LoRA IN THE LAST LAYERS: SCORE-CONSTRAINED PLACEMENT SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-rank adaptation (LoRA) enables parameter-efficient fine-tuning of pretrained models, but parameter efficiency does not imply training efficiency. Adaptive methods select modules, parameters, or ranks, often changing adaptation capacity together with layer placement. Yet placement determines how far gradients propagate through the frozen backbone: similar parameter budgets can require materially different computation. We analyze this placement-dependent backward cost and introduce LaLoS, a score-constrained search over LoRA placements close to the output. A briefly trained probe and a held-out selection score determine the adapter placement for each task, without prescribing a fixed budget. Across language and vision benchmarks, placement close to the output reduces final fine-tuning computation and elapsed time at matched adapter capacity. LaLoS achieves competitive performance with task-dependent adapter counts. Fixed placements in the last layers remain strong baselines. Together, the analysis and experiments show that layer placement affects fine-tuning efficiency beyond trainable-parameter count.

1 INTRODUCTION

Low-rank adaptation (LoRA) is a parameter-efficient fine-tuning (PEFT) method that keeps pretrained weights frozen and inserts trainable low-rank matrices (Hu et al., 2022). A projection with this update is called a LoRA layer. With LoRA rank, projection dimensions, and the number of trainable parameters fixed, different adapter placement strategies can produce differences in training time and graphics processing unit (GPU) memory usage (Xu et al., 2026).

LoRA improvements. Recent work improves LoRA initialization (Paischer et al., 2025, Meng et al., 2024), rank allocation (Zhang et al., 2023b), and selection of target module types (Hayou et al., 2026). These directions make adaptation more selective, but they address different parts of the LoRA configuration.

The remaining LoRA adapter placement question is which layers receive low-rank updates. Existing methods use importance scores (Zhang et al., 2023b), sensitivity (He et al., 2023a), representation (Ogawa et al., 2026), or data-dependent signals (Hayou et al., 2026) to choose trainable components. These signals indicate which components may be useful for the task. Rank allocation, parameter selection, adapter selection, and selection of target module types can vary adaptation capacity together with layer placement. To isolate placement-dependent cost, we compare placements with the same adapter capacity. Layer placement determines how far the loss gradient must pass through the frozen backbone. Placing LoRA layers close to the output shortens gradient paths and reduces the portion of the network for which activations must be retained. Placing LoRA layers close to the input requires propagating gradients through a larger fraction of the frozen backbone (Xu et al., 2026). Figure 1 illustrates the difference in backward floating-point operations (FLOPs). Our placement objective combines a task-score constraint with shorter backward paths through pretrained weights. We therefore ask: *How can we select a LoRA adapter placement that preserves task performance while reducing backpropagation through the frozen backbone?*

Fine-tuning cost. Full fine-tuning stores pretrained weights, their gradients, and optimizer states. DINOv2-G/14 has approximately 1.136 billion backbone parameters (Oquab et al., 2024). Assuming float32 storage without optimizer sharding, its weights and gradients each occupy about 4.23 gibibytes

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.

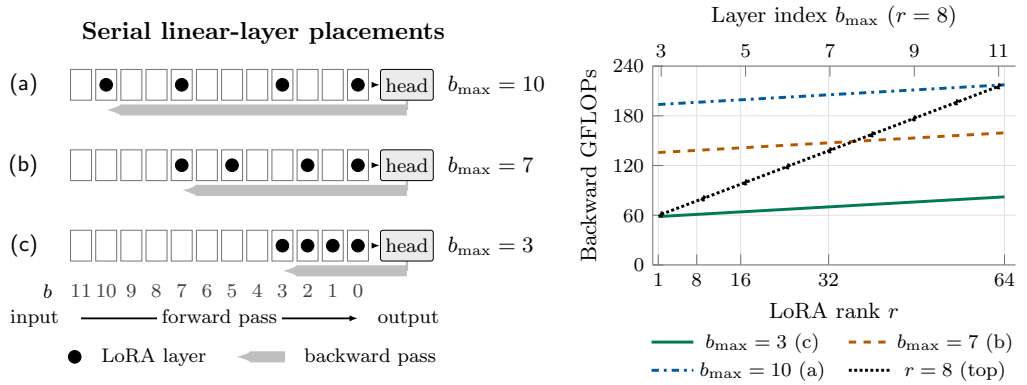

Figure 1: LoRA placement and analytical backward cost in a serial chain of linear layers, with $K = 4$ LoRA layers per placement. The farthest LoRA layer's output-first index $b_{\max}$ determines the backward path (left). On the right, colored curves vary rank r at fixed $b_{\max} = 3, 7, 10$ (bottom axis). The black dotted curve varies $b_{\max}$ at fixed rank $r = 8$ (independent top axis). Appendix B.3 gives the assumptions.

(GiB, 2^{30} bytes). Two Adam moment tensors require another 8.47 GiB. Together, these tensors occupy 16.93 GiB, excluding activations, task-head tensors, and temporary buffers. Rank-eight LoRA on its 160 attention projections introduces 3.93 million trainable backbone parameters, or 0.346% of the backbone. Their float32 weights, gradients, and two Adam moments occupy 60 mebibytes (MiB, 2^{20} bytes). The frozen backbone weights remain resident. These are parameter-storage estimates, not measured peak GPU memory. LoRA reduces parameter-related storage, but does not remove backbone forward computation or activation gradients needed to train LoRA layers nearer the input. Placement near the output can shorten this backward path and reduce retained activations (Xu et al., 2026). Fine-tuning cost therefore depends on both the amount and placement of trainable adaptation.

Placement objective and comparison design. Our search evaluates placements in the last layers under a held-out selection-score constraint. We examine placement-dependent training cost with LoRA rank, adapter count, and projection dimensions fixed. Moving LoRA layers toward the output shortens backpropagation through pretrained weights (Xu et al., 2026). A fixed budget in the last layers can perform well. However, a fixed adapter count may be too high or too low for a given task.

We introduce Score-Constrained LoRA Search in the Last Layers (LaLoS), motivated by projection-level backward-cost analysis at matched adapter capacity. It searches placements close to the output under a held-out selection-score constraint. Evaluation reports task performance, training time, FLOPs, and peak memory. Our contributions are as follows:

1. We analyze how LoRA placement in the last layers affects backward-pass computation. These layers are nearest to the output. The analysis holds LoRA rank and the trainable-parameter budget fixed. Based on this analysis, we introduce the fixed Last-K baseline, where K is the number of LoRA layers. It adapts the output-side half of the eligible projection layers.
2. Building on this analysis, we introduce LaLoS. It evaluates placements through forward passes on one frozen probe. A held-out selection score determines how many of the last layers receive LoRA for each task. It then selects which LoRA layers to retain under the same score constraint. The resulting placement is fine-tuned from the original pretrained checkpoint.
3. We compare LaLoS, fixed Last-K baselines, and other PEFT methods across language and vision benchmarks. We assess task performance, FLOPs, elapsed time, and peak memory.

Section 3 derives placement-dependent backward cost, and Section 4 presents LaLoS. Section 5 summarizes the language and vision experimental setup, and Section 6 reports the results. Sections 7 and 8 discuss limitations and conclusions. The appendices provide algorithmic details, cost derivations, detailed protocols, complete task-level results, and sensitivity analyses. The reproducibility and responsible-use statements follow the conclusion.

2 Related Work

Parameter-efficient adaptation. PEFT methods train small adapter modules (Houlsby et al., 2019), continuous prefixes (Li & Liang, 2021), or prompt embeddings (Lester et al., 2021). BitFit instead updates bias parameters (Ben Zaken et al., 2022). He et al. (2022) connect several PEFT methods through their modifications to pretrained hidden states. These designs determine how adaptation is parameterized. Our focus is where standard LoRA updates are placed.

Layer placement and module selection. LoRA adapter placement determines which frozen weight matrices receive low-rank updates. The original LoRA study compares which attention projection types receive updates (Hu et al., 2022). Precise LoRA Placement (PLoP) ranks target module types using module–data alignment from forward passes (Hayou et al., 2026). AdapterDrop removes lower-layer adapters during training and inference (Rücklé et al., 2021). Parameter, Memory, and Time Efficient Visual Adapter (E^3VA) avoids backpropagation through a frozen visual backbone (Yin et al., 2024), whereas Layer-wise LoRA selects layers using representation similarity (Ogawa et al., 2026). Layer Card records residual signal, activation conditioning, task performance, and computational cost for layer groups (Xu et al., 2026). It matches a target task to reference tasks and leaves the trade-off between task performance and fine-tuning cost to the user. These studies motivate cost-aware placement. Our analysis quantifies projection-level backward FLOPs and logical memory traffic at matched adapter capacity, while LaLoS uses a selection-score constraint to guide layer placement.

Rank allocation and parameter selection. Adaptive methods allocate rank or select trainable parameters and adapters. AdaLoRA allocates rank using importance scores and prunes low-importance singular values without repeatedly computing a singular value decomposition (Zhang et al., 2023b). Integrated-gradient sensitivity and uncertainty-aware scores determine layerwise ranks in IGU-LoRA (Cui et al., 2026). Sensitivity-aware visual Parameter-efficient fine-Tuning (SPT) selects parameters by sensitivity and combines unstructured and structured tuning (He et al., 2023a). During a short warm-up with a sparsity constraint, WeightLoRA learns importance weights and retains only adapters with nonzero weights (Veprikov et al., 2026). Explained Variance Adaptation (EVA) uses activation-informed initialization to redistribute rank (Paischer et al., 2025), whereas Weight-Decomposed Low-Rank Adaptation (DoRA) changes the low-rank update parameterization (Liu et al., 2024). These methods vary LoRA rank, parameter selection, adapter selection, or update parameterization. Our cost analysis holds the LoRA configuration fixed across placements, and LaLoS selects placements by held-out scores from a frozen probe rather than learned importance weights.

Memory and computation efficiency. QLoRA quantizes frozen pretrained weights while retaining gradient propagation to LoRA factors (Dettmers et al., 2023). LoRA-FA freezes the projection-down factor to avoid storing full-rank input activations (Zhang et al., 2023a). Ladder Side-Tuning trains a separate side network without backpropagation through the backbone (Sung et al., 2022). Computation-Efficient LoRA (CE-LoRA) approximates matrix products used to compute activation gradients (Chen et al., 2025). These methods change weight storage, trainable factors, network structure, or gradient computation. Our placement search shortens the backward path while retaining standard LoRA gradients on that path.

3 LoRA Placement and Backward Cost

LoRA freezes the pretrained weights (Hu et al., 2022), but adapter placement changes how far gradients propagate through them (Xu et al., 2026). The model has L eligible linear projection layers, indexed $b = 0, \dots, L-1$ in a fixed order from output to input. Layer 0 is nearest the output. In self-attention, the query, key, and value projections operate in parallel on the same input. Their indices identify layers for selection. The selected layer indices form the support S. Its size $K = |S|$ is the number of LoRA layers. Layer b has LoRA rank r_b and input/output widths $d_{\mathrm{in},b}, d_{\mathrm{out},b}$. Its trainable factors are A_b and B_b, with dimensions shown beneath their product,

$$\underbrace{B_b}_{d_{\mathrm{out},b}\times r_b}\ \underbrace{A_b}_{r_b\times d_{\mathrm{in},b}}\ .$$

Each selected layer has its own factor pair, containing $r_b d_{\text{in},b} + r_b d_{\text{out},b}$ parameters. The number of trainable LoRA parameters in support S is $P_{\text{LoRA}}(S)$:

$$P_{\text{LoRA}}(S) = \sum_{b \in S} r_b \big(d_{\text{in},b} + d_{\text{out},b}\big). \tag{1}$$

This sum allows different ranks and widths at different layers and excludes the separately trained task head. We isolate placement by matching the number, ranks, widths, and projection types of the LoRA layers across placements.

