma\bar{\lambda}. \tag{24}$$

Removing the residual-only constant from Eq. (10) and dividing by $\gamma^2$ gives $\Gamma = f + \bar{\lambda}\bar{e}$, where

$$f(\bar{e}, \widetilde{\mathbf{a}}, \widetilde{\mathbf{b}}) = -\bar{e}\widetilde{\mathbf{a}}^\top \mathbf{C}\widetilde{\mathbf{b}} + \frac{\bar{e}^2}{2}(\widetilde{\mathbf{a}}^\top \mathbf{H}\widetilde{\mathbf{a}})\|\widetilde{\mathbf{b}}\|_2^2. \tag{25}$$

**Initialization.** For coordinate step $\varepsilon > 0$, evaluate each input/output coordinate pair $(i, j)$:

$$\begin{aligned} t_{ij} &= \min\left\{\varepsilon, \frac{|C_{ij}|}{\max(H_{ii}, 10^{-30})}\right\}, \\ d_{ij} &= t_{ij}|C_{ij}| - \frac{1}{2}t_{ij}^2 \max(H_{ii}, 10^{-30}). \end{aligned} \tag{26}$$

Choose the pair maximizing $d_{ij}$, breaking exact ties by flattened coordinate order, and initialize

$$\widetilde{\mathbf{a}}^{\mathrm{init}} = \mathbf{e}_i^{(p)}, \qquad \widetilde{\mathbf{b}}^{\mathrm{init}} = \mathrm{sign}(C_{ij})\mathbf{e}_j^{(q)}, \qquad \bar{e}^{\mathrm{init}} = t_{ij}, \tag{27}$$

where $\mathbf{e}_i^{(p)}$ and $\mathbf{e}_j^{(q)}$ are standard basis vectors. Set

$$\bar{\lambda}^{\mathrm{init}} = -\frac{f(\bar{e}^{\mathrm{init}}, \widetilde{\mathbf{a}}^{\mathrm{init}}, \widetilde{\mathbf{b}}^{\mathrm{init}})}{\bar{e}^{\mathrm{init}}}, \tag{28}$$

so that the initial penalized objective equals zero.

**Coordinate changes.** Use scaled factors $\mathbf{w}^a = \bar{e}\widetilde{\mathbf{a}}$ and $\mathbf{w}^b = \bar{e}\widetilde{\mathbf{b}}$. With the opposite normalized factor fixed, the penalty is $\bar{\lambda}\|\mathbf{w}^\nu\|_1$ on either side $\nu \in \{a, b\}$. The reconstruction gradients and coordinate curvatures are

$$\begin{aligned} \mathbf{g}^a &= -\mathbf{C}\widetilde{\mathbf{b}} + \|\widetilde{\mathbf{b}}\|_2^2 \mathbf{H}\mathbf{w}^a, \qquad c_i^a = \|\widetilde{\mathbf{b}}\|_2^2 H_{ii}, \\ \mathbf{g}^b &= -\mathbf{C}^\top\widetilde{\mathbf{a}} + (\widetilde{\mathbf{a}}^\top \mathbf{H}\widetilde{\mathbf{a}})\mathbf{w}^b, \quad c_i^b = \widetilde{\mathbf{a}}^\top \mathbf{H}\widetilde{\mathbf{a}}. \end{aligned} \tag{29}$$

Here $i$ indexes a coordinate on the selected side. A scalar increment $\delta$ has exact reconstruction-loss change $\Delta f_i^\nu(\delta)$ and an additional acceptance margin $\xi_i^\nu(\delta)$:

$$\begin{aligned} \Delta f_i^\nu(\delta) &= \delta g_i^\nu + \frac{1}{2}\delta^2 c_i^\nu, \\ \xi^{\mathrm{floor}} &= 10^{-12}\max\left\{\max_{i,j}|C_{ij}|, 10^{-30}\right\}, \\ \xi_i^\nu(\delta) &= \max\left\{\frac{1}{2}\delta^2 c_i^\nu, \xi^{\mathrm{floor}}\right\}. \end{aligned} \tag{30}$$

**Backward moves.** Each iteration first considers shrinking active coordinates:

$$\delta = -\,\mathrm{sign}(w_i^\nu)\min(\varepsilon, |w_i^\nu|). \tag{31}$$

Within each factor, choose the coordinate minimizing the left-hand side of

$$\Delta f_i^\nu(\delta) - \bar{\lambda}|\delta| + \xi_i^\nu(\delta) \le 0, \tag{32}$$

breaking exact ties by coordinate order. Try the input-factor proposal first and the output-factor proposal only if the first is not accepted. Accept a proposal when Eq. (32) holds and $\bar{e} - |\delta| > 10^{-15}$. The move decreases $\bar{e}$ by $|\delta|$ and leaves $\bar{\lambda}$ unchanged.

**Forward moves.** If neither backward proposal is accepted, consider $\delta = -\varepsilon\,\mathrm{sign}(g_i^\nu)$ for coordinates with nonzero gradient. A proposal is legal when $w_i^\nu = 0$ or $\mathrm{sign}(\delta) = \mathrm{sign}(w_i^\nu)$. Among legal proposals, maximize

$$n_i^\nu = -\Delta f_i^\nu(\delta) - \xi_i^\nu(\delta). \tag{33}$$

Exact ties favor side $a$, then the smallest coordinate index. For the selected proposal, set

$$\bar{\lambda}^{\mathrm{trial}} = \frac{n_i^\nu}{\varepsilon}, \qquad \bar{\lambda}^{\mathrm{next}} = \min(\bar{\lambda}, \bar{\lambda}^{\mathrm{trial}}). \tag{34}$$

Stop if no legal proposal exists or if the trial price is at or below the configured nonnegative threshold. Otherwise, apply the move, increase $\bar{e}$ by $\varepsilon$, and set $\bar{\lambda} = \bar{\lambda}^{\mathrm{next}}$. Thus the regularization price is nonincreasing along the path.

**Renormalization.** After an accepted input-factor move, update

$$\begin{aligned}(\mathbf{w}^a)' &= \mathbf{w}^a + \delta \mathbf{e}_i^{(p)}, \quad \bar{e}' = \|(\mathbf{w}^a)'\|_1, \\ \widetilde{\mathbf{a}}' &= (\mathbf{w}^a)'/\bar{e}', \qquad \widetilde{\mathbf{b}}' = \widetilde{\mathbf{b}}, \\ (\mathbf{w}^b)' &= \bar{e}'\widetilde{\mathbf{b}}'.\end{aligned} \tag{35}$$

For an output-factor move, update $\mathbf{w}^b$ analogously using $\mathbf{e}_i^{(q)}$, normalize $\widetilde{\mathbf{b}}$, retain $\widetilde{\mathbf{a}}$, and synchronize $(\mathbf{w}^a)' = \bar{e}'\widetilde{\mathbf{a}}$. This restores the factor constraints without further changing the updated rank-one term.

