

RAPA: Recursively Aligned Pathway Adaptation of Large Language Models

Sol Namkung^{1*}, Sunghyeon Woo^{1,3*}, Inho Jeong², Dongsuk Jeon^{1,2}

¹Interdisciplinary Program in Artificial Intelligence, Seoul National University

²Department of Intelligence and Information, Seoul National University

³NAVER Cloud

{nksol0405, wsh0917, djeon1}@snu.ac.kr

Abstract

Parameter-Efficient Fine-Tuning (PEFT) adapts large language models (LLMs) by training only a small fraction of parameters. Adapter-based approaches reduce the number of trainable parameters, but introduce practical overhead from the additional adapter path (e.g., extra kernel launches and activation storage). Adapter-free approaches avoid this structural overhead by directly updating pretrained weights; however, random index selection can fragment the trainable subspace, while calibrated weight selection incurs extra cost, limiting accuracy or efficiency. We propose **Recursively Aligned Pathway Adaptation (RAPA)**, an adapter-free PEFT method that forms index-consistent pathways through depth. RAPA follows two principles: (i) selecting balanced submatrices that maximize the number of weights alignable across layers, and (ii) recursively aligning these indices across layers and residual connections. In experiments, RAPA matches or surpasses strong PEFT baselines across most benchmarks while preserving adapter-free efficiency with minimal memory and compute overhead.

1 Introduction

Large Language Models (LLMs) now match or surpass expert-level performance in many natural language processing tasks (OpenAI et al., 2024; Team et al., 2024; Dubey et al., 2024; DeepSeek-AI et al., 2025; Jiang et al., 2024; Chowdhery et al., 2023). In particular, to specialize these LLMs for downstream applications, fine-tuning is commonly employed to adjust their parameters using task-specific data. However, fine-tuning these large-scale models for specific tasks demands substantial computational resources and memory (Brown et al., 2020; Zhao et al., 2024; Kaplan et al., 2020), posing significant challenges to practical applications.

*Sol Namkung and Sunghyeon Woo contributed equally to this work.

Code: <https://github.com/freeSoul-SNU/rapa>.

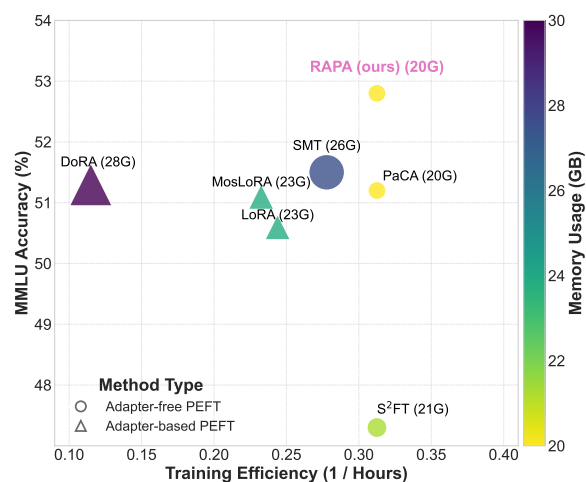


Figure 1: 5-shot MMLU performance comparison across representative PEFT methods. Numbers in parentheses indicate peak GPU memory usage (GB), which is also encoded by marker color and size. Methods are evaluated under comparable trainable-parameter budgets. The x-axis shows training efficiency, measured as the inverse of training time (1/hours).

To address these challenges, a wide range of Parameter-Efficient Fine-Tuning (PEFT) schemes have been proposed, aiming at reducing computational costs and memory footprints required during fine-tuning. Representative techniques such as Adapter Tuning (Houlsby et al., 2019; Pfeiffer et al., 2020), Low-Rank Adaptation (LoRA) (Hu et al., 2022) and its variants (Liu et al., 2024; Wu et al., 2024) significantly improve efficiency by freezing pre-trained weights and introducing a small number of trainable adapter parameters.

In theory, adapters introduce only a small amount of additional computation, but they can result in significant training-time overhead in practice; the base and adapter paths are executed sequentially, and computing gradients for adapter parameters typically requires retaining full input activations, which can increase kernel launches and activation memory. By contrast, adapter-free

approaches avoid these structural overheads by directly updating pretrained weights, but weight selection remains a key challenge. Random or structure-based selection methods, such as PaCA (Woo et al., 2025) and S²FT (Yang et al., 2024), may fragment trainable pathways across layers, while calibrated selection methods such as SMT (He et al., 2025) incur additional selection cost. Our study focuses on characterizing these trade-offs and designing a calibration-free, pathway-aligned adapter-free PEFT approach.