Backward cost of LoRA placement. The analysis assumes that only adapters and the task head are trainable, with no model-input gradients. The illustrative model for Figure 1 is a serial chain of linear layers, with common widths $d_{\text{in}}, d_{\text{out}}$ and LoRA rank r across placements, matching Eq. (1). N counts processed token positions per physical minibatch, including padding. For fixed-length inputs, N equals batch size times sequence length. Local tensors carry the layer index b. The dropout-free derivation assumes inputs $X_b \in \mathbb{R}^{d_{\text{in}} \times N}$, with one token position per column. $W_{0,b} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ contains that projection's frozen weights, with 0 denoting the pretrained component. For a LoRA projection, α is the scaling coefficient, A_b and B_b have the common dimensions above, and ΔW_b is their low-rank weight update. $\mathcal{L}$ is the scalar training loss, and $G_b \in \mathbb{R}^{d_{\text{out}} \times N}$ is its gradient at projection output Y_b, received from operations closer to the output. With ∇ denoting a gradient and $^\top$ matrix transposition, a LoRA projection's parameterization and parameter gradients are

$$\begin{aligned} \Delta W_b &= \tfrac{\alpha}{r} B_b A_b, \qquad Y_b = W_{0,b} X_b + \tfrac{\alpha}{r} B_b (A_b X_b), \qquad G_b = \nabla_{Y_b} \mathcal{L}, \\ \nabla_{A_b} \mathcal{L} &= \tfrac{\alpha}{r} (B_b^\top G_b) X_b^\top, \qquad \nabla_{B_b} \mathcal{L} = \tfrac{\alpha}{r} G_b (A_b X_b)^\top. \end{aligned} \tag{2}$$

Both parameter gradients use the received G_b, following the reverse-mode chain rule (Baydin et al., 2018). A directly connected chain illustrates its source, assuming $Y_b = X_{b-1}$ for $b > 0$. With common widths across projections, this requires $d_{\text{in}} = d_{\text{out}}$. For nonempty S, $b_{\max}(S) = \max S$ is the index of the LoRA layer farthest from the output. We write $b_{\max}$ for a fixed S. $b_{\max}(S)$ is undefined for empty support. Layers with indices $b < b_{\max}$ may also contain LoRA. $G_{\text{base},b-1}$ and $G_{\text{LoRA},b-1}$ are the pretrained-weight and LoRA input-gradient contributions of projection $b-1$. The cases below describe that producing projection, with $G_{\text{LoRA},b-1}$ defined only when it contains LoRA:

$$\underbrace{G_{\text{base},b-1}}_{d_{\text{in}} \times N} = \underbrace{W_{0,b-1}^\top}_{d_{\text{in}} \times d_{\text{out}}} \underbrace{G_{b-1}}_{d_{\text{out}} \times N}, \quad G_{\text{LoRA},b-1} = \Delta W_{b-1}^\top G_{b-1} = \tfrac{\alpha}{r} A_{b-1}^\top (B_{b-1}^\top G_{b-1}),$$

$$G_b = \nabla_{X_{b-1}} \mathcal{L} = \begin{cases} G_{\text{base},b-1}, & \text{pretrained weights only,} \\ G_{\text{base},b-1} + G_{\text{LoRA},b-1}, & \text{LoRA layer,} \end{cases} \qquad 1 \le b \le b_{\max}. \tag{3}$$

The LoRA parameter gradients $\nabla_{A_{b_{\max}}} \mathcal{L}$ and $\nabla_{B_{b_{\max}}} \mathcal{L}$ are computed using $G_{b_{\max}}$, without the input gradient $\nabla_{X_{b_{\max}}} \mathcal{L}$ (Rücklé et al., 2021). At $b_{\max} = 0$, G_0 arrives directly from operations closer to the output. Freezing skips pretrained-weight gradients and optimizer state, not required gradient propagation. In a transformer, attention branches and intervening operations determine the required gradients and follow their own chain rules.

For each modeled projection, the local cost vectors are $\mathbf{c}_{\text{base}}$, $\mathbf{c}_{\text{param}}$, and $\mathbf{c}_{\text{LoRA}}$ for the frozen-weight input gradient, LoRA parameter gradients, and additional LoRA input gradient, respectively. Their FLOP components are $2N d_{\text{in}} d_{\text{out}}$, $2Nr(2d_{\text{out}} + d_{\text{in}})$, and $2Nr d_{\text{in}}$. This counts two FLOPs per multiply–add and reuses $A_b X_b$ and $B_b^\top G_b$. Appendix B.2 derives the complete FLOP/read/write vectors and their exclusions. Logical memory traffic counts operand reads and result writes. It is distinct from GPU transactions and peak allocated memory.

These local costs are fixed at matched N, widths, and rank. $\mathbf{c}_{\text{path}}(S)$ is the modeled projections' backward subtotal, excluding intervening operations and the task head. For a sole LoRA layer ($K = 1$) at index $b_{\max}$, each projection $b = 0, \ldots, b_{\max} - 1$ has pretrained weights only and contributes one input-gradient product. For example, the direct chain with $b_{\max} = 3$ uses only the first case of Eq. (3), and $G_3 = W_{0,2}^\top G_2$ supplies both LoRA parameter gradients. Therefore,

$$\mathbf{c}_{\text{path}}(S) = b_{\max} \mathbf{c}_{\text{base}} + \mathbf{c}_{\text{param}}. \tag{4}$$

Increasing $b_{\max}$ by one adds $\mathbf{c}_{\text{base}}$, increasing FLOPs and logical reads and writes. At $b_{\max} = 0$, only the parameter gradients remain in this projection cost. For K LoRA layers in the same model,

$\mathbf{c}_{\text{param}}$ is counted at every LoRA layer, including layer $b_{\max}$. $\mathbf{c}_{\text{LoRA}}$ is counted only at the $K-1$ LoRA layers with $b < b_{\max}$, where input gradients are required. The shared pretrained-weight path is counted once, giving

$$\mathbf{c}_{\text{path}}(S) = b_{\max}\mathbf{c}_{\text{base}} + K\mathbf{c}_{\text{param}} + (K-1)\mathbf{c}_{\text{LoRA}}. \tag{5}$$

The $b_{\max}\mathbf{c}_{\text{base}}$ term includes pretrained-weight input gradients in both LoRA layers and projections with pretrained weights only. At fixed N, widths, rank, and K, only the farthest layer index $b_{\max}$ varies across these chain placements. For $S = \varnothing$, this backbone cost is zero. With multiple selected layers on parallel branches, the required LoRA input-gradient products need not total $K-1$. Their dependencies are discussed in Appendix B. Figure 1 shows the resulting rank-dependent slopes and offsets determined by the farthest layer index. Appendix B.3 gives the assumptions and worked examples.

Layer-placement comparisons. **Last-K** adapts the K eligible projection layers nearest the output. For $K > 0$, their indices are $b = 0, \ldots, K-1$, with last selected index $b_{\max} = K-1$. It matches adapter count, but layer identities and projection types may differ from those in a learned placement. Equal adapter counts match the trainable-parameter budget only at matched ranks and projection dimensions. Full LoRA adapts every eligible projection, whereas Head-only trains the classifier and pooler without backbone adapters. The Equal-Span Dense Support (ESDS) control extends a learned placement to a dense output-side layer set. Its GLUE construction appears in Appendix F. Densifying a placement can change adapter count, the trainable-parameter budget, optimization, and regularization.

4 Score-Constrained LoRA Search in the Last Layers

For each task, LaLoS selects how many of the last layers receive LoRA before fine-tuning. Candidate placements are represented by support S. Each evaluation, called a query, uses forward passes without backpropagation or parameter updates. An adapter mask activates LoRA only at layers indexed by S. All queries share one model with fixed parameters (the probe) and data reserved for adapter selection.

Frozen probe and score constraint. Using the output-first indices from Section 3, $S_b = \{0, \ldots, b\}$ places LoRA at every eligible layer from layer 0 through layer b. For nonempty S_b, the last selected index is $b_{\max}(S_b) = b$ and the layer count is $K = |S_b| = b + 1$. Within this placement family, a smaller b selects fewer LoRA layers from the output end. A larger b includes more layers toward the input. The empty placement is $S_{-1} = \varnothing$, while S represents an arbitrary placement. Training data are partitioned into an adapter-training subset and a disjoint, held-out selection subset. For a final fine-tuning schedule of T optimizer updates, ρ is the fraction used for initial LoRA training (warm-up). LaLoS trains all eligible adapters, S_{L-1}, using the adapter-training subset for $\lceil \rho T \rceil$ updates, the smallest integer greater than or equal to ρT. The saved model checkpoint is then frozen in evaluation mode for placement selection. The empty placement disables backbone adapters but retains the task head trained during warm-up. $\mathcal{A}(S)$ is the probe's higher-is-better task score on the selection subset under placement S. The allowed score drop from full support is $\epsilon \geq 0$. The selection-score constraint is

$$S \text{ is feasible } \iff \mathcal{A}(S) \geq \mathcal{A}(S_{L-1}) - \epsilon. \tag{6}$$

Search over the last layers. The fixed list $\mathcal{E}$ contains the allowed placement indices in ascending order, including -1 and $L-1$. Algorithm 1 first queries the full and empty placements. A placement passes if its score satisfies Eq. (6). If the empty placement passes, no LoRA layers are selected. Otherwise, binary search starts with a failing lower bound $b_{\text{lo}} = -1$ and a passing upper bound $b_{\text{hi}} = L-1$. While candidates remain between the bounds, binary search tests the middle placement S_b. If there are two middle entries, the larger index is used. A pass sets $b_{\text{hi}} = b$. Otherwise, $b_{\text{lo}} = b$.

After binary search ends, the score constraint is checked for S_b at the nearest candidate in $\mathcal{E}$ below b_{lo}, then at the nearest candidate above b_{hi}. A check is skipped if no candidate exists on that side. Each check evaluates the complete placement S_b with the frozen probe and reuses its stored score if available. After these checks, the smallest passing index queried so far becomes $b^\star$, the upper bound for the single restart. The bounds are reset to $b_{\text{lo}} = -1$ and $b_{\text{hi}} = b^\star$. Binary search runs once more

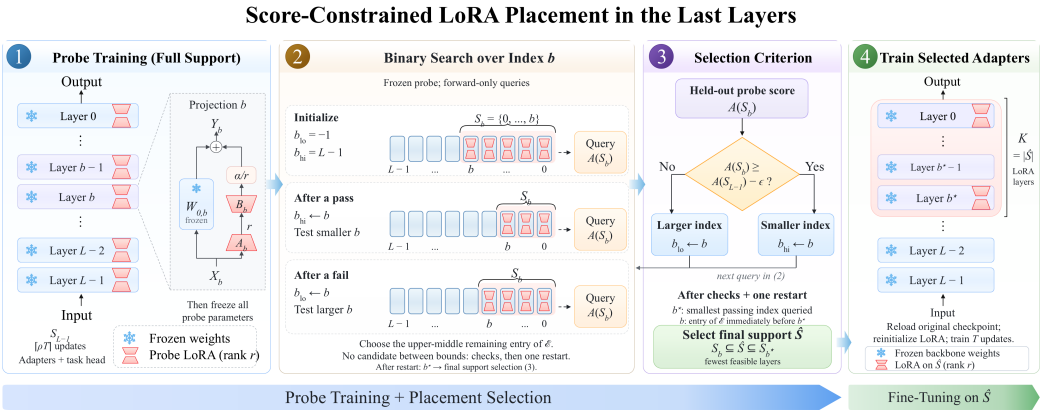

Figure 2: Overview of LaLoS. A briefly trained, frozen probe evaluates placements through forward-only queries. Score-constrained search selects $\widehat{S}$. Final fine-tuning reloads the pretrained checkpoint and reinitializes the selected LoRA factors. Algorithm 1 gives the search rules.

with the same update and stopping rules, reusing stored scores. Its final upper bound becomes $b^\star$. The final $b^\star$ is the smallest passing index among the queried candidates. When scores are nonmonotonic, an unqueried smaller placement may still pass. The index $b = \max\{b \in \mathcal{E} : b < b^\star\}$ immediately precedes $b^\star$ in $\mathcal{E}$. The returned placement $\widehat{S}$ retains all layers in S_b and contains only layers from $S_{b^\star}$. It uses the fewest LoRA layers meeting the score threshold under these constraints:

$$\widehat{S} \in \underset{\substack{S_b \subseteq S \subseteq S_{b^\star} \\ S \text{ feasible}}}{\arg\min} |S|. \tag{7}$$

Standard LoRA fine-tuning. The probe is discarded, the original pretrained checkpoint is reloaded, and LoRA factors are reinitialized at the layers indexed by $\widehat{S}$. The selected placement remains fixed during the T-step LoRA fine-tuning schedule. Appendix A gives the complete procedure, illustrated in Figure 2.