**Controls and returned states.** The coordinate step, price threshold, iteration budget, probe interval, and cache/check intervals are fixed before fitting. The path terminates under the stopping rules above or when the iteration budget is exhausted. Positive-amplitude states recorded at the probe interval and the final endpoint form the sampled collection; initialization is not a separate probe. Return amplitudes at their original scale, $\widetilde{e} = \gamma\bar{e}$; Appendix B.3 specifies the numerical verification used during selection.

### B.3 Shared Continuation and Tolerance-Specific Candidates

All comparisons use $\mathbf{H}_m = \mathbf{H}_m^{\mathrm{sel}}$ and the fixed denominator $Z_m$ from Eq. (11), with $\epsilon_{\mathrm{den}} = 10^{-12}$.

**Sampled versions.** At residual step $k$ in module $m$, the solver in Appendix B.2 records versions of the new term $\mathbf{T}_{m,k}$. Let $\mathcal{T}_{m,k}$ be this collection, with $\ell$ indexing the versions in evaluation order:

$$\mathbf{T}_{m,k}^{(\ell)} = \widetilde{e}_{m,k}^{(\ell)} \widetilde{\mathbf{b}}_{m,k}^{(\ell)} (\widetilde{\mathbf{a}}_{m,k}^{(\ell)})^\top. \tag{36}$$

Each stored version retains its unit-$\ell_1$ factors, original-scale amplitude $\widetilde{e}_{m,k}^{(\ell)} > 0$, and index. Initialization appears only as the final endpoint when no move is made.

**Continuation selection and admission.** For a nonempty collection, choose

$$\mathbf{T}_{m,k}^{\mathrm{cont}} \in \operatorname*{arg\,min}_{\mathbf{T}_{m,k}^{(\ell)} \in \mathcal{T}_{m,k}} \|\mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{(\ell)}\|_{\mathbf{H}_m}^2, \tag{37}$$

breaking exact ties by the smaller index $\ell$. Compute $g_{m,k}$ using Eq. (12). If $g_{m,k} \geq g_{\min}$, admit the term and set $\mathbf{R}_{m,k} = \mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{\mathrm{cont}}$, so that $\widetilde{\Delta\mathbf{W}}_{m,k} = \Delta\mathbf{W}_m - \mathbf{R}_{m,k}$. Let $L_m$ denote the number of terms admitted under the stopping rules in Section 3.2.

**Shortest feasible prefix.** The tolerance set $\mathcal{R} = \{\rho_1, \ldots, \rho_J\}$, with $0 \leq \rho_1 < \cdots < \rho_J$, is fixed before refitting. For $\rho \in \mathcal{R}$, select

$$\tilde{r}_m(\rho) = \min\left\{ k \in \{1, \ldots, L_m\} \;\middle|\; \frac{\|\mathbf{R}_{m,k}\|_{\mathbf{H}_m}^2}{Z_m} \leq \rho \right\}. \tag{38}$$

If this set is empty, no candidate is produced for that tolerance.

**Terminal selection.** For $k = \tilde{r}_m(\rho)$, keep the first $k-1$ continuation terms fixed. The feasible terminal versions are

$$\begin{aligned}\mathcal{F}_{m,k,\rho} = \Bigg\{ \mathbf{T}_{m,k}^{(\ell)} \in \mathcal{T}_{m,k} \;\Bigg|\; & \frac{\|\mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{(\ell)}\|_{\mathbf{H}_m}^2}{Z_m} \leq \rho, \\ & \frac{\|\mathbf{R}_{m,k-1}\|_{\mathbf{H}_m}^2 - \|\mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{(\ell)}\|_{\mathbf{H}_m}^2}{Z_m} \geq g_{\min} \Bigg\}.\end{aligned} \tag{39}$$

The admitted continuation term is feasible, so this collection is nonempty whenever the prefix qualifies. Choose the terminal version by lexicographically minimizing

$$\begin{aligned}\mathbf{T}_m^{\mathrm{term}}(\rho) \in \operatorname*{arg\,min}_{\mathbf{T}_{m,k}^{(\ell)} \in \mathcal{F}_{m,k,\rho}} \Big( & \mathrm{nnz}(\widetilde{\mathbf{a}}_{m,k}^{(\ell)}) + \mathrm{nnz}(\widetilde{\mathbf{b}}_{m,k}^{(\ell)}), \\ & \|\mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{(\ell)}\|_{\mathbf{H}_m}^2, \; \ell \Big).\end{aligned} \tag{40}$$

**Algorithm 2** Adaptive Sparsity with shared continuation sequences

**Require:** Source model $\Theta_{\text{source}}$ and source updates $\{\Delta\mathbf{W}_m\}$; fixed moments $\mathbf{H}_m^{\text{fit}}$ and $\mathbf{H}_m^{\text{sel}}$; fixed task-metric evaluator $\mathbf{q}_{\mathcal{A}}$; source component counts $\{r_m\}$; ordered tolerances $\rho_1 < \cdots < \rho_J$; minimum gain $g_{\min} > 0$; fixed solver and sampling settings from Appendix B.2; fixed module assembly order
**Ensure:** Model $\Theta$ after cumulative acceptance checks