In this paper, we first analyze the sources of training overhead in widely used adapter-based methods. We then introduce **Recursively Aligned Pathway Adaptation (RAPA)**, an adapter-free PEFT algorithm that selects and aligns indices across all layers so that the updated weights form a consistent gradient pathway. By aligning the trainable coordinates through depth (and, when applicable, choosing shape-balanced submatrices), RAPA aims to enhance representational capacity and stabilize gradient flow while avoiding both adapter-induced overhead and per-layer re-sampling costs.

We evaluate RAPA across diverse models and tasks. Specifically, we conduct 5-shot MMLU fine-tuning (Hendrycks et al., 2021) (Figure 1), commonsense reasoning benchmarks, and instruction tuning followed by MT-Bench evaluation. All methods are compared under matched trainable-parameter budgets and comparable optimization settings to ensure fair comparison with prior PEFT baselines. Full datasets, hyperparameters, and evaluation protocols are provided in the experimental setup for reproducibility.

2 Background and Motivation

2.1 Challenges in Adapter-based PEFT

Fine-tuning LLMs typically incurs significant memory and computation costs. This is mainly because back-propagation requires storing all intermediate activations and updating a massive set of parameters. These requirements hinder practical deployment of LLMs, especially on resource-constrained hardware.

To address these challenges, PEFT methods aim to reduce the number of trainable parameters while maintaining task accuracy. One widely used approach is LoRA (Hu et al., 2022), which introduces additional rank-constrained matrices to the existing pre-trained weights as below:

$$W = W_0 + \Delta W = W_0 + s \cdot BA$$

where $A \in \mathbb{R}^{r \times d_{in}}$, $B \in \mathbb{R}^{d_{out} \times r}$, and s is a scaling factor. While LoRA reduces the number of trainable parameters, it still requires storing all input activations in memory, as the gradients for the low-rank adapter matrices must be computed with respect to the original input activation. In addition, LoRA introduces sequential computation steps and hence suffers from latency overhead during training; the forward propagation of LoRA proceeds sequentially through the pre-trained weights and the low-rank adapter, which hinders hardware-level parallelism. It was reported that LoRA incurs approximately 33% training-time overhead compared to full fine-tuning, as the forward pass must compute both the pre-trained weight and the low-rank adapter in series rather than in parallel (Hu et al., 2022; Woo et al., 2025). In addition, its low-rank structure limits representational capacity on complex tasks.

2.2 Approaches to Weight Selection in Adapter-Free PEFT

The central challenge in adapter-free PEFT is selecting which subset of weights to update. Existing strategies range from random or structure-aware selection, which prioritizes speed and implementation efficiency (Woo et al., 2025; Yang et al., 2024), to importance-based approaches that score weights via gradient magnitudes or activation sensitivities (He et al., 2025). However, random or efficiency-oriented selection may still yield fragmented trainable pathways across depth, while importance-driven methods typically require a costly precalibration step, introducing additional overhead and potential sensitivity to the calibration data.

In this work, we propose a *calibration-free, structure-aware selection* strategy that aligns trainable coordinates across layers (Section 3.2). By leveraging the inherent model architecture, our method avoids precomputation overhead while yielding tight seed-to-seed variability (Appendix A) and achieving high performance across tasks.

3 Analysis and Method

In this section, we first characterize the training-time overhead of adapter-based PEFT. We then introduce *Recursively Aligned Pathway Adaptation (RAPA)*, a calibration-free, structure-aware weight selection method. The alignment is applied *recursively* across all layers—mirroring the top-down