Head-only and full-support configurations. An empty support trains only the task head. Full support trains all eligible adapters. Both retain the probe and query cost. The experimental query families and reference policy are specified in Section 5.2 and Appendix C.

5 Experimental Setup

The experiments test whether layer placement changes training cost at fixed adapter capacity and whether a score constraint can choose an adapter placement for each task. The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) provides the primary controlled comparison. The Visual Task Adaptation Benchmark (VTAB-1k) (Zhai et al., 2019) tests these questions with a different backbone and modality.

5.1 Benchmarks and Evaluation Protocol

We evaluate DeBERTaV3-base (He et al., 2023b) on seven GLUE classification tasks: SST-2, QNLI, CoLA, RTE, MRPC, QQP, and MNLI. For LaLoS, adapter selection uses a validation subset reserved from each training set. The probe is trained on the remaining examples. Final fine-tuning uses the full training set. Performance is evaluated on the official development sets. We also evaluate DINOv2-G/14 (Oquab et al., 2024) on all nineteen VTAB-1k tasks. LaLoS probe training and adapter selection use separate training and validation subsets. Final fine-tuning uses both subsets together, and performance is evaluated on the test sets.

After selection, LaLoS fine-tunes from the original checkpoint with reinitialized LoRA factors. Each task uses a fixed number of optimizer updates. Only the final checkpoint is evaluated. Evaluation

data are not used for adapter or checkpoint selection. Unless otherwise specified, LoRA uses rank $r = 8$. AdaLoRA and IGU-LoRA are exceptions. We use five optimization seeds and NVIDIA H200 GPUs. Task metrics, eligible projections, and training settings appear in Appendix E.

5.2 Comparison design

We compare LoRA placements with the same rank, adapter count, and projection dimensions. This keeps the trainable-parameter budget fixed and tests the cost analysis in Section 3. The Last-K baseline adapts the output-side half of the eligible projection layers, with $K = L/2$. GLUE compares Last-18 with WeightLoRA (Veprikov et al., 2026) at $K = 18$. VTAB compares Last-80, WeightLoRA-80, and PLoP-Type2 (Hayou et al., 2026) at $K = 80$.

The broader comparison asks what each selection or allocation mechanism achieves at its realized configuration. WeightLoRA, PLoP, hybrid SPT (He et al., 2023a), AdaLoRA (Zhang et al., 2023b), and IGU-LoRA (Cui et al., 2026) select different objects and need not realize identical budgets. We report performance together with allocated adaptation parameters, adapter placement, and resources under shared final fine-tuning configurations.

The fixed Last-K baselines use the same adapter count K across tasks. LaLoS instead selects how many of the last layers receive LoRA for each task. It uses a held-out probe-score constraint. The transferred probe design and score tolerances of 0.5 and 1.0 percentage points are fixed. Its support may be empty, intermediate, or full. Head-only and Full LoRA provide the corresponding final configurations. Algorithm 1 presents the detailed LaLoS procedure. Appendix C details the benchmark search configurations.

5.3 Computation and systems measurements

The main comparison reports final fine-tuning cost. F_{ft} is the FLOP total of supported operations in the forward and backward passes, and T_{ft} is elapsed wall time. M_{ft} is peak allocated GPU memory, reported in gibibytes (GiB, 2^{30} bytes). K denotes the number of active LoRA layers. Table 1 reports K/L as a percentage, with $L = 36$ eligible layers on GLUE and $L = 160$ on VTAB. P_A is the number of allocated backbone adaptation parameters, excluding the task head, reported in millions.

For each resource, U denotes the corresponding Full-LoRA value on the same task and optimization seed. Table 1 reports FLOP and time savings as $100(1 - F_{\mathrm{ft}}/U)$ and $100(1 - T_{\mathrm{ft}}/U)$, respectively, in percent. Task/seed ratios are formed before averaging. Higher savings indicate lower cost. Memory values average per-run peaks over tasks and seeds. Scores report the mean and sample standard deviation (SD) across seed-level benchmark means. Appendix F gives the aggregation rule. Appendix D provides counter implementations and measurement scopes.

6 Results and Analysis

6.1 Layer placement and fine-tuning cost

We compare placements at matched rank and active projection-layer count to isolate placement-dependent training cost in Table 1. Appendices F and G provide complete per-task GLUE and VTAB results, respectively.

The fixed Last-K baseline activates half of the L eligible projection layers, $K = L/2$: Last-18 uses 18 of 36 on GLUE, and Last-80 uses 80 of 160 on VTAB. On GLUE, Last-18 and WeightLoRA-18 each train 221,184 adapter parameters at different layer placements. Relative to Full LoRA, they save 26.9% and 6.0% of final fine-tuning FLOPs and 47.1% and 10.6% of training time, respectively (Table 1a). The learned placements require longer backward paths through pretrained weights.

For vision, Last-80, WeightLoRA-80, and PLoP-Type2 each use 1,966,080 adapter parameters at different layers. Relative to Full LoRA, Last-80 saves 25.7% of final fine-tuning FLOPs and 38.5% of training time. WeightLoRA-80 saves 1.7% and 26.0%, respectively (Table 1b). Equal parameter counts therefore do not imply equal fine-tuning computation.

Table 1: Performance and final fine-tuning efficiency. Scores are means ± sample SD across five seed-level benchmark averages. The $b_{\max}$ column reports the mean last selected projection index, with zero nearest the output. Savings are relative to uniform Full LoRA. Excluding Head-only and the descriptive $b_{\max}$ column, **bold** and underlining mark the best and second-best distinct values per benchmark and column. Ties share formatting. Light blue rows denote LaLoS.

(a) GLUE-7 (Language), DeBERTaV3-base

Method	Budget rule	Score ↑	$K/36$ ↓	$b_{\max}$ ↓	P_A ↓ (M)	Savings ↑ FLOPs	Savings ↑ Time	M_{ft} ↓ (GiB)	Score/ FLOPs ↑
Head-only	None	68.65 ± 0.13	—	—	—	52.9%	94.1%	3.51	145.75
Full LoRA	All	85.99 ± 0.20	100.0%	35.0	0.442	0.0%	0.0%	37.24	85.99
WeightLoRA-18	Fixed K	85.56 ± 0.38	50.0%	31.7	0.221	6.0%	10.6%	33.97	91.02
PLoP-Type1	Layer Types	83.98 ± 0.17	**33.3%**	33.0	**0.147**	5.3%	8.9%	34.25	88.68
PLoP-Type2	Layer Types	85.69 ± 0.33	66.7%	34.0	0.295	2.6%	0.7%	36.17	87.98
Hybrid SPT	Params.	85.98 ± 0.19	80.6%	35.0	0.361	−1.7%	0.2%	37.15	84.54
AdaLoRA	Rank	78.21 ± 0.16	95.1%	34.2	0.664	−0.3%	−3.9%	37.26	77.98
IGU-LoRA	Rank	74.19 ± 0.42	99.7%	35.0	0.664	−0.3%	−5.1%	37.28	73.97
Last-18 (Baseline)	Fixed K	86.02 ± 0.18	50.0%	17.0	0.221	**26.9%**	**47.1%**	**24.29**	**117.67**
LaLoS-0.5 (Ours)	Task-wise K	85.95 ± 0.22	65.6%	22.7	0.290	17.1%	33.9%	28.24	103.68
LaLoS-1.0 (Ours)	Task-wise K	**86.12 ± 0.28**	54.8%	18.9	0.242	22.8%	44.0%	25.62	111.55

(b) VTAB-19 (Vision), DINOv2-G/14

Method	Budget rule	Accuracy ↑ (%)	$K/160$ ↓	$b_{\max}$ ↓	P_A ↓ (M)	Savings ↑ FLOPs	Savings ↑ Time	M_{ft} ↓ (GiB)	Score/ FLOPs ↑
Head-only	None	$61.53\% \pm 0.03\%$	—	—	—	51.1%	76.7%	4.40	125.83
Full LoRA	All	$77.26\% \pm 0.58\%$	100.0%	159.0	3.932	0.0%	0.0%	10.98	77.26
WeightLoRA-80	Fixed K	$78.01\% \pm 0.40\%$	50.0%	153.5	1.966	1.7%	26.0%	10.43	79.36
PLoP-Type1	Layer Types	$77.34\% \pm 0.82\%$	25.0%	156.1	0.983	0.5%	38.4%	10.23	77.73
PLoP-Type2	Layer Types	$77.39\% \pm 0.62\%$	50.0%	157.0	1.966	0.2%	24.4%	10.48	77.55
Hybrid SPT	Params.	$77.22\% \pm 0.09\%$	**9.2%**	157.2	**0.403**	−9.3%	36.0%	10.78	70.65
AdaLoRA	Rank	$76.55\% \pm 0.10\%$	95.0%	158.0	5.900	−0.2%	−57.6%	11.11	76.40
IGU-LoRA	Rank	$76.51\% \pm 0.32\%$	99.7%	158.9	5.900	−0.3%	−95.4%	11.12	76.28
Last-80 (Baseline)	Fixed K	**$78.63\% \pm 0.21\%$**	50.0%	79.0	1.966	25.7%	38.5%	7.65	105.83
LaLoS-0.5 (Ours)	Task-wise K	$78.24\% \pm 0.73\%$	47.9%	78.1†	1.882	26.3%	40.2%	7.56	106.16
LaLoS-1.0 (Ours)	Task-wise K	$78.22\% \pm 0.84\%$	43.3%	73.8†	1.701	**28.7%**	**43.4%**	**7.26**	**109.71**

Savings average $100(1 - \text{ratio})$, where each FLOP or time ratio divides method cost by Full-LoRA cost for the same task and seed. Negative savings indicate increased cost. Score/FLOPs divides the displayed mean score by $1 -$ FLOP savings$/100$, excluding selection. K/L reports active-layer percentages, with $L = 36$ for GLUE and 160 for VTAB. K counts structured SPT adapters and positive-final-rank adapters for adaptive-rank methods. — denotes empty support. Hybrid SPT indices include unstructured updates. †: indices are averaged over nonempty task placements within each seed. Other metrics use all runs. Appendix D.3 gives aggregation details. Method suffixes give K for Last and WeightLoRA, or tolerance in pp for LaLoS. Fixed and task-wise K use preset and task/seed-dependent adapter counts. Layer Types, Params., and Rank denote module-type, parameter, and rank budgets. P_A counts allocated backbone adaptation parameters, excluding the head. M_{ft} is the mean per-run peak allocated GPU memory.

6.2 Task-dependent adaptation with LaLoS

On GLUE, LaLoS-1.0 selects a mean of 19.71 active projection layers, from 14.8 on MRPC to 25.6 on CoLA. It ranks first at 86.12 ± 0.28. Last-18 ranks second at 86.02 ± 0.18, just 0.10 points below LaLoS-1.0. It uses fewer active projection layers and less final fine-tuning computation and time than LaLoS-1.0 (Table 1).

On VTAB-1k, Last-80 ranks first at $78.63 \pm 0.21\%$. LaLoS-0.5 ranks second at $78.24 \pm 0.73\%$, 0.39 percentage points below Last-80. It averages 76.6 active projection layers and uses slightly less final fine-tuning computation and time than Last-80 (Table 1). LaLoS-1.0 averages 69.2 active projection layers and scores $78.22 \pm 0.84\%$, while Full LoRA scores $77.26 \pm 0.58\%$. LaLoS selects how many of the last projection layers receive LoRA for each task. This makes fine-tuning computation task-dependent. Fixed Last-K placements remain competitive.

Empty support ($K = 0$) retains selection and head-training costs. Full support has $K = 36$ for language and $K = 160$ for vision. For the low-accuracy vision probes, reported final outcomes use matched Full-LoRA runs with selection costs retained. This reference policy keeps all tasks included in the benchmark results.

We also conducted an ablation study on EuroSAT and the Describable Textures Dataset (DTD). We varied warm-up budget and score tolerance to measure their effects on final accuracy and the number of LoRA layers. Appendix H reports these results.

7 LIMITATIONS

Probe warm-up and subsequent selection queries can offset fine-tuning savings. Table 2 reports warm-up FLOPs of supported operations F_{warm}, elapsed time T_{warm}, and peak allocated GPU memory M_{warm}. Probe training can retain Full LoRA's peak memory despite a smaller final placement. The probe score is an approximate proxy for final performance. Its constraint need not hold after final fine-tuning. The recorded searches can miss feasible placements if nonmonotonicity does not trigger their scans. Even at fixed adapter capacity, accuracy differences also depend on projection type and selection.