**for** each adapted module $m$ **do**
  *Build one shared continuation sequence.*
  Set $\mathbf{R}_{m,0} \leftarrow \Delta\mathbf{W}_m$, $L_m \leftarrow 0$, and $Z_m \leftarrow \|\Delta\mathbf{W}_m\|^2_{\mathbf{H}_m^{\text{sel}}} + 10^{-12}$
  **for** $k = 1, \ldots, r_m$ **do**
    Construct sampled alternatives $\mathcal{T}_{m,k}$ for one rank-one term using $\mathbf{R}_{m,k-1}$ and $\mathbf{H}_m^{\text{fit}}$
    If $\mathcal{T}_{m,k} = \varnothing$, **break**
    Reconcile cached errors with the full quadratic comparisons before selection (Appendix B.3)
    Select $\mathbf{T}_{m,k}^{\text{cont}}$ with minimum residual error under $\mathbf{H}_m^{\text{sel}}$ using Eq. (37)
    Compute its normalized gain $g_{m,k}$ using Eq. (12)
    If $g_{m,k} < g_{\min}$, discard the proposal and **break**
    Admit $\mathbf{T}_{m,k}^{\text{cont}}$; set $\mathbf{R}_{m,k} \leftarrow \mathbf{R}_{m,k-1} - \mathbf{T}_{m,k}^{\text{cont}}$ and $L_m \leftarrow k$
  **end for**
  *Reuse its prefixes and sampled terminal alternatives.*
  **for** $j = 1, \ldots, J$ **do**
    Find the smallest $k \in \{1, \ldots, L_m\}$ with $\|\mathbf{R}_{m,k}\|^2_{\mathbf{H}_m^{\text{sel}}}/Z_m \le \rho_j$
    If no such $k$ exists, skip this tolerance
    Set $\tilde{r}_m(\rho_j) \leftarrow k$ and keep the first $k-1$ continuation terms fixed
    Select $\mathbf{T}_m^{\text{term}}(\rho_j)$ with the fewest factor nonzeros among stored states in $\mathcal{T}_{m,k}$ meeting both the tolerance and gain tests (Eq. (40))
    Export $\widetilde{\Delta\mathbf{W}}_m^{(\rho_j)} \leftarrow \sum_{k'=1}^{k-1} \mathbf{T}_{m,k'}^{\text{cont}} + \mathbf{T}_m^{\text{term}}(\rho_j)$; verify feasibility with the full quadratic form
    If verification fails, exclude the candidate from model trials
  **end for**
**end for**
*Check replacements cumulatively in the full model.*
Set $\Theta \leftarrow \Theta_{\text{source}}$ and evaluate the fixed reference metrics $\mathbf{q}_{\mathcal{A}}(\Theta_{\text{source}})$
**for** each module $m$ in the fixed assembly order **do**
  **for** $j = 1, \ldots, J$ with a locally valid candidate for $\rho_j$ **do**
    Form $\Theta_{\text{cand}}$ by replacing module $m$ in the current $\Theta$ with $\widetilde{\Delta\mathbf{W}}_m^{(\rho_j)}$
    **if** $\mathbf{q}_{\mathcal{A}}(\Theta_{\text{cand}}) \succeq \mathbf{q}_{\mathcal{A}}(\Theta_{\text{source}})$ **then**
      Set $\Theta \leftarrow \Theta_{\text{cand}}$ and **break** the candidate loop
    **end if**
  **end for**
  If no candidate passes, retain the source representation of module $m$
**end for**
**return** $\Theta$

Here nnz counts nonzero factor entries. The three entries prioritize their total count, reconstruction error, and evaluation order, respectively. Export the candidate in Eq. (13) using these sampled factors and amplitudes without reoptimization.

**Numerical verification.** All selection rules use full quadratic errors. If cache verification changes a ranking or feasibility decision, recompute the affected comparisons and reapply the selection rule. Only candidates passing the final full-quadratic feasibility check enter the full-model trials.

**Effect of the gain threshold.** Any further gain is bounded by the remaining normalized residual error. Thus no additional term can be admitted once $\|\mathbf{R}_{m,k}\|^2_{\mathbf{H}_m}/Z_m < g_{\min}$. A tolerance below $g_{\min}$ must be reached by an admitted term without relaxing the gain threshold.

### B.4 Cumulative Model-Level Acceptance

Algorithm 2 implements the cumulative check in Eq. (14), using the fixed acceptance subset $\mathcal{A}$ from Appendix B.1. Candidate factors, amplitudes, and input moments remain fixed during assembly. The reference metrics $\mathbf{q}_{\mathcal{A}}(\Theta_{\text{source}})$ are evaluated once and reused for all trials.

# C Experimental Settings and Evaluation Protocols

The experiments use backbone-specific adaptation settings: DeBERTaV3-Base adapts attention and feed-forward projections, whereas Qwen3-4B-Base adapts query and value projections (Table 4). The reported ranges cover task-specific settings and configurations considered. We specify initialization and evaluation separately below because the two benchmarks use different training budgets and score-aggregation procedures. Both backbones are optimized with AdamW (Loshchilov & Hutter, 2019).

Table 4: Training settings for DeBERTaV3-Base on GLUE and Qwen3-4B-Base on Math-CoT-20k.

| Setting | DeBERTaV3-Base | Qwen3-4B-Base |
|---|---|---|
| Target projections | Query, key, value, attention output, and two feed-forward projections | Query and value |
| Adapted layers / modules | 12 / 72 | 36 / 72 |
| Optimizer | AdamW | AdamW |
| Effective batch size | 32 | 16 |
| Learning rate | $5 \times 10^{-4}$–$2.2 \times 10^{-3}$ | $5 \times 10^{-5}$–$10^{-4}$ |
| Training budget | 5–50 epochs | 2 epochs (2,560 updates) |
| Initial components per module | 1 | 8 or 16 |
| Growth threshold $\tau$ | 0.20–0.50 | 0.20–0.50 |
| Learning-rate schedule | Linear warmup and decay | Linear warmup and decay |
| Weight decay | 0.01 | 0 |

For Qwen3-4B-Base, we use 76 warmup updates, a maximum training length of 20,000 tokens, and zero adapter dropout.

## C.1 DeBERTaV3-Base on GLUE

We evaluate DeBERTaV3-Base on eight GLUE tasks, using the setups of AdaLoRA and IncreLoRA (Zhang et al., 2023b;a) to guide the task-specific settings in Table 4. We report the Matthews correlation coefficient for CoLA, the mean of Pearson and Spearman correlations for STS-B, and accuracy for the remaining tasks, using the matched split for MNLI. Published baseline results retain the reporting conventions of their original sources; scores using incompatible metrics are omitted. Avg. denotes the unweighted mean across all eight tasks and is reported only for complete rows.

All GLUE experiments conducted in this work are repeated with three random seeds, including the reconstruction comparisons in Section 4.3 and the cross-method adaptation experiments in Section 4.4. We report means over the three seeds.

## C.2 Qwen3-4B-Base on Math-CoT-20k

We fine-tune Qwen3-4B-Base on the 20,480 examples in Math-CoT-20k and evaluate on MATH500 and AIME2024 (Zhang & Math-AI Team, 2024), following the data format and evaluation protocol of Ren et al. (2026). We apply adapters using the settings in Table 4 while keeping the pretrained weights frozen.