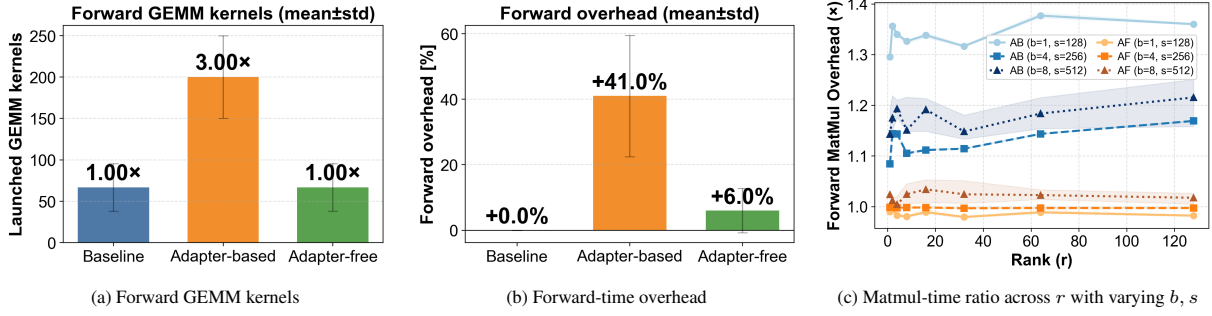


Figure 2: **(a)** Forward GEMM kernel launches. Adapter-based (AB) issues $\sim 3\times$ more kernels than the baseline. **(b)** Forward-time overhead. AB increases latency by $\sim 40\%$, whereas adapter-free (AF) closely matches baseline. **(c)** Forward-propagation matmul time ratio (each method to baseline). Overhead depends more on batch size b and sequence length s than to rank r .

flow of backpropagation—to form coherent pathways. This is achieved by tuning balanced square submatrices of pretrained weights, enforcing cross-layer consistency in the chosen coordinates, and maximizing shared indices across layers under a fixed parameter budget. In experiments, RAPA achieves faster convergence and higher accuracy while preserving adapter-free training efficiency and avoiding calibration or per-layer resampling overhead.

3.1 Revisiting Adapter-based PEFT Overhead

To motivate our adapter-free (AF) approach, we conduct a controlled microbenchmark to isolate the sources of overhead in adapter-based (AB) methods. On a single linear layer (4096×4096 , bfloat16) using an NVIDIA A100 (80 GB) GPU, we compare an AB setup (LoRA (Hu et al., 2022)) against an AF setup and a frozen baseline, keeping the trainable parameter budget identical for both AB and AF. For AF, we directly tune a $k \times k$ subblock of the frozen weight matrix W . We measure throughput with CUDA events and collect CUDA-kernel statistics using `torch.profiler`; results are aggregated over multiple trials (median, with 95% bootstrap confidence intervals).

As illustrated in Figure 2(a), the AB method substantially increases the number of General Matrix Multiplication (GEMM) kernel launches—by over $3\times$ on average, which corresponds to a forward-time overhead of $\sim 41\%$ (Figure 2(b)). In contrast, AF keeps the kernel count essentially unchanged and incurs only a small overhead ($\sim 6\%$).

These results indicate that the dominant inefficiency is not added FLOPs but the proliferation of small, sequential kernel launches that underutilize the GPU (Shi et al., 2016). Figure 2(c) shows that the forward matrix-multiplication overhead is more sensitive to batch size b and sequence length

s than to the adapter rank r . Simply scaling b or s to improve utilization is often impractical due to activation-memory growth in deep models. This structural overhead of separate adapter modules motivates RAPA, which aims to retain high performance without incurring this computational burden. We further confirm that this trend holds consistently for both a smaller and a larger model; the corresponding overhead measurements are reported in Appendix B.

3.2 Recursively Aligned Pathway Adaptation

3.2.1 Balanced Submatrix Selection

Let $W \in \mathbb{R}^{d \times d}$ be a pretrained weight matrix. LoRA with rank r learns a low-rank update $\Delta W = BA$ and introduces $2rd$ trainable parameters. We keep the same budget, but allocate it to a square-shaped indexed submatrix:

$$k^2 = 2rd \implies k \approx \sqrt{2rd}, \quad (1)$$

(where k can be rounded to an integer in practice).

We tune a square-shaped submatrix of W with size $k \times k$, i.e., we directly update k^2 entries while keeping the rest of W frozen. This changes *where* parameters are placed (a 2-D block) without changing *how many* are trained.

An important structural consequence is the width of the selected pathway. For index sets I and J , both the rank of a selected update and the bottleneck dimension of the paired pathway $W_{J,I}^{(b,2)} W_{I,J}^{(b,1)}$ are upper-bounded by $w_{\text{path}} := \min(|I|, |J|)$. Under the fixed parameter budget $|I||J| = 2rd$, w_{path} is maximized when $|I| \approx |J|$. Thus, balanced selection maximizes the local rank ceiling and the number of coordinates that can be consistently coupled across adjacent projections under the same budget.