Table 2: Warm-up cost only, excluding setup, calibration, selection queries, and final fine-tuning.

GLUE-7 (Language), DeBERTaV3-base				VTAB-19 (Vision), DINOv2-G/14			
Method	F_{warm}	T_{warm}	M_{warm}	Method	F_{warm}	T_{warm}	M_{warm}
WeightLoRA-18	1.046	2.561	38.465	WeightLoRA-80	0.392	0.169	11.926
LaLoS-0.5 (Ours)	20.779	50.212	37.242	LaLoS-0.5 (Ours)	2.937	1.159	10.984
LaLoS-1.0 (Ours)	20.779	50.212	37.242	LaLoS-1.0 (Ours)	2.937	1.159	10.984

Units: PFLOP (10^{15} FLOPs), min, and GiB for F_{warm}, T_{warm}, and M_{warm}. Means use 35 GLUE or 95 VTAB task/seed outcomes per method. WeightLoRA suffixes specify K. Each LaLoS tolerance includes full probe training. Omitted methods have no separate warm-up but may incur selection cost.

Future work could extend the fixed-budget study in Appendix H to adaptive probe stopping. Alternative searches and broader placement families could be compared under equal query budgets to identify lower-cost selection procedures. Activation checkpointing could reduce probe memory at the cost of recomputation.

8 CONCLUSION

Our analysis quantifies how LoRA placement changes backward computation at matched adapter capacity. Placing LoRA in the last layers shortens gradient paths through pretrained weights. Placement-cost comparisons across language and vision hold adapter capacity fixed. Last-layer placements require fewer final fine-tuning FLOPs and less elapsed time than corresponding WeightLoRA placements. LaLoS uses frozen-probe scores to select how many of the last layers receive LoRA for each task. The selected adapter count and final fine-tuning cost vary across tasks. LaLoS-1.0 achieves the highest mean GLUE score, while fixed Last-80 achieves the highest mean VTAB score. These results identify layer placement as a determinant of fine-tuning cost beyond rank and trainable-parameter count.

REPRODUCIBILITY AND RESPONSIBLE USE

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the LoRA placement formulation, backward-cost assumptions, and search procedure. Section 5 presents the experimental design. Appendices A and E provide algorithmic details, data partitions, adapter configurations, optimization schedules, and evaluation protocols. Appendix D specifies computation-counter coverage and resource normalization. The task-level results and sensitivity study document the evaluated configurations and aggregation rules, including matched Head-only and Full-LoRA references. We include reproducibility code with this submission and are committed to making the full codebase publicly available on GitHub upon acceptance.

ETHICS STATEMENT

This work studies LoRA placement and fine-tuning cost using existing language and vision benchmarks and pretrained models. It introduces no new human-subject data. Adapted models may inherit biases and other limitations of the datasets and pretrained representations. Computational efficiency does not establish fairness, privacy, or suitability for high-stakes applications. Dataset and model reuse remains subject to the respective licenses and access conditions.

USE OF LARGE LANGUAGE MODELS

Generative AI tools assisted manuscript preparation, including language drafting and revision, LaTeX organization, figure code, and consistency checks. The authors remain responsible for the methods, experimental records, citations, results, conclusions, and content of the paper.

REFERENCES

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

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. Association for Computational Linguistics, May 2022. doi: 10.18653/v1/2022.acl-short.1. URL https://aclanthology.org/2022.acl-short.1/.

Guanduo Chen, Yutong He, Yipeng Hu, Kun Yuan, and Binhang Yuan. CE-LoRA: Computation-efficient LoRA fine-tuning for language models, 2025. URL https://arxiv.org/abs/2502.01378.

Xuan Cui, Huiyue Li, Run Zeng, Yunfei Zhao, Jinrui Qian, Wei Duan, Bo Liu, and Zhanpeng Zhou. IGU-LoRA: Adaptive rank allocation via integrated gradients and uncertainty-aware scoring. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=MnToYQx9My.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023. doi: 10.52202/075280-0441. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract.html.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*, 2005. URL https://aclanthology.org/I05-5002/.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. PLoP: Precise LoRA placement for efficient finetuning of large models. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/6ace8b8873adf5af379e8b5e0dcd8664-Paper-Conference.pdf.

Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Sensitivity-aware visual parameter-efficient fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11825–11835, 2023a. doi: 10.1109/ICCV51070.2023.01086. URL https://openaccess.thecvf.com/content/ICCV2023/html/He_Sensitivity-Aware_Visual_Parameter-Efficient_Fine-Tuning_ICCV_2023_paper.html.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=0RDcd5Axok.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=sE7-XhLxHA.

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.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059. Association for Computational Linguistics, November 2021. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597. Association for Computational Linguistics, August 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

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 *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6ZBHIEtdP4.

NVIDIA. *Matrix Multiplication Background User's Guide*. NVIDIA Corporation, 2023. URL https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html. Updated February 1, 2023.

Keith Ando Ogawa, Bruno Lopes Yamamoto, Lucas Lauton de Alcantara, Lucas Pellicer, Rosimeire Pereira Costa, Edson Bollis, Anna Helena Reali Costa, and Artur Jordao. Layer-wise lora fine-tuning: a similarity metric approach. In *Pattern Recognition. ICPR 2026*, volume 16819 of *Lecture Notes in Computer Science*, pp. 370–385. Springer Nature Switzerland, 2026. doi: 10.1007/978-3-032-31404-8_25.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=a68SUt6zFt. Featured Certification.

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, 2025. doi: 10.52202/085713-1539. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/41d33bd41fd44bd9dba0e092047cf213-Paper-Conference.pdf.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, November 2016. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7930–7946. Association for Computational Linguistics, November 2021. doi: 10.18653/v1/2021.emnlp-main.626. URL https://aclanthology.org/2021.emnlp-main.626/.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://aclanthology.org/D13-1170/.

Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. LST: Ladder side-tuning for parameter and memory efficient transfer learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 12991–13005. Curran Associates, Inc., 2022. doi: 10.52202/068431-0944. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/54801e196796134a2b0ae5e8adef502f-Abstract-Conference.html.

Andrey Veprikov, Vladimir Solodkin, Zyl Alexander, Andrey Savchenko, and Aleksandr Beznosikov. WeightLoRA: Keep only necessary adapters. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12419–12437. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.566. URL https://aclanthology.org/2026.acl-long.566/.

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 *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL https://aclanthology.org/W18-5446/.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641, 2019. doi: 10.1162/tacl_a_00290. URL https://aclanthology.org/Q19-1040/.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122. Association for Computational Linguistics, June 2018. doi: 10.18653/v1/N18-1101. URL https://aclanthology.org/N18-1101/.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785. URL https://doi.org/10.1145/1498765.1498785.

Yichen Xu, Yuyang Liang, Shan Dai, Tianyang Hu, Tsz Nam Chan, and Chenhao Ma. Understanding and guiding layer placement in parameter-efficient fine-tuning of large language models, 2026. URL https://arxiv.org/abs/2602.04019.

Dongshuo Yin, Xueting Han, Bin Li, Hao Feng, and Jing Bai. Parameter-efficient is not sufficient: Exploring parameter, memory, and time efficient adapter tuning for dense predictions. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 1398–1406. Association for Computing Machinery, 2024. doi: 10.1145/3664647.3680940. URL https://doi.org/10.1145/3664647.3680940.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, Lucas Beyer, Olivier Bachem, Michael Tschannen, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, and Neil Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark, 2019. URL https://arxiv.org/abs/1910.04867.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning, 2023a. URL https://arxiv.org/abs/2308.03303v1.

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

A LaLoS Algorithm

Algorithm 1 specifies the binary search in Section 4 and Figure 2. The model has L eligible projection layers for low-rank adaptation (LoRA) (Hu et al., 2022), indexed $b = 0, \ldots, L-1$ from the output toward the input. Support S lists the selected layer indices, and $K = |S|$ is the number of LoRA layers. Placement $S_b = \{0, \ldots, b\}$ extends from layer 0 through layer b. For nonempty S_b, the last selected index is $b_{\max}(S_b) = b$ and the layer count is $K = |S_b| = b + 1$. Smaller values of b select fewer layers from the output end, while larger values include more layers toward the input. The empty placement is $S_{-1} = \varnothing$. The fixed list $\mathcal{E}$ contains the allowed placement indices in ascending order, including -1 and $L-1$.

The probe is the model frozen after initial full-support training (warm-up). A query enables only the LoRA adapters in S and uses forward passes on held-out selection data to obtain $\mathcal{A}(S)$ (higher is better). Each queried placement is evaluated once, with stored results reused thereafter. A queried placement passes (is feasible) if $\mathcal{A}(S) \geq \mathcal{A}(S_{L-1}) - \epsilon$, and fails otherwise. Here $\epsilon \geq 0$ is the allowed score drop, and the right-hand side is the score threshold. $b^\star$ denotes the smallest queried index with a passing placement. After binary search and the neighboring-placement checks, every nonempty selection triggers one restart with lower bound $b_{\text{lo}} = -1$ and upper bound $b_{\text{hi}} = b^\star$. Binary search uses the larger index when there are two middle entries. When scores are nonmonotonic, an unqueried smaller placement may still pass. $\widehat{S}$ is the final support after individual LoRA layers are chosen. Appendices C and D describe the benchmark search configurations and resource measurements.

Algorithm 1 LaLoS and LoRA fine-tuning

Input: pretrained checkpoint, sorted candidate index list $\mathcal{E}$, disjoint adapter-training and selection subsets, final update count T, warm-up fraction ρ, and score drop ϵ.

1 Full-support LoRA trains on the adapter-training subset for $\lceil \rho T \rceil$ updates, rounding any fractional update up. The checkpoint is frozen in evaluation mode.

2 Full support S_{L-1} is queried, followed by empty support. Empty support disables backbone adapters but retains the probe's task head.

3 If the empty placement passes, $b^\star = -1$, $\widehat{S} = \varnothing$, and steps 4–10 are skipped.

4 The bounds are $b_{\text{lo}} = -1$ and $b_{\text{hi}} = L - 1$, with failing and passing placements, respectively.

5 While candidates remain between the bounds, binary search tests the middle placement S_b. A pass sets $b_{\text{hi}} = b$. Otherwise, $b_{\text{lo}} = b$.

6 After step 5 ends, the complete placement S_b at the nearest candidate in $\mathcal{E}$ below b_{lo} is checked against the score threshold. The check is skipped if no such candidate exists.

7 The complete placement S_b at the nearest candidate in $\mathcal{E}$ above b_{hi} is checked against the score threshold. The check is skipped if no such candidate exists.

8 After these checks, the smallest passing index queried so far becomes $b^\star$, the upper bound for the single restart.

9 The search restarts once with new bounds $b_{\text{lo}} = -1$ and $b_{\text{hi}} = b^\star$. The loop in step 5 runs again to completion, reusing stored scores. Its final upper bound becomes $b^\star$.

10 The index $b = \max\{b \in \mathcal{E} : b < b^\star\}$ immediately precedes $b^\star$ in $\mathcal{E}$. The returned placement retains all layers in S_b and uses the fewest LoRA layers meeting the score threshold: $\widehat{S} \in \arg\min_{\substack{S_b \subseteq S \subseteq S_{b^\star} \\ S \text{ feasible}}} |S|$.

11 The probe is discarded and the original checkpoint reloaded. LoRA factors are reinitialized only in $\widehat{S}$. Final fine-tuning runs for all T updates.

Output: final model, $\widehat{S}$, stored query results, and separate warm-up, query, fine-tuning, and total cost records.

B Placement-Dependent Backward Computation

A projection layer is a linear map. Low-rank adaptation (LoRA) trains low-rank factors while keeping the pretrained weights frozen. The model's L eligible projections have fixed indices $b = 0, \ldots, L-1$. These indices run from the output toward the input, with layer 0 nearest the output. In self-attention, the query, key, and value projections form parallel branches that receive the same input. Their indices identify layers for selection.