**Initial capacity.** We initialize each adapted module with 8 or 16 components. Exploratory runs quickly grew beyond eight components per module. The larger initialization reduces this early growth overhead while leaving subsequent growth task-dependent. Related Qwen3 reasoning studies use adapter ranks of 8, 16, and 32: Gong et al. (2026) use LoRA rank 8 for Qwen3-4B, Ju et al. (2025) use QLoRA rank 16 for Qwen3-4B-Base, and Elrefai et al. (2025) use LoRA rank 32 for Qwen3-4B-Base in inheritance reasoning.

**Evaluation.** We follow the prompts, sampling settings, and answer-verification procedure of Ren et al. (2026). For each method, we repeat fine-tuning with three random seeds, yielding three independently trained models. We evaluate each model by sampling three responses per MATH500

problem and ten per AIME2024 problem, using temperature 0.6 and top-$p$ 0.95. For each model, correctness is averaged first across responses for each problem and then across problems, yielding one avg@3 score for MATH500 and one avg@10 score for AIME2024. We report the mean of these scores across the three training seeds.

# D Task-wise Capacity and Performance

Figure 1 compares task performance and adapter size at the available settings of LoRA, AdaLoRA, IncreLoRA, SoRA, and ElaLoRA, together with both stages of SPAR. For SPAR, the allocated factor count before sparse refitting ranges from 0.540M on RTE to 1.534M on QNLI, illustrating the different capacities returned across tasks. The paired points then show how much of each learned representation AS retains. On MRPC, for example, the allocated count decreases from 0.606M to 0.228M while the mean score changes from 91.91 to 91.99. The baseline points provide context at their reported operating points, with the parameter-count conventions specified below.

**Parameter counts.** For SPAR, parameter counts include all entries, including zeros, in the retained input and output factors across adapted modules. They exclude scalar amplitudes, the task head, and the frozen backbone. The annotation $\bar{r}$ denotes the average number of retained rank-one components per adapted module. Baseline parameter counts and rank annotations follow their source reports; IncreLoRA's parameter counts are averages across tasks (Zhang et al., 2023a).

# E Detailed Reconstruction Results

## E.1 Comparison of Reconstruction Methods

**Experimental setup.** We apply Post-Squeeze, PARA, ComPEFT, and Adaptive Sparsity to the same trained **SPAR w/o AS** adapters. For each source model, the methods reconstruct its learned updates while keeping the backbone and task head fixed. Task performance is evaluated on the same examples across methods, and reconstruction error is measured on the same cached source-model inputs. Task metrics follow Appendix C.1.

**Parameter counts and reconstruction error.** Allocated A/B parameters count all factor entries, including zeros; nonzero A/B parameters count only nonzero entries. Both counts include unchanged modules and exclude the backbone, task head, and separate scalar amplitudes. The total component count $r_{\text{total}}$ is the number of retained rank-one terms summed across all adapted modules.

For module $m$, let $\mathbf{X}_m \in \mathbb{R}^{p_m \times n_m}$ contain $n_m$ cached input vectors as columns, where $p_m$ is the module's input dimension. Let $\Delta\mathbf{W}_m$ and $\widetilde{\Delta\mathbf{W}}_m$ denote its source and reconstructed updates, respectively. We measure their relative update-action error as

$$e_X = \left( \frac{\sum_m n_m^{-1} \|(\widetilde{\Delta\mathbf{W}}_m - \Delta\mathbf{W}_m)\mathbf{X}_m\|_F^2}{\sum_m n_m^{-1} \|\Delta\mathbf{W}_m \mathbf{X}_m\|_F^2} \right)^{1/2}, \tag{41}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. The numerator aggregates mean squared output discrepancies across modules, and the denominator applies the same aggregation to the source updates' output energies. Taking the square root gives the reported relative error.

**Task-level comparison.** The aggregate comparison conceals substantial variation across tasks (Table 5). Post-Squeeze and PARA return smaller representations, but their score reductions are particularly pronounced on MNLI and QQP. The ordering by reconstruction error also differs from the ordering by task performance: on MRPC, both methods achieve lower $e_X$ than **SPAR**, yet lower task scores. Thus, $e_X$ characterizes fidelity to the source updates on cached inputs, while the paired task scores measure the downstream effect of reconstruction. Both are needed to interpret the reported compactness at these operating points.

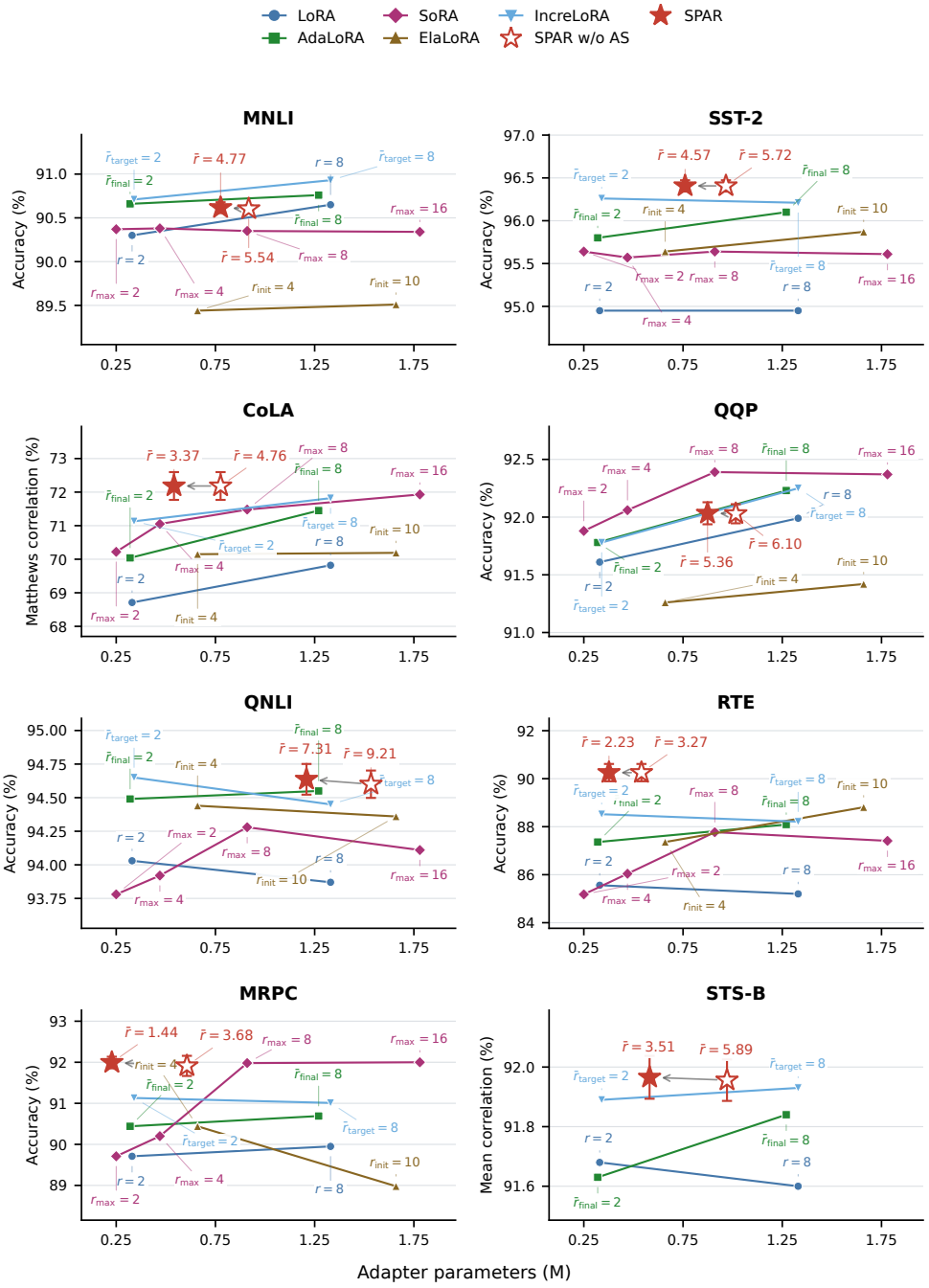


Figure 1: Task-wise GLUE performance versus adapter parameter count. Open red stars denote SPAR w/o AS and filled red stars denote SPAR; arrows connect their paired task means. Stars show means over three seeds, with vertical bars indicating sample standard deviations in score. Both stages use $\bar{r}$ for the mean stored component count per adapted module.

Table 5: Task-wise performance, adapter size, and reconstruction error for methods applied to the same SPAR w/o AS source adapters. A/B parameter counts are in millions. Bold marks the best value among reconstructed models for each task and metric. Dashes denote source models, for which no reconstruction error is reported.

| Method | MNLI | SST-2 | CoLA | QQP | QNLI | RTE | MRPC | STS-B |
|---|---|---|---|---|---|---|---|---|
| *Task score* ↑ | | | | | | | | |
| SPAR w/o AS | 90.60 | 96.41 | 72.18 | 92.03 | 94.60 | 90.25 | 91.91 | 91.96 |
| Post-Squeeze | 82.89 | 96.20 | 71.51 | 88.61 | 94.24 | 90.17 | 91.72 | **91.99** |
| PARA | 82.35 | 95.72 | 70.69 | 78.75 | 94.18 | 89.41 | 91.09 | 91.98 |
| ComPEFT | 88.19 | 95.60 | 68.70 | 90.19 | 93.37 | 87.12 | 90.77 | 90.93 |
| SPAR | **90.62** | **96.41** | **72.18** | **92.03** | **94.64** | **90.25** | **91.99** | 91.97 |
| *Allocated A/B parameters (M)* ↓ | | | | | | | | |
| SPAR w/o AS | 0.918 | 0.966 | 0.776 | 1.016 | 1.534 | 0.540 | 0.606 | 0.972 |
| Post-Squeeze | 0.375 | 0.379 | 0.375 | 0.380 | 0.385 | 0.342 | 0.359 | 0.385 |
| PARA | **0.186** | **0.201** | **0.173** | **0.225** | **0.319** | **0.124** | **0.141** | **0.208** |
| ComPEFT | 0.918 | 0.966 | 0.776 | 1.016 | 1.534 | 0.540 | 0.606 | 0.972 |
| SPAR | 0.776 | 0.730 | 0.541 | 0.873 | 1.157 | 0.377 | 0.228 | 0.581 |
| *Nonzero A/B parameters (M)* ↓ | | | | | | | | |
| SPAR w/o AS | 0.918 | 0.966 | 0.776 | 1.016 | 1.534 | 0.540 | 0.606 | 0.972 |
| Post-Squeeze | 0.375 | 0.379 | 0.375 | 0.380 | 0.385 | 0.342 | 0.359 | 0.385 |
| PARA | **0.186** | **0.201** | **0.173** | **0.225** | **0.319** | **0.124** | **0.141** | **0.208** |
| ComPEFT | 0.459 | 0.483 | 0.388 | 0.508 | 0.767 | 0.270 | 0.303 | 0.486 |
| SPAR | 0.739 | 0.664 | 0.494 | 0.842 | 1.114 | 0.356 | 0.145 | 0.528 |
| *Stored components* $r_{\text{total}}$ ↓ | | | | | | | | |
| SPAR w/o AS | 399.0 | 412.0 | 342.7 | 439.3 | 663.0 | 235.3 | 264.7 | 424.0 |
| Post-Squeeze | 162.9 | 164.0 | 163.6 | 164.7 | 166.9 | 148.7 | 155.7 | 166.7 |
| PARA | **99.3** | **102.3** | **85.3** | **109.3** | **165.3** | **58.3** | **65.7** | **105.7** |
| ComPEFT | 399.0 | 412.0 | 342.7 | 439.3 | 663.0 | 235.3 | 264.7 | 424.0 |
| SPAR | 343.7 | 319.0 | 242.3 | 385.7 | 510.7 | 160.3 | 104.0 | 253.0 |
| *Update-action error* $e_X$ ↓ | | | | | | | | |
| SPAR w/o AS | — | — | — | — | — | — | — | — |
| Post-Squeeze | 0.5419 | 0.3025 | 0.1148 | 0.5904 | 0.3568 | 0.0300 | **0.0450** | 0.0865 |
| PARA | 0.5525 | 0.2514 | 0.1000 | 0.5773 | 0.2289 | 0.0659 | 0.0809 | 0.0551 |
| ComPEFT | 0.9415 | 0.9002 | 0.7360 | 1.1017 | 0.9081 | 0.6808 | 0.7767 | 0.6785 |
| SPAR | **0.0339** | **0.0301** | **0.0186** | **0.0278** | **0.0401** | **0.0219** | 0.1332 | **0.0328** |

# F Structural Adaptation Across Adapter Methods

We evaluate the two stages separately to examine their applicability beyond the complete SPAR pipeline. On the eight GLUE tasks with DeBERTaV3-Base, we incorporate AR into LoRA and LoRA+ training and apply AS after AdaLoRA and IncreLoRA training. These comparisons extend Section 4.4 with the paired performance and structure measurements needed to assess each stage.