An input gradient is the derivative of the loss with respect to a projection's input. When several branches share that input, its gradient is the sum of the contributions from all branches that use it. This includes branches with pretrained weights only. The shared-input gradient is needed when it carries

the loss gradient to trainable parameters closer to the model input. Each branch's matrix-product cost depends on its own input and output dimensions. For a branch containing LoRA, the cost also depends on that branch's rank.

B.1 Projection dimensions and gradients

Only adapters and the task head are trainable in this calculation. Gradients with respect to the model input are not computed. To isolate the effect of placement, the local calculation assumes common widths $d_{\text{in}}, d_{\text{out}}$ and LoRA rank r across placements. It also assumes that dropout is disabled. N is the number of token positions per physical minibatch, including padding. Tensors carry layer index b. Inputs $X_b \in \mathbb{R}^{d_{\text{in}} \times N}$ have one token position per column. The trainable factor A_b has dimensions $r \times d_{\text{in}}$. The other factor, B_b, has dimensions $d_{\text{out}} \times r$. Their product is scaled by α/r, where α is the LoRA scaling coefficient. For scalar training loss $\mathcal{L}$, $G_b \in \mathbb{R}^{d_{\text{out}} \times N}$ is the loss gradient at projection output Y_b. It arrives through backpropagation from operations closer to the model output. Matrix transposition is denoted by $^\top$. The LoRA parameter gradients are given in Eq. (2). For directly connected projections, Eq. (3) describes how the received gradient G_b is computed.

B.2 Local computation and logical memory traffic

For each modeled projection, $\mathbf{c}_{\text{base}}$ gives the cost of the input gradient through the frozen pretrained weights. $\mathbf{c}_{\text{param}}$ gives the cost of the LoRA parameter gradients. $\mathbf{c}_{\text{LoRA}}$ gives the additional cost of the input gradient through LoRA. Each vector lists floating-point operations (FLOPs), logical read bytes, and logical write bytes in that order. Each multiply–add costs two FLOPs (NVIDIA, 2023). The logical traffic calculation assumes s bytes per scalar. For each matrix product, every operand element is read once and every result element is written once. Costs are combined by adding the corresponding entries of their vectors. Dividing a byte volume by s gives the number of scalar elements read or written. Costs exclude scaling, dropout, gradient summation, and optimizer updates. GPU memory transactions additionally depend on fusion, tiling, caching, and checkpointing. Runtime depends on memory bandwidth and attainable arithmetic throughput (Williams et al., 2009).

$W_{0,b} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ contains projection b's frozen weights. The subscript 0 denotes the pretrained component. $G_{\text{base},b}$ is the input-gradient contribution through those weights. $G_{\text{LoRA},b}$ is the contribution through the LoRA factors. The latter exists only when the projection contains LoRA. Computing $G_{\text{base},b}$ forms Nd_{in} dot products of length d_{out}. These read $d_{\text{in}}d_{\text{out}}$ weight and Nd_{out} gradient scalars and write Nd_{in} results. The parameter-gradient calculation reuses A_bX_b from the forward pass. It first forms $B_b^\top G_b$ at a cost of $2Nrd_{\text{out}}$ FLOPs. The product $(B_b^\top G_b)X_b^\top$ then requires $2Nrd_{\text{in}}$ FLOPs. The other parameter-gradient product, $G_b(A_bX_b)^\top$, requires $2Nrd_{\text{out}}$ FLOPs. Computing $G_{\text{LoRA},b}$ reuses the previously formed $B_b^\top G_b$. The shared product is included in $\mathbf{c}_{\text{param}}$, so $\mathbf{c}_{\text{LoRA}}$ charges only its multiplication by $A_b^\top$. These products give

$$
\begin{aligned}
\mathbf{c}_{\text{base}} &= \big(2Nd_{\text{in}}d_{\text{out}}, \quad sd_{\text{out}}(d_{\text{in}} + N), \quad sNd_{\text{in}}\big), \\
\mathbf{c}_{\text{param}} &= \big(2Nr(2d_{\text{out}} + d_{\text{in}}), \quad s[rd_{\text{out}} + N(2d_{\text{out}} + 2r + d_{\text{in}})], \quad sr(N + d_{\text{in}} + d_{\text{out}})\big), \\
\mathbf{c}_{\text{LoRA}} &= \big(2Nrd_{\text{in}}, \quad sr(d_{\text{in}} + N), \quad sNd_{\text{in}}\big).
\end{aligned}
$$

The selected layer indices form support S. Its size $K = |S|$ is the number of LoRA layers. For nonempty S, $b_{\max}(S) = \max S$ is the index of the LoRA layer farthest from the output. We write $b_{\max}$ for a fixed S. $b_{\max}(S)$ is undefined for empty support. Table 3 applies Eq. (5) to the three $K = 4$ placements at matched N, widths, and rank. Moving from case (a) to case (c) removes seven pretrained-weight input-gradient products without changing the LoRA gradient terms. The reduction is $7\mathbf{c}_{\text{base}}$, equivalent to $14Nd_{\text{in}}d_{\text{out}}$ FLOPs, $7sd_{\text{out}}(d_{\text{in}} + N)$ read bytes, and $7sNd_{\text{in}}$ write bytes.

Table 3: Placement-dependent backward computation and logical memory traffic for the three $K = 4$ cases in Figure 1.

Case	Layers with LoRA	Farthest layer index $b_{\max}$	$\mathbf{c}_{\text{path}}(S)$
(a)	$\{0, 3, 7, 10\}$	10	$10\mathbf{c}_{\text{base}} + 4\mathbf{c}_{\text{param}} + 3\mathbf{c}_{\text{LoRA}}$
(b)	$\{0, 2, 5, 7\}$	7	$7\mathbf{c}_{\text{base}} + 4\mathbf{c}_{\text{param}} + 3\mathbf{c}_{\text{LoRA}}$
(c)	$\{0, 1, 2, 3\}$	3	$3\mathbf{c}_{\text{base}} + 4\mathbf{c}_{\text{param}} + 3\mathbf{c}_{\text{LoRA}}$

B.3 Backward-cost graph construction

Model and inputs. The right-hand graph in Figure 1 gives the analytical backward cost of the serial chain in Section 3. The chain has $L = 12$ layers indexed $b = 0, \ldots, 11$ from output to input. The $K = 4$ selected LoRA layers have a common rank r. All layers have widths $d_{\text{in}} = d_{\text{out}} = 768$. The illustrative physical minibatch contains 32 sequences padded to 512 tokens each. This gives $N = 32 \times 512 = 16384$ token positions per forward/backward microbatch, before gradient accumulation. The resulting tensor shapes are

$$W_{0,b} \in \mathbb{R}^{768\times768}, \qquad A_b \in \mathbb{R}^{r\times768}, \qquad B_b \in \mathbb{R}^{768\times r},$$
$$X_b \in \mathbb{R}^{768\times16384}, \qquad G_b \in \mathbb{R}^{768\times16384}.$$

The graph uses the main-text trainability assumptions and the caching and arithmetic conventions above. Dropout and checkpoint recomputation are excluded. It excludes forward, task-head, attention, nonlinear, elementwise, and optimizer computation. The elementwise exclusion includes LoRA scaling by α/r. The bytes-per-scalar value s determines the logical memory traffic.

From gradient products to plotted cost. One GFLOP is 10^9 FLOPs. The local FLOP components per physical minibatch are given below in GFLOPs. All outputs are computed at full precision before being rounded to at most four decimal places for display.

$$\mathbf{c}_{\text{base}}: \quad \frac{2 \times 16384 \times 768^2}{10^9} \approx 19.3274,$$
$$\mathbf{c}_{\text{param}}: \quad \frac{2 \times 16384 \times r \times (2 \times 768 + 768)}{10^9} \approx 0.0755r,$$
$$\mathbf{c}_{\text{LoRA}}: \quad \frac{2 \times 16384 \times r \times 768}{10^9} \approx 0.0252r.$$

$\mathbf{c}_{\text{path}}(S)$ is the projections' backward subtotal, excluding intervening operations and the task head. Substitution into Eq. (5) with $K = 4$ gives the plotted cost:

$$\begin{aligned}\text{Backward GFLOPs} &= \frac{2 \times 16384 \times 768}{10^9}\big(768b_{\max} + 4(3r) + 3r\big) \\ &= \frac{2 \times 16384 \times 768}{10^9}(768b_{\max} + 15r).\end{aligned}$$

Curves and worked examples. Increasing the farthest layer index $b_{\max}$ by one adds approximately 19.3274 GFLOPs at fixed rank. A unit rank increment adds approximately 0.3775 GFLOPs at fixed placement. The colored curves vary rank over $r = 1, \ldots, 64$ on the bottom axis. Case (c) uses the fixed layer set $\{0, 1, 2, 3\}$, whose farthest layer index is $b_{\max} = 3$. Case (b) uses $\{0, 2, 5, 7\}$ and has $b_{\max} = 7$. Case (a) uses $\{0, 3, 7, 10\}$ and has $b_{\max} = 10$. The black dotted curve fixes the rank at $r = 8$. It varies $b_{\max} = 3, \ldots, 11$ on the top axis using the representative support $\{0, 1, 2, b_{\max}\}$. Four distinct selected layers require $b_{\max} \geq 3$. Other four-layer supports with the same $b_{\max}$ have the same modeled cost. The two horizontal axes measure different quantities and use independent scales.

For $b_{\max} = 7$ and $r = 8$, each LoRA parameter-gradient term costs approximately 0.6040 GFLOPs. Each additional LoRA input-gradient term costs approximately 0.2013 GFLOPs. The plotted subtotal is

$$\frac{2 \times 16384 \times 768}{10^9}(768 \times 7 + 15 \times 8) \approx 138.3114 \text{ GFLOPs}.$$

The trainable LoRA parameter total $P_{\mathrm{LoRA}}(S)$ excludes the separately trained task head. For these dimensions, Eq. (1) gives $P_{\mathrm{LoRA}}(S) = 4r(768 + 768) = 6144r$ trainable LoRA parameters. Table 4 gives representative inputs and outputs. Changing $b_{\max}$ at fixed rank preserves this budget. Increasing rank changes capacity along a colored curve. The black curve retains 49,152 parameters. Empirical resource measurements are reported separately in Appendix D.

Table 4: Example inputs and analytical outputs for the graph, with the shared network and minibatch dimensions specified above.

Farthest layer index $b_{\max}$	Rank r	Backward GFLOPs	LoRA parameters
3	8	61.0020	49,152
7	8	138.3114	49,152
10	8	196.2934	49,152
11	8	215.6208	49,152
7	16	141.3313	98,304
7	64	159.4507	393,216

C Benchmark Search Configuration

Section 4 and Algorithm 1 specify the search procedure. This appendix gives the benchmark-specific candidate placements. The General Language Understanding Evaluation (GLUE) setup has $L = 36$ eligible projection layers. The allowed nonempty indices are $b = 2, 5, \ldots, 35$, with $b = -1$ for empty support.

VTAB configuration. The Visual Task Adaptation Benchmark (VTAB) setup has $L = 160$ eligible projection layers. The allowed indices are $b = -1, 0, \ldots, 159$. The reported main vision runs evaluate the larger tolerance first. The smaller tolerance reuses the frozen probe and available query scores. Appendix G provides the per-task VTAB results.

D Computation and Systems Measurements

D.1 Measurement scope and normalization

The main comparison reports final fine-tuning resources: F_{ft} for supported forward and backward FLOPs, T_{ft} for training wall time, and M_{ft} for peak allocated GPU memory. Selection is measured separately, distinguishing the cost of choosing an adapter placement from the cost of training the selected configuration.

For benchmark tasks $d \in \mathcal{D}$, methods m, optimization seeds $i \in \mathcal{I}$, and resource c, let c_{dmi} be the method's value and $U_{di}(c)$ the corresponding same-task/seed Full-LoRA value. The benchmark ratio is

$$R_m(c, U) = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \frac{c_{dmi}}{U_{di}(c)}. \tag{8}$$

Ratios are formed before task/seed averaging and rounding, using the same resource and measurement procedure within each benchmark. The main table reports FLOP and time savings as $100[1 - R_m(c, U)]\%$, with $c = F_{\mathrm{ft}}$ and $c = T_{\mathrm{ft}}$, respectively. M_{ft} is the mean of per-run peaks in GiB. The Full-LoRA reference measures final fine-tuning alone.

D.2 FLOP counter implementations and supported operations

FLOPs cover model operations supported by the PyTorch instrumentation (Paszke et al., 2019). Elapsed time includes optimizer and allocator arithmetic. One PFLOP denotes 10^{15} operations. For GLUE, `torch.profiler` measures supported forward/backward FLOPs for a physical micro-batch, the examples processed together in one pass. This count is projected over the recorded training schedule. Probe training is counted separately. Placement-query FLOPs use the full-support probe's

forward profile multiplied by the number of selection minibatches and unique queries. A common full-support forward-cost estimate is used across queried masks.

VTAB. Registered FLOP formulas accumulate supported operations during execution, including dense matrix products, convolutions, and fused attention and its backward operator. Counts retain physical batch shapes and executed mixed-precision retries. Layer normalization, the sigmoid linear unit (SiLU), losses, elementwise operations, gradient scaling, and optimizer updates are outside this counter's coverage. Reporting evaluations are recorded separately.

D.3 Adapter placement and allocated trainable parameters

Saved projection identities define the support S. K counts LoRA-adapted projections. For Hybrid SPT, it counts structured adapters. K_A counts projections with any structured or unstructured update. Their union determines where adaptation occurs. On GLUE, this union contains all 36 eligible projections.

P_A counts allocated backbone adaptation parameters in millions, excluding the task head and temporary selector gates. For Hybrid SPT, $P_A = P_s + P_u$, where P_s and P_u count structured and unstructured parameters. A rank-eight adapter contains 12,288 factor parameters in DeBERTa and 24,576 in DINOv2.

Physical rank describes allocated factor-tensor dimensions. Logical rank describes active, unmasked components. Their sums are R_p and R_l. The evaluated AdaLoRA and IGU-LoRA integrations maintain physical rank sums of 432 on GLUE and 1920 on VTAB, targeting logical rank sums of 288 and 1280, respectively. Their main-table K counts projections with nonzero final logical rank, whereas P_A counts the maintained tensors, including singular-value vectors. The tensor allocation stays fixed as logical components are masked.

Table 1 reports the mean last selected projection index in output-first order, with zero nearest the output. For LoRA placements, $b_{\max} = \max S$ is undefined for empty support. Each seed averages indices over its nonempty task placements, followed by averaging the five seed means. For vision, LaLoS-0.5 and LaLoS-1.0 have 92 and 88 nonempty outcomes out of 95. All other reported quantities use all runs. Hybrid SPT's last index includes both structured and unstructured updates, while its K counts structured LoRA layers. Adaptive-rank methods use positive final logical ranks.

E Experimental Protocols and Reproducibility

E.1 GLUE: controlled language adaptation

We evaluate DeBERTaV3-base (He et al., 2023b) on seven tasks from the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018): SST-2, QNLI, CoLA, RTE, MRPC, QQP, and MNLI. SST-2 is the binary Stanford Sentiment Treebank (Socher et al., 2013), and CoLA is the Corpus of Linguistic Acceptability (Warstadt et al., 2019). Question-answering natural language inference (QNLI) is GLUE's classification conversion of the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016, Wang et al., 2018). MRPC is the Microsoft Research Paraphrase Corpus (Dolan & Brockett, 2005). MNLI is the Multi-Genre Natural Language Inference corpus (Williams et al., 2018). Recognizing Textual Entailment (RTE) and Quora Question Pairs (QQP) use the versions distributed with GLUE (Wang et al., 2018).

Following the DeBERTaV3-base GLUE training protocol used by WeightLoRA (Veprikov et al., 2026), we apply rank-eight LoRA to the 36 self-attention query, key, and value projections and use the same task-wise training horizons, effective batch size, and numerical precision. Optimizer coefficients, weight decay, learning-rate schedule, scheduler warm-up, and maximum sequence length also follow that protocol. Concretely, training uses bfloat16 (BF16), an effective batch size of 64, a maximum sequence length of 512, AdamW with $(\beta_1, \beta_2) = (0.9, 0.99)$ and numerical epsilon 10^{-8}, weight decay 10^{-4}, and a linear schedule with 100 learning-rate warm-up updates. We use 7 epochs for MNLI, 10 for SST-2, 15 for CoLA, 5 for QQP and QNLI, 30 for MRPC, and 50 for RTE. The physical batch size is 32 with two gradient-accumulation steps for MNLI, SST-2, QQP, and QNLI, and 64 without accumulation for CoLA, MRPC, and RTE. All controlled methods use a fixed learning rate of 5×10^{-5}.

LaLoS adds a data partition and a placement-selection stage. Using a fixed split seed of 2027, each task's training set is divided into an adapter-training subset and a disjoint held-out selection subset. The full-support probe is trained on the adapter-training subset for $\lceil 0.1T \rceil$ optimizer updates, where T is the corresponding final fine-tuning schedule, and placement is selected on the held-out subset. Final fine-tuning then reloads the original pretrained checkpoint, reinitializes the selected LoRA factors, and uses the complete training set. The two reported configurations allow score drops of 0.5 and 1.0 percentage points from the full-support probe and are denoted LaLoS-0.5 and LaLoS-1.0, respectively. Probe training is distinct from the 100-update learning-rate warm-up above.

Performance is evaluated on the official GLUE development splits (Wang et al., 2018). CoLA uses Matthews correlation. MRPC and QQP use the mean of accuracy and F1. MNLI uses the mean of matched and mismatched development accuracy. SST-2, QNLI, and RTE use accuracy. Scores are reported on a 0–100 scale, and GLUE-7 is the unweighted mean of these seven task scores.

E.2 VTAB-1K: TRANSFER TO VISION

DINOv2-G/14 (Oquab et al., 2024) has 160 eligible projection layers: forty each of query, key, value, and attention output. The 19 VTAB-1k tasks comprise seven Natural, four Specialized, and eight Structured tasks (Zhai et al., 2019). Appendix G lists the tasks and groups. The dSprites tasks Shape position and Shape orientation predict a synthetic shape's horizontal location and rotation, respectively. Backbone weights, biases, and normalization parameters are frozen, while the task classifier is trained.

Probe training uses `train800`. Placement queries use the disjoint `val200`. Final fine-tuning uses all 1,000 examples in `train800val200`, followed by test reporting. Inputs are 224×224 pixels. AdamW (Loshchilov & Hutter, 2019) uses a learning rate of 10^{-3}, cosine decay to zero, weight decay 10^{-4}, coefficients $(0.9, 0.999)$, numerical epsilon 10^{-8}, and gradient clipping at norm 1. A physical batch size of eight and eight gradient-accumulation steps give an effective batch size of 64. Partial epoch-end windows retain division by eight. There is no augmentation, mixup, cutmix, or label smoothing.

The main probe runs for three epochs (300 microbatches, 39 optimizer updates). Final fine-tuning runs for thirty epochs (3,750 successful microbatches, 480 updates). The score tolerances match GLUE. Execution uses H200 GPUs, 16-bit floating-point (FP16) autocast with gradient scaling, PyTorch 2.6.0+cu124, torchvision 0.21.0+cu124, CUDA 12.4, and disabled TensorFloat-32 (TF32). On gradient-scaling overflow, the same data window and random-number state are retried. Executed work follows Appendix D. Reporting uses a physical batch size of sixteen and the true final remainder.

E.3 CONTROLLED METHOD INTEGRATIONS

Fixed-rank placement and WeightLoRA. Last-K retains the K eligible projections nearest the output, with indices $b = 0, \ldots, K-1$ for $K > 0$. The baseline uses half of the eligible layers, with $K = 18$ on GLUE and $K = 80$ on VTAB. Implementation order is query/key/value in DeBERTa and query/key/value/output in DINOv2, traversed from input to output. Selection ties retain this declared implementation order. The manuscript describes placements using output-first indices. WeightLoRA (Veprikov et al., 2026) learns module gates, then reinitializes standard LoRA factors at the selected projections. Its vision warm-up is five updates, or forty microbatches.

PLoP. Precise LoRA Placement (PLoP) Type1 and Type2 (Hayou et al., 2026) select one or two projection types across the backbone, giving 12/24 adapters in GLUE and 40/80 in vision. Vision calibration uses 100 seeded training examples, a physical batch size of eight, and one Gaussian reference per hook invocation. The normalized feature-norm ratios are averaged over minibatches and the forty instances of each type. The variants share calibration and fine-tune separately with reinitialized factors.

Hybrid SPT. Hybrid sensitivity-aware visual parameter-efficient fine-tuning (SPT) integrations (He et al., 2023a) retain structured LoRA factors and unstructured trainable coordinates. Vision sensitivity accumulates squared loss gradients over 512 seeded examples in sixteen batches of 32, without optimizer updates. Structured conversion uses a factor of ten and a 400,000-parameter target. The sweep tests increasing prefixes of sensitivity-ranked weight coordinates. Prefix size increases by

20,000 up to 1,580,000. Coordinate indices resolve equal sensitivity scores. Realized allocations range from 338,671 to 460,239 parameters.

AdaLoRA and IGU-LoRA. The controlled integrations (Zhang et al., 2023b, Cui et al., 2026) maintain a physical rank of twelve per candidate projection and target the Full-LoRA-r8 logical rank budget. Table 5 gives the GLUE schedules. Vision uses $T = 480$, an initial fixed-rank interval of 80, a final fixed-budget interval of 240, and an allocation interval of two. The orthogonality weight is 0.1. IGU-LoRA samples seeded paths over $\{1/20, \ldots, 20/20\}$. Allocation occurs within final training. These controlled integrations use shared benchmark training settings. Method-specific hyperparameters retain the specified configurations.

Table 5: Executed GLUE online-rank schedules, in optimizer updates. t_{init} and t_{final} are the initial fixed-rank and final fixed-budget intervals. ΔT is the allocation interval.

Task	T	t_{init}	t_{final}	ΔT
SST-2	10530	1251	4586	21
QNLI	8185	1000	4000	50
CoLA	2010	240	1050	3
RTE	1950	300	900	1
MRPC	1740	303	908	1
QQP	28430	4000	12501	50
MNLI	42952	4000	25000	50

F Complete GLUE Results

Table 6 gives all seven development-task scores for the primary methods and the Equal-Span Dense Support (ESDS) control. Each cell is the mean and sample standard deviation (SD) over five optimization seeds, computed from the recorded task/seed scores. The task metrics follow Appendix E.1.

For task score a_{dmi}, the seed-level benchmark score a_{mi}, benchmark mean $\bar{a}_m$, and sample standard deviation are

$$a_{mi} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} a_{dmi}, \qquad \bar{a}_m = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} a_{mi},$$
$$\mathrm{SD}_m^2 = \frac{1}{|\mathcal{I}| - 1} \sum_{i \in \mathcal{I}} (a_{mi} - \bar{a}_m)^2. \tag{9}$$

VTAB uses the same aggregation rule. Per-task SD describes variation across seeds on one task. Benchmark SD is computed after equally averaging tasks within each seed.

ESDS-18 retains complete query/key/value triples from the farthest adapted WeightLoRA-18 block to the output. Its suffix identifies the source WeightLoRA budget. The main benchmark table reports final-training resources. Appendix D gives their normalization.

Table 6: Complete GLUE development scores (scale of 100). Each cell shows the mean above its ± sample SD over five seeds. Task-specific metrics are defined in Appendix E.1.