## F.1 Adaptive Rank in LoRA and LoRA+

We apply Adaptive Rank to LoRA (Hu et al., 2022) and LoRA+ (Hayou et al., 2024) and compare the resulting variants with their fixed-rank counterparts. The LoRA scores reproduce the published reference in Table 1; its allocated A/B count is calculated from the reported rank and adapted projections (Zhang et al., 2023b). LoRA + AR corresponds to **SPAR w/o AS**. Both LoRA+ variants are our implementations. Table 6 reports the task-wise scores and mean allocated A/B counts across the eight tasks.

**Task-dependent capacity.** With LoRA+, AR raises the average task score from 87.87 to 88.48 while reducing the mean allocated A/B count from 1.825M to 1.382M. The reported mean score increases on all seven classification tasks. For example, on QNLI, accuracy increases from 93.88 to 94.09, while allocated A/B parameters decrease from 2.654M to 2.177M. On STS-B, the allocated count decreases by 14.8%, from 1.327M to 1.130M, with a 0.04-point decrease in mean score. These results illustrate the performance and parameter-count trade-off obtained by applying AR with LoRA+'s training rule.

The LoRA comparison also illustrates that the returned capacity can exceed a fixed-rank reference: AR uses more parameters on QNLI and fewer on the other tasks (Appendix D).

Table 6: Task-wise GLUE scores with and without Adaptive Rank. Avg. and allocated A/B counts are unweighted means across the eight tasks. Bold marks the better value within each method pair, including ties.

| Method | MNLI Acc.↑ | SST-2 Acc.↑ | CoLA MCC↑ | QQP Acc.↑ | QNLI Acc.↑ | RTE Acc.↑ | MRPC Acc.↑ | STS-B Corr.↑ | Avg. ↑ | Allocated A/B (M) ↓ |
|---|---|---|---|---|---|---|---|---|---|---|
| LoRA | **90.65** | 94.95 | 69.82 | 91.99 | 93.87 | 85.20 | 89.95 | 91.60 | 88.50 | 1.327 |
| LoRA + AR | 90.60 | **96.41** | **72.18** | **92.03** | **94.60** | **90.25** | **91.91** | **91.96** | **89.99** | **0.916** |
| LoRA+ | 89.76 | 94.92 | 66.33 | 92.38 | 93.88 | 84.00 | 90.52 | **91.19** | 87.87 | 1.825 |
| LoRA+ + AR | **90.12** | **95.60** | **67.35** | **92.40** | **94.09** | **86.28** | **90.85** | 91.15 | **88.48** | **1.382** |

## F.2 Adaptive Sparsity on AdaLoRA and IncreLoRA

To assess whether Adaptive Sparsity also applies to updates learned by other methods, we refit our trained AdaLoRA (Zhang et al., 2023b) and IncreLoRA (Zhang et al., 2023a) adapters while keeping the backbone and task head fixed. These source adapters are distinct from the published baselines in Table 1. Table 7 reports the paired task scores, parameter counts, component counts, and reconstruction errors, together with their averages across tasks. The evaluation metrics and counting conventions follow Appendices C.1 and E.1.

For both source methods, AS reduces allocated and nonzero factor counts and stored component counts on every task, with no decrease in any reported per-task mean score. The mean nonzero count decreases from 0.353M to 0.295M for AdaLoRA and from 1.995M to 1.537M for IncreLoRA, while their mean scores change from 88.50 to 88.58 and from 88.39 to 88.51, respectively. The corresponding mean update-action errors are 0.0426 and 0.0328. These paired comparisons support applying AS to the learned updates of both adapter methods under the evaluated settings.

The amount of compression nevertheless varies across tasks. For IncreLoRA, nonzero factor counts decrease from 2.046M to 0.682M on RTE and from 2.178M to 2.085M on STS-B, reductions of 66.7% and 4.3%, respectively. The similar source sizes therefore lead to substantially different retained fractions, despite neither task showing a decrease in its reported mean score.

Table 7: Task-wise performance, adapter size, and reconstruction error before and after Adaptive Sparsity for our trained AdaLoRA and IncreLoRA adapters. Avg. is the unweighted mean across the eight tasks. A/B parameter counts are in millions. Task metrics follow Appendix C.1; structural metrics and $e_X$ follow Appendix E.1. Bold marks better scores and smaller counts within each source/refit pair, including ties. Reconstruction errors are reported only for refitted models.

| Method | MNLI | SST-2 | CoLA | QQP | QNLI | RTE | MRPC | STS-B | Avg. |
|---|---|---|---|---|---|---|---|---|---|
| *Task score* ↑ | | | | | | | | | |
| AdaLoRA | 90.29 | **95.41** | 68.33 | 91.49 | **94.33** | 86.64 | 89.95 | 91.54 | 88.50 |
| AdaLoRA + AS | **90.31** | **95.41** | **68.44** | **91.51** | **94.33** | **86.88** | **90.20** | **91.56** | **88.58** |
| IncreLoRA | 90.31 | **95.30** | 68.28 | **91.84** | 94.27 | 86.28 | **89.46** | 91.38 | 88.39 |
| IncreLoRA + AS | **90.35** | **95.30** | **68.46** | **91.84** | **94.31** | **87.00** | **89.46** | **91.39** | **88.51** |
| *Allocated A/B parameters (M)* ↓ | | | | | | | | | |
| AdaLoRA | 0.498 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.353 |
| AdaLoRA + AS | **0.442** | **0.290** | **0.276** | **0.302** | **0.313** | **0.275** | **0.285** | **0.292** | **0.309** |
| IncreLoRA | 1.509 | 2.046 | 2.083 | 1.935 | 2.074 | 2.046 | 2.092 | 2.178 | 1.995 |
| IncreLoRA + AS | **1.382** | **1.714** | **1.284** | **1.809** | **1.722** | **0.760** | **1.796** | **2.093** | **1.570** |
| *Nonzero A/B parameters (M)* ↓ | | | | | | | | | |
| AdaLoRA | 0.498 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.332 | 0.353 |
| AdaLoRA + AS | **0.438** | **0.277** | **0.249** | **0.298** | **0.309** | **0.259** | **0.252** | **0.279** | **0.295** |
| IncreLoRA | 1.509 | 2.046 | 2.083 | 1.935 | 2.074 | 2.046 | 2.092 | 2.178 | 1.995 |
| IncreLoRA + AS | **1.365** | **1.692** | **1.194** | **1.802** | **1.702** | **0.682** | **1.779** | **2.085** | **1.537** |
| *Stored components* $r_{\text{total}}$ ↓ | | | | | | | | | |
| AdaLoRA | 216.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 144.0 | 153.0 |
| AdaLoRA + AS | **183.0** | **121.3** | **117.0** | **126.0** | **135.0** | **114.3** | **120.0** | **125.3** | **130.3** |
| IncreLoRA | 648.0 | 864.0 | 864.0 | 864.0 | 888.0 | 864.0 | 864.0 | 872.0 | 841.0 |
| IncreLoRA + AS | **594.7** | **706.7** | **513.7** | **806.0** | **720.3** | **329.7** | **727.0** | **847.3** | **655.7** |
| *Update-action error* $e_X$ ↓ | | | | | | | | | |
| AdaLoRA | — | — | — | — | — | — | — | — | — |
| AdaLoRA + AS | 0.0141 | 0.0477 | 0.0632 | 0.0186 | 0.0187 | 0.0473 | 0.0850 | 0.0460 | 0.0426 |
| IncreLoRA | — | — | — | — | — | — | — | — | — |
| IncreLoRA + AS | 0.0300 | 0.0292 | 0.0490 | 0.0107 | 0.0288 | 0.0955 | 0.0148 | 0.0046 | 0.0328 |