Method	SST-2	QNLI	CoLA	RTE	MRPC	QQP	MNLI
Head-only	79.50% ±0.68%	72.18% ±0.18%	47.47 ±0.33	63.25% ±0.47%	74.80 ±0.00	77.51 ±0.08	65.81% ±0.05%
Full LoRA	95.14% ±0.25%	92.61% ±0.26%	64.27 ±0.70	82.45% ±0.60%	89.62 ±0.35	87.85 ±0.11	90.00% ±0.05%
Last-18	95.46% ±0.19%	92.46% ±0.15%	63.68 ±0.59	83.39% ±1.69%	90.57 ±0.75	87.17 ±0.07	89.38% ±0.05%
WeightLoRA-18	95.60% ±0.44%	92.26% ±0.32%	63.28 ±0.94	81.95% ±3.07%	89.88 ±0.53	87.26 ±0.09	88.72% ±2.01%
ESDS-18	95.28% ±0.24%	92.65% ±0.30%	63.98 ±0.77	82.31% ±0.81%	89.54 ±0.46	87.86 ±0.08	90.01% ±0.04%
PLoP-Type1	94.11% ±0.19%	91.32% ±0.29%	62.28 ±0.42	75.96% ±1.32%	88.36 ±0.24	86.71 ±0.09	89.10% ±0.15%
PLoP-Type2	94.79% ±0.28%	92.65% ±0.10%	63.79 ±1.23	81.23% ±1.82%	90.06 ±0.40	87.58 ±0.08	89.76% ±0.06%
Hybrid SPT	95.28% ±0.34%	92.68% ±0.17%	64.07 ±0.88	82.38% ±0.78%	89.79 ±0.41	87.68 ±0.09	90.00% ±0.14%
AdaLoRA	93.78% ±0.21%	89.19% ±0.11%	51.58 ±0.35	63.75% ±0.98%	74.64 ±0.09	85.63 ±0.07	88.91% ±0.03%
IGU-LoRA	88.35% ±2.59%	77.65% ±1.00%	47.75 ±0.25	63.75% ±1.07%	74.80 ±0.00	81.91 ±0.16	85.14% ±0.23%
LaLoS-0.5	95.62% ±0.22%	92.63% ±0.15%	64.54 ±0.92	82.09% ±2.00%	89.56 ±0.74	87.54 ±0.11	89.68% ±0.06%
LaLoS-1.0	95.64% ±0.34%	92.41% ±0.10%	64.93 ±1.03	82.67% ±2.34%	90.61 ±0.66	87.17 ±0.10	89.38% ±0.07%

G COMPLETE VTAB-1K RESULTS

Tables 7 and 8 report all nineteen VTAB-1k tasks for the eleven primary methods. Tasks run down the page, grouped as Natural, Specialized, and Structured. The two tables show different methods on the same comparison population. Each entry reports mean accuracy and sample SD across five optimization seeds. The benchmark equally averages the nineteen task scores within each seed, as in Appendix F.

Table 7: VTAB-1k test accuracy (%): reference placements, WeightLoRA, and LaLoS. Each cell shows the mean above its ± sample SD over five seeds.

Task	Head-only	Full LoRA	Last-80	WeightLoRA -80	LaLoS -0.5	LaLoS -1.0
Natural						
Caltech101	89.14% ±0.27%	89.03% ±0.45%	89.02% ±0.60%	88.84% ±0.55%	90.27% ±0.35%	90.05% ±0.46%
CIFAR-100	77.25% ±0.06%	84.30% ±0.67%	84.90% ±0.19%	85.10% ±0.24%	84.75% ±0.76%	84.84% ±0.41%
DTD	78.22% ±0.16%	82.07% ±0.49%	81.95% ±0.21%	82.10% ±0.42%	81.97% ±0.46%	81.80% ±0.51%
Flowers102	99.69% ±0.00%	99.70% ±0.01%	99.71% ±0.00%	99.70% ±0.01%	99.69% ±0.01%	99.69% ±0.01%
Oxford Pets	95.09% ±0.09%	95.28% ±0.23%	95.04% ±0.14%	95.26% ±0.13%	95.42% ±0.17%	95.28% ±0.22%
SUN397	60.27% ±0.09%	62.40% ±0.28%	62.13% ±0.48%	61.89% ±0.13%	61.64% ±0.29%	61.32% ±0.44%
SVHN	40.59% ±0.05%	93.20% ±0.42%	90.28% ±0.76%	91.13% ±1.49%	92.53% ±0.74%	92.58% ±0.61%
Specialized						
Patch Camelyon	84.35% ±0.23%	87.18% ±1.32%	87.78% ±0.94%	87.05% ±0.97%	87.57% ±0.82%	87.62% ±0.73%
EuroSAT	93.81% ±0.07%	96.33% ±0.12%	96.18% ±0.36%	96.51% ±0.22%	96.17% ±0.13%	96.14% ±0.31%
RESISC45	86.03% ±0.13%	91.57% ±0.49%	91.93% ±0.46%	91.47% ±0.66%	91.48% ±0.56%	91.33% ±0.51%
Retinopathy	76.32% ±0.04%	77.01% ±0.84%	77.16% ±0.46%	76.70% ±1.00%	76.53% ±0.88%	76.96% ±0.49%
Structured						
CLEVR count	46.58% ±0.16%	78.65% ±2.95%	93.50% ±0.76%	89.25% ±7.05%	90.28% ±4.64%	89.90% ±6.98%
CLEVR distance	37.21% ±0.15%	62.61% ±0.90%	63.13% ±1.11%	64.16% ±1.19%	63.31% ±0.83%	62.80% ±1.26%
Shape position	13.42% ±0.07%	86.86% ±7.73%	91.86% ±0.82%	90.87% ±3.02%	88.52% ±7.29%	88.54% ±7.30%
Shape orientation	38.64% ±0.04%	59.65% ±0.86%	63.47% ±0.92%	59.88% ±1.51%	61.43% ±1.29%	62.12% ±1.29%
SmallNORB azimuth	20.04% ±0.05%	34.20% ±0.34%	35.10% ±0.63%	34.06% ±1.17%	35.30% ±0.92%	34.88% ±1.77%
SmallNORB elevation	31.12% ±0.11%	42.80% ±2.12%	44.82% ±1.08%	44.63% ±1.66%	44.65% ±0.62%	44.84% ±1.59%
DMLab	47.16% ±0.07%	61.43% ±0.63%	62.17% ±0.42%	60.41% ±2.59%	61.57% ±0.26%	62.09% ±0.48%
KITTI distance	54.12% ±0.47%	83.66% ±1.11%	83.94% ±1.20%	83.21% ±1.15%	83.40% ±0.96%	83.43% ±0.67%

Table 8: VTAB-1k test accuracy (%): the remaining primary methods on the same nineteen tasks and five seeds. Each cell shows mean accuracy above its ± sample SD.

Task	PLoP Type1	PLoP Type2	Hybrid SPT	AdaLoRA	IGU-LoRA
Natural					
Caltech101	88.25% ±0.67%	88.87% ±0.74%	89.69% ±0.98%	89.16% ±0.31%	88.91% ±0.22%
CIFAR-100	84.59% ±0.38%	84.41% ±0.32%	84.86% ±0.33%	81.37% ±0.12%	81.58% ±0.27%
DTD	81.94% ±0.42%	81.81% ±0.61%	82.02% ±0.08%	80.70% ±0.12%	80.24% ±0.38%
Flowers102	99.70% ±0.01%	99.71% ±0.00%	99.70% ±0.01%	99.69% ±0.00%	99.69% ±0.00%
Oxford Pets	95.15% ±0.12%	95.22% ±0.18%	95.41% ±0.12%	95.12% ±0.17%	95.27% ±0.17%
SUN397	61.60% ±0.17%	61.97% ±0.27%	61.84% ±0.32%	60.54% ±0.12%	60.57% ±0.16%
SVHN	90.21% ±0.22%	91.50% ±0.64%	82.50% ±0.46%	83.39% ±0.52%	87.27% ±0.73%
Specialized					
Patch Camelyon	87.06% ±0.55%	87.05% ±0.65%	85.98% ±0.27%	86.81% ±0.24%	85.86% ±0.40%
EuroSAT	96.25% ±0.22%	96.34% ±0.18%	96.59% ±0.18%	96.11% ±0.04%	96.11% ±0.15%
RESISC45	91.44% ±0.13%	91.86% ±0.37%	91.59% ±0.22%	90.55% ±0.20%	90.63% ±0.32%
Retinopathy	76.22% ±0.63%	76.80% ±0.50%	75.20% ±0.46%	76.17% ±0.13%	75.35% ±0.98%
Structured					
CLEVR count	94.93% ±0.80%	93.21% ±2.49%	91.67% ±0.97%	91.59% ±0.65%	86.51% ±1.95%
CLEVR distance	65.15% ±0.86%	63.60% ±0.88%	60.14% ±0.92%	64.85% ±0.18%	61.02% ±0.31%
Shape position	76.46% ±13.86%	75.91% ±10.92%	89.03% ±1.05%	84.93% ±0.64%	85.30% ±1.72%
Shape orientation	60.19% ±0.79%	59.67% ±0.88%	59.32% ±0.51%	58.87% ±0.15%	61.60% ±0.53%
SmallNORB azimuth	33.79% ±0.66%	33.73% ±0.68%	34.85% ±0.50%	30.33% ±0.24%	32.23% ±0.78%
SmallNORB elevation	43.31% ±0.98%	43.06% ±1.28%	43.93% ±0.64%	41.34% ±0.34%	41.40% ±0.77%
DMLab	60.83% ±0.75%	61.49% ±0.67%	59.18% ±0.44%	59.00% ±0.42%	59.64% ±0.56%
KITTI distance	82.42% ±0.66%	84.19% ±0.55%	83.77% ±1.57%	83.97% ±0.48%	84.50% ±0.48%

G.1 SELECTED LAYERS AND FINAL FINE-TUNING RESOURCES

For task d and seed i, K_{di} counts selected LoRA projections. With $L = 36$ eligible projections for GLUE and $L = 160$ for VTAB, the percentage of selected LoRA layers is $p_{di} = 100K_{di}/L$. Table 9 reports task means and sample SDs for these percentages and the final fine-tuning resources defined in Section 5.3. Last-18 on GLUE and Last-80 on VTAB each retain 50% of the eligible adapters. LaLoS-1.0 retains 7.00% on Caltech101, 4.00% on Oxford Pets, 41.25% on EuroSAT, and 70.88% on SVHN. The selected extent is therefore task-dependent.

Table 9: Task-wise LoRA selection and final fine-tuning resources. Cells show the mean above its ± sample SD over five seeds. K/L is shown in percent, with $L = 36$ for GLUE and 160 for VTAB. F_{ft}, T_{ft}, and M_{ft} denote supported FLOPs, wall time, and peak allocated GPU memory. Warm-up and selection are excluded.

Task	LaLoS-0.5				LaLoS-1.0			
	K/L (%)	F_{ft} (PFLOP)	T_{ft} (min)	M_{ft} (GiB)	K/L (%)	F_{ft} (PFLOP)	T_{ft} (min)	M_{ft} (GiB)
Language: GLUE-7, DeBERTaV3-base								
SST-2	69.44 ±0.00	136.387 ±0.000	265.89 ±0.27	20.773 ±0.000	63.33 ±3.62	130.891 ±3.111	247.61 ±16.43	19.753 ±0.709
QNLI	66.11 ±6.92	103.208 ±4.749	201.94 ±19.79	20.266 ±1.239	57.22 ±5.05	97.517 ±3.162	178.34 ±19.48	18.734 ±1.059
CoLA	74.44 ±15.14	26.835 ±2.481	53.49 ±10.39	42.421 ±5.475	71.11 ±10.87	26.289 ±1.780	51.20 ±7.45	41.215 ±3.930
RTE	59.44 ±31.46	23.499 ±5.202	42.97 ±20.09	37.197 ±11.272	53.89 ±26.25	22.663 ±4.359	38.55 ±16.89	35.000 ±9.469
MRPC	58.33 ±18.11	20.904 ±2.649	37.03 ±9.49	36.781 ±6.194	41.11 ±15.27	18.368 ±2.327	27.18 ±8.10	30.567 ±5.353
QQP	71.67 ±5.69	370.495 ±16.706	749.77 ±45.02	21.191 ±1.022	51.67 ±7.24	326.490 ±16.397	554.24 ±81.42	17.630 ±1.395
MNLI	59.44 ±2.48	518.253 ±12.589	959.91 ±1.22	19.047 ±0.287	45.00 ±1.24	469.958 ±2.056	738.87 ±53.79	16.416 ±0.435
Vision: VTAB-19, DINOv2-G/14								
Caltech101	8.63 ±6.22	19.551 ±1.183	4.30 ±0.68	4.921 ±0.417	7.00 ±5.95	19.290 ±1.170	4.16 ±0.66	4.835 ±0.398
CIFAR-100	63.38 ±20.66	30.005 ±3.853	10.32 ±2.18	8.605 ±1.363	55.75 ±10.69	28.576 ±2.076	9.63 ±1.22	8.099 ±0.729
DTD	46.63 ±18.63	26.718 ±3.550	8.46 ±1.95	7.450 ±1.252	38.13 ±13.96	25.195 ±2.634	7.67 ±1.52	6.908 ±0.930
Flowers102	3.63 ±5.74	18.660 ±1.150	3.73 ±0.66	4.631 ±0.381	0.13 ±0.28	18.022 ±0.131	3.35 ±0.09	4.413 ±0.021
Oxford Pets	12.25 ±4.61	20.278 ±0.790	4.71 ±0.43	5.173 ±0.283	4.00 ±3.71	18.732 ±0.702	3.78 ±0.38	4.645 ±0.222
SUN397	21.00 ±15.76	21.975 ±2.986	5.61 ±1.74	5.779 ±1.053	14.50 ±11.43	20.749 ±2.136	4.98 ±1.28	5.346 ±0.755
SVHN	79.88 ±14.49	33.117 ±2.808	12.36 ±1.77	9.701 ±0.988	70.88 ±9.07	31.419 ±1.637	11.30 ±0.93	9.102 ±0.582
Patch Camelyon	44.75 ±7.07	26.568 ±1.326	8.23 ±0.84	7.384 ±0.469	44.00 ±8.05	26.380 ±1.536	8.11 ±0.91	7.320 ±0.541
EuroSAT	42.63 ±3.46	26.124 ±0.684	8.13 ±0.42	7.230 ±0.239	41.25 ±1.77	25.868 ±0.333	7.97 ±0.14	7.141 ±0.118
RESISC45	28.38 ±8.44	23.309 ±1.558	6.41 ±0.90	6.244 ±0.552	25.50 ±9.42	22.837 ±1.773	6.11 ±1.00	6.074 ±0.626
Retinopathy	58.13 ±17.37	29.034 ±3.332	9.78 ±1.95	8.258 ±1.174	47.63 ±6.32	26.987 ±1.160	8.62 ±0.79	7.539 ±0.412
CLEVR count	55.88 ±10.43	28.657 ±1.981	9.51 ±1.13	8.123 ±0.698	55.13 ±10.64	28.493 ±2.063	9.52 ±1.22	8.058 ±0.724
CLEVR distance	75.75 ±6.81	32.377 ±1.397	11.71 ±0.76	9.430 ±0.483	75.50 ±6.62	32.321 ±1.363	11.66 ±0.69	9.418 ±0.474
Shape position	93.13 ±15.37	35.507 ±2.790	13.61 ±1.82	10.542 ±0.986	93.00 ±15.65	35.427 ±2.970	13.60 ±1.82	10.517 ±1.043
Shape orientation	77.50 ±11.65	32.632 ±2.132	12.09 ±1.63	9.532 ±0.756	71.50 ±11.78	31.514 ±2.306	11.50 ±1.71	9.137 ±0.809
SmallNORB azimuth	46.88 ±17.48	26.892 ±3.303	8.39 ±1.93	7.504 ±1.166	38.75 ±22.55	25.356 ±4.359	7.55 ±2.48	6.961 ±1.533
SmallNORB elevation	62.13 ±16.74	29.722 ±3.249	10.20 ±2.23	8.505 ±1.142	54.38 ±8.03	28.320 ±1.542	9.35 ±1.05	8.008 ±0.543
DMLab	56.63 ±21.98	28.818 ±4.146	9.54 ±2.46	8.178 ±1.464	53.38 ±22.81	28.038 ±4.405	9.27 ±2.46	7.913 ±1.550
KITTI distance	32.25 ±7.19	24.077 ±1.430	6.83 ±0.76	6.513 ±0.501	31.63 ±7.13	24.049 ±1.404	6.85 ±0.74	6.498 ±0.492

Table 10: Sensitivity to warm-up budget and score tolerance with matched three-seed references. Tasks receive equal weight within each seed. Entries show the mean ± sample SD across the three resulting seed values. Accuracy is in percent. Baseline layer totals are fixed. Twelve configurations reuse matched Full-LoRA final results. Bold marks the reporting reference.

ρ	ϵ (pp)	Test accuracy (%)	Mean K
Full LoRA		$89.088\% \pm 0.331\%$	160
Last-80		$88.934\% \pm 0.087\%$	80
1%	0.5	$88.564\% \pm 0.495\%$	91.00 ± 56.46
1%	1.0	$88.152\% \pm 1.173\%$	85.83 ± 58.94
1%	1.5	$88.288\% \pm 1.295\%$	84.00 ± 57.07
1%	2.0	$88.170\% \pm 1.083\%$	77.50 ± 51.05
2%	0.5	$89.084\% \pm 0.355\%$	84.50 ± 19.94
2%	1.0	$88.931\% \pm 0.393\%$	73.00 ± 33.05
2%	1.5	$88.684\% \pm 0.507\%$	66.83 ± 35.91
2%	2.0	$88.926\% \pm 0.604\%$	62.50 ± 35.44
5%	0.5	$89.206\% \pm 0.400\%$	81.83 ± 22.54
5%	1.0	$88.936\% \pm 0.165\%$	71.33 ± 14.22
5%	1.5	$89.190\% \pm 0.415\%$	65.83 ± 12.58
5%	2.0	$89.085\% \pm 0.287\%$	60.83 ± 12.42
10%	0.5	$89.271\% \pm 0.394\%$	72.00 ± 16.02
10%	**1.0**	$88.992\% \pm 0.254\%$	58.67 ± 5.39
10%	1.5	$89.025\% \pm 0.213\%$	52.17 ± 4.93
10%	2.0	$88.871\% \pm 0.133\%$	43.67 ± 7.09

H Sensitivity to Warm-up Budget and Score Tolerance

This study evaluates Score-Constrained LoRA Search in the Last Layers (LaLoS) on EuroSAT and the Describable Textures Dataset (DTD). It examines how warm-up budget and score tolerance affect final accuracy and the number of LoRA layers.

Protocol and scope. The backbone, eligible layers (linear projections that may receive LoRA), and data partitions follow Appendix E.2. Both tasks use optimization seeds 1–3. LoRA rank (the width of each adapter's low-rank factors) and final fine-tuning follow Section 5.1. Probe training warms up LoRA at all eligible layers. The resulting checkpoint is then frozen and put in evaluation mode. Each query evaluates a placement represented by support S. Adapters outside S are masked for that query. For a fixed task, seed, and warm-up budget, all queries use the same frozen probe and held-out selection subset. Each query uses only forward passes through the frozen probe. Final fine-tuning starts from the original checkpoint with reinitialized LoRA factors. The warm-up budget ρ is a fraction of the final fine-tuning schedule of $T = 480$ optimizer updates. Budgets $\rho \in \{1\%, 2\%, 5\%, 10\%\}$ execute 5, 10, 24, and 48 probe updates. Their realized fractions are 1.0417%, 2.0833%, 5%, and 10%. The figure labels use the nominal fractions ρ. Each probe has its own cosine learning-rate schedule. Each tolerance has a separate query cache of evaluated placements and scores. Full support activates all eligible adapters. A placement is feasible when its probe accuracy is at least full-support accuracy minus the tolerance $\epsilon \geq 0$. Selection uses accuracy expressed as a fraction. The tolerances $\epsilon = 0.005, 0.010, 0.015, 0.020$ permit drops of 0.5, 1.0, 1.5, and 2.0 percentage points (pp), respectively. Tables and figures express tolerance in pp.

H.1 Final Accuracy

Paired mean accuracy differences from Full LoRA range from -0.404 to $+0.183$ pp at budgets of 2% and above. At a 1% budget, they range from -0.936 to -0.524 pp. Matched Full LoRA and Last-80 score $89.088 \pm 0.331\%$ and $88.934 \pm 0.087\%$, respectively.

Reporting reference. The reporting reference uses a 10% warm-up budget and a 1.0-pp tolerance. These settings follow the General Language Understanding Evaluation (GLUE) warm-up fraction

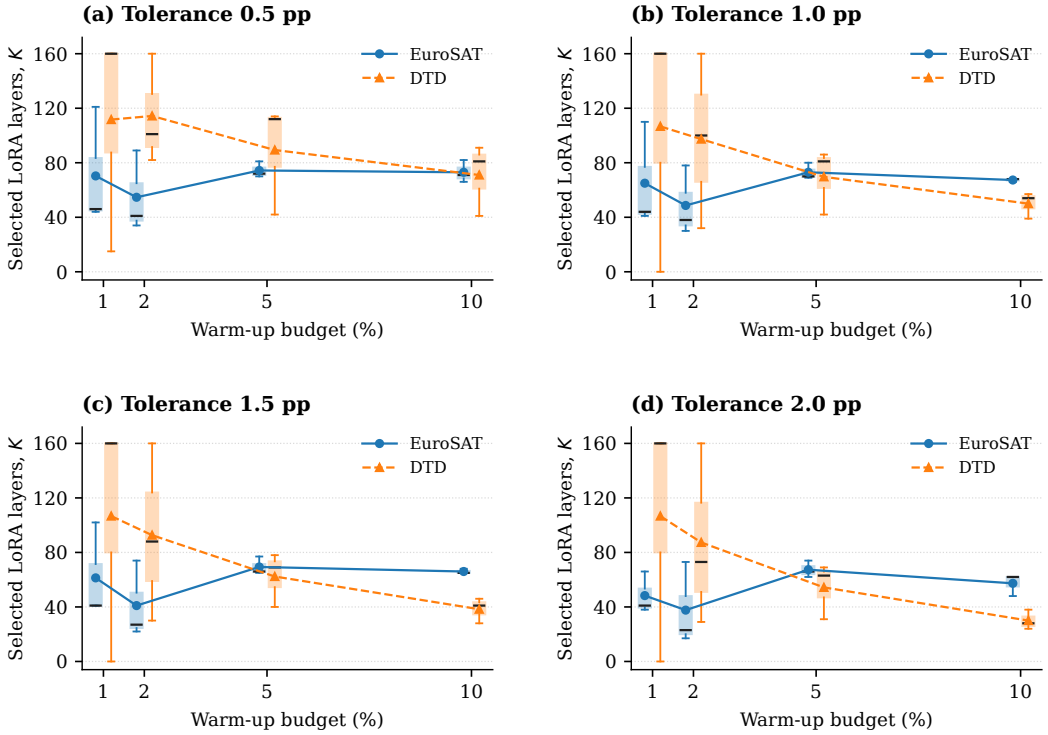

Figure 3: Number of selected LoRA layers at each tolerance. Each box summarizes three seeds for one task and warm-up budget. Boxes span the interquartile range, central lines mark medians, and whiskers show the minimum and maximum. Connected circles and triangles show mean K for EuroSAT and DTD, respectively. $K = 0$ and $K = 160$ denote Head-only and Full LoRA.

and tolerance convention. The reference achieves $88.992 \pm 0.254\%$ accuracy with mean $K = 58.67$. At the same warm-up budget, 1.5 pp yields $89.025 \pm 0.213\%$ and mean $K = 52.17$. The looser constraint selects fewer layers, with a paired mean accuracy difference of $+0.033 \pm 0.315$ pp. The 5%/1.5-pp setting achieves $89.190 \pm 0.415\%$ accuracy with mean $K = 65.83$. Its mean accuracy is $+0.256$ pp above that of Last-80.

H.2 Number of LoRA Layers

Mean K decreases with budget and tolerance in Table 10. However, increasing tolerance changes which placements are feasible for the same frozen probe. Increasing the budget instead changes the trained probe itself. Across the 72 comparisons between adjacent tolerance settings, K never increases. In contrast, K increases in 27 of the 72 comparisons between adjacent budgets. For example, DTD seed 2 at 0.5 pp selects $K = 42$ with a 5% warm-up budget and $K = 91$ with a 10% warm-up budget.

At 1%/1.0 pp, Figure 3 shows DTD selecting $K = 160, 160, 0$ and EuroSAT $110, 44, 41$. The pooled mean summarizes this mix of Full-LoRA, Head-only, and intermediate placements. At every tolerance, Full-LoRA references occur in two DTD seeds at 1% and one at 2%. All six task/seed cases at 5% and 10% use recorded search outcomes.

Task differences are clearer in selected layers than in small accuracy changes. At 1.0 pp, increasing the budget from 5% to 10% changes mean EuroSAT K from 73.00 to 67.33 and mean DTD K from 69.67 to 50.00. These reductions do not consistently improve final accuracy. Matched Full LoRA exceeds Head-only by 2.50 pp on EuroSAT and 3.76 pp on DTD. These gains are measured after final fine-tuning. Selection instead compares the full probe with the same probe with its adapters disabled.