## G Component Counts

Adaptive Rank produces different allocations across tasks, layers, and projection types. We examine the final allocations of DeBERTaV3-Base on the eight GLUE tasks to identify where capacity is added.

**Final component allocation.** Figure 2 shows the mean component count for each layer and projection type at the end of training. Allocations are nonuniform within tasks. For example, individual RTE modules finish with 2–7 components, while MNLI modules finish with 2–13 components. The projection-level pattern also varies: attention-output projections average 5.64 components on MNLI, compared with 3.83 for query projections, whereas the corresponding RTE means are 2.67 and 2.86. The reversal between these projection types illustrates that allocation depends on both the task and the module.

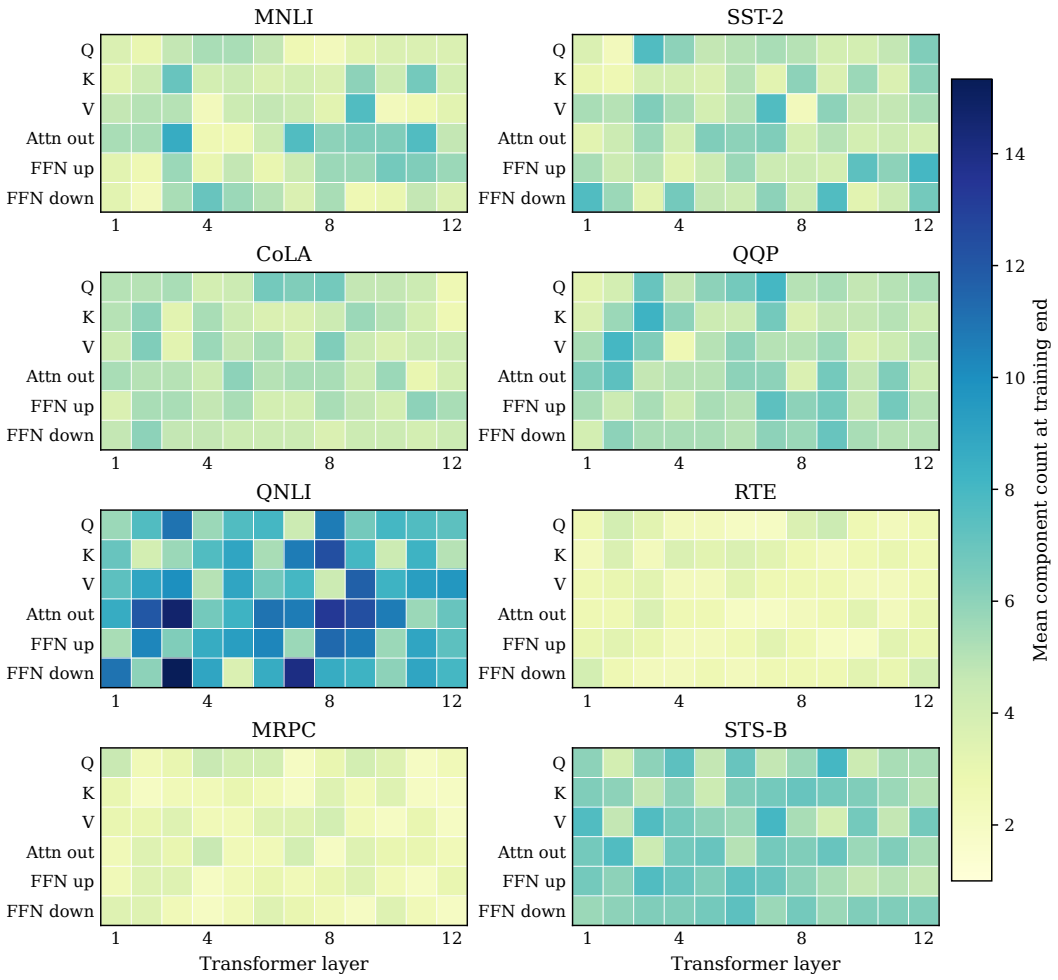


Figure 2: Mean component allocations at the end of GLUE training. Rows denote projection types and columns denote Transformer layers. All panels share the same color scale.

## H Census of Adaptive-Rank and Sparse LoRA Methods

Tables 8–10 compare how the methods in Section 2, together with neighboring approaches, determine component counts and coordinate sparsity. The "Amount determination" column describes how each structural quantity is fixed or selected. "Specified controls" lists relevant structural settings

and computational limits. Data-dependent outcomes remain conditional on these controls: selecting hyperparameters across runs and determining structure within a run are distinct decisions. For comparison, we report the corresponding characteristics of SPAR at the end of the relevant tables.

Table 8: Census I: Component-level methods that allocate, remove, replace, or grow low-rank terms.

| Method | Timing | Structural outcome | Specified controls | Amount determination |
|---|---|---|---|---|
| AdaLoRA (Zhang et al., 2023b) | During training | Allocation and survival of singular triplets across modules | Initial and target total-rank budgets; budget schedule, warm-up/final phases, and pruning interval | Prescribed total |
| IncreLoRA (Zhang et al., 2023a) | During training | Which important modules receive successive rank-one additions | Final total rank, number of modules expanded per event, allocation interval, and growth horizon | Prescribed total |
| ElaLoRA (Chang et al., 2025) | During training | Module-wise rank pattern produced by paired pruning and expansion | Constant global rank pool, initial rank, schedule, transfer count, and phase boundaries | Prescribed total |
| BeamLoRA (Gu et al., 2025) | During training | Replacement of low-score slots by expansions of high-score slots | Per-module rank, initialization proportion, replacement interval, and training horizon | Fixed per module |
| PrunedLoRA (Yu et al., 2025) | During training | Components retained after structured rank pruning | Oversized initial rank, target rank, pruning interval, and removal count | Prescribed target |
| SoRA (Ding et al., 2023) | During training | Active rank-one components selected by proximal gate sparsification | Maximum rank, sparsity regularization, proximal schedule, and training horizon | Path-dependent within cap |
| LAARA (Tripathi et al., 2026) | During training | Module ranks mapped from normalized Fisher-based importance | Minimum and maximum ranks, update interval, score-compression/blending controls, and vote count | Path-dependent within range |
| BaRA (Duan et al., 2026) | During training / inference | Global and input-dependent active subsets of latent rank factors | Maximum rank, sparse-prior parameters, KL weights, support threshold, and training horizon | Context-dependent within cap |
| BoostLoRA (Anantha et al., 2026) | Across rounds | Cumulative merged low-rank updates and failure-based early termination | Rank per round, maximum rounds, failure-count stopping threshold, and basis policy | Failure-stopped or round-limited |
| XGBLoRA (Zhang et al., 2024) | Across rounds | Accumulated low-rank updates merged into sampled layers | Rank per booster, number of sampled layers, total training steps, and merge interval | Horizon-set booster count |
| AROMA (Sheng et al., 2025) | During training | Module-wise count of retained sequential rank-one updates | Inner and outer tolerances, update scale, inner-step limit, check interval, and total training horizon | Signal-stopped or horizon-limited |
| InRank / InRank-Efficient (Zhao et al., 2023) | During training | Rank of cumulative low-rank weight updates | Initial and buffer ranks, explained-ratio threshold, check interval, and training horizon; rank-determination horizon for the efficient variant | Signal-adapted; then fixed in efficient variant |
| SPAR: Adaptive Rank (ours) | During training | Module-wise trained component counts $r_m$ | Initial component counts, relative gradient-signal threshold, sensitivity-based examination schedule, and training horizon | Signal-stopped growth or training horizon |

Table 9: Census II: Search-based and pre-fine-tuning probe methods. Search can select a data-dependent configuration, while the user still defines the candidate space and evaluation budget.

| Method | Timing | Structural outcome | Specified controls | Amount determination |
|---|---|---|---|---|
| HPO (Tribes et al., 2023) | Before final run | Global LoRA rank jointly selected with tuning hyperparameters | Rank bounds, hyperparameter search space, trial budget, and validation objective | Search-selected |
| LangBO (Baek et al., 2026) | Before final run | Global LoRA rank jointly selected with other hyperparameters | Candidate space, Bayesian-optimization budget, and validation objective | Search-selected |
| AutoPEFT (Zhou et al., 2024) | Before final run | PEFT operators, dimensions, and layer placements on a performance–size frontier | Architecture space, evaluation budget, objectives, and user-selected operating point | Search-selected |
| BIPEFT (Chang et al., 2024) | During search | Hybrid PEFT choices and layer-wise dimensions | Target parameter budget, candidate dimensions, search triggers, and training horizon | Budget-constrained |
| LoNAS (Munoz et al., 2024) | Before final run | Joint base-model subnetwork and elastic-LoRA configuration | Elastic supernetwork, maximum candidate ranks, and resource/performance objective | Search-selected |
| MARS (Cho et al., 2026) | Before final run | Modality-specific rank configuration | Candidate rank pairs, calibration anchors, and selection objective | Search-selected |
| GeLoRA (Ed-dib et al., 2025) | Before training | Module ranks derived from estimated intrinsic-dimension changes | Dimension estimator, rank floor, and rule sharing ranks within a block | Probe-returned |
| GoRA (He et al., 2025) | Before training | Module-wise ranks and initialization from weight/gradient importance | Reference rank inducing a total budget, rank bounds, and number of probe batches | Prescribed total |
| EVA (Paischer et al., 2025) | Before training | Module-wise ranks and input-factor initialization from activation spectra | Total rank budget, per-matrix maximum rank, and probe convergence threshold | Prescribed total |
| RSRA (Liu et al., 2026) | Before training | Module ranks from activation spectra and virtual-update displacement | Average rank, rank bounds, probe rank, threshold, and calibration sample size | Prescribed average |

Table 10: Census III: Within-component sparsity and neighboring element-wise sparse adaptation methods. The neighboring methods isolate sparsity amount and coordinate selection in base-weight updates rather than LoRA factors. Adaptive Sparsity can change both component counts and within-factor supports when refitting a learned adapter.

| Method | Timing | Structural outcome | Specified controls | Amount determination |
|---|---|---|---|---|
| RoseLoRA (Wang et al., 2024) | During training | Row- and column-wise coordinate patterns in LoRA factors | LoRA rank, final surviving fraction, cubic annealing schedule, and sensitivity smoothing | Prescribed ratio |
| GRASP LoRA (Hassan & Chen, 2026) | Online search; then fixed-mask tuning | Factor entries retained under one learned global pruning ratio | Feasible ratio interval, LoRA rank, controller horizon, and policy-training controls | Learned ratio, then fixed |
| SHiRA (Bhardwaj et al., 2024) | Before training | Sparse updates to selected base-weight entries | Mask-construction rule and prescribed trainable density | Prescribed density |
| Random Masking (Xu & Zhang, 2024) | Before training | Uniformly sampled trainable base-weight entries | Bernoulli selection probability or matched parameter count | Prescribed density |
| SpaRTA (Rios et al., 2025) | Before training | Randomly selected trainable base-weight entries | Prescribed number or density of trainable parameters | Prescribed density |
| SPAR: Adaptive Sparsity (ours) | After training | Refit component counts $\tilde{r}_m$ and factor-coordinate supports, or retained source representations | Sparse-path settings, reconstruction tolerances, minimum gain, training-data calibration, fixed module order, and no-drop task-quality criterion | Path-selected or source retained |