In terms of training-time compute, LoRA introduces an additional low-rank update path, whereas

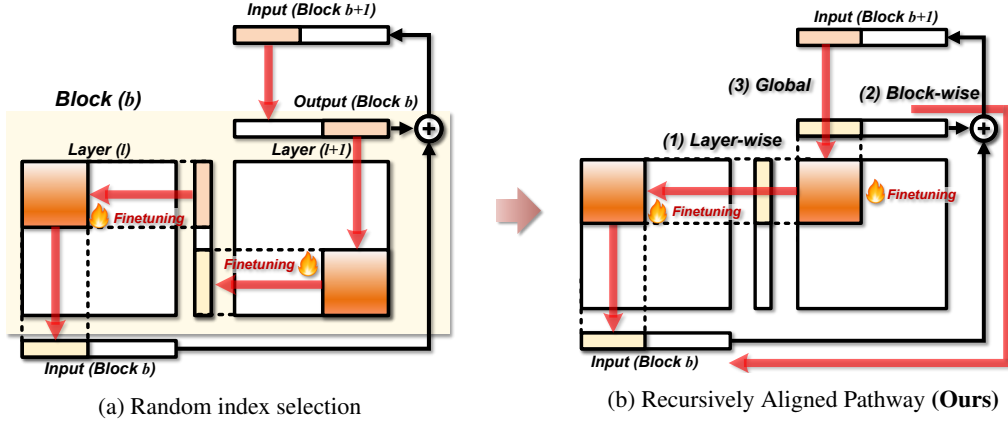


Figure 3: Comparison between independent and recursively aligned index selection. **(a)** Independent selection can fragment the trainable support across connected projections, disrupting the continuity of the selected trainable pathway. **(b)** RAPA aligns balanced submatrices across layers, residual endpoints, and depth, maintaining a coherent trainable pathway; this alignment accelerates loss convergence in our ablation.

our implementation keeps the base linear matmul and computes/updates only the selected ($\text{row}_r \times \text{col}_r$) block via index-based gather/scatter and a custom backward on pruned activations. Thus, the dominant cost for learning the update remains comparable to LoRA up to lightweight indexing overhead, with the exact runtime depending on the implementation. The ratio sweep in Section 4.1 supports this structural choice: accuracy improves as the selected support approaches a balanced regime, with the best result near $|I| = |J|$.

3.2.2 Structural Index Alignment for Coherent Trainable Pathways

RAPA recursively aligns selected row and column coordinates across connected projections and residual endpoints, forming a depth-consistent trainable support. The shared coordinates provide a common pathway for selected weights to co-adapt across depth, while frozen pretrained weights remain active in the full forward and backward computations.

Throughout this paper, we use the term *block* to denote a Transformer sub-module, namely an attention or a feed-forward network (FFN) block. For notational clarity, we model each block as **two consecutive linear layers** whose input and output are linked by a residual connection (He et al., 2016b,a).

We consider three levels of index alignment:

- **Layer-wise alignment:** the output indices of one linear layer match the input indices of the next.
- **Block-wise alignment:** the residual input/output indices match within a two-layer residual block.

- **Global alignment:** the same residual index set is reused across multiple residual-connected blocks.

Method. Figure 3 overviews the mechanism. With a $k \times k$ trainable submatrix per projection, independently sampled index sets may mismatch the selected output coordinates of one projection with the selected input coordinates of the next, fragmenting the direct pathway among trainable weights. RAPA instead reuses compatible indices across residual-connected projections to build a shared aligned pathway.

- Define a shared row/column index structure** under a fixed parameter budget.
- Tie this structure across residual-connected layers** so updates stay on the same coordinates.

This shared-index tying induces *layer-wise*, *block-wise*, and *global* alignment.

We place trainable weights within block b on $W_{I,J}^{(b,1)}$ for the first linear layer and $W_{J,I}^{(b,2)}$ for the second, so the intermediate activation lives on I and the residual input/output lives on J :

$$\begin{aligned} x_I^{(b,1)} &= W_{I,J}^{(b,1)} x_J^{(b)}, \\ x_J^{(b+1)} &= x_J^{(b)} + W_{J,I}^{(b,2)} x_I^{(b,1)}. \end{aligned} \quad (2)$$

Eq. 2 illustrates **layer-wise alignment** via the shared intermediate index set I across the two linear layers, and **block-wise alignment** via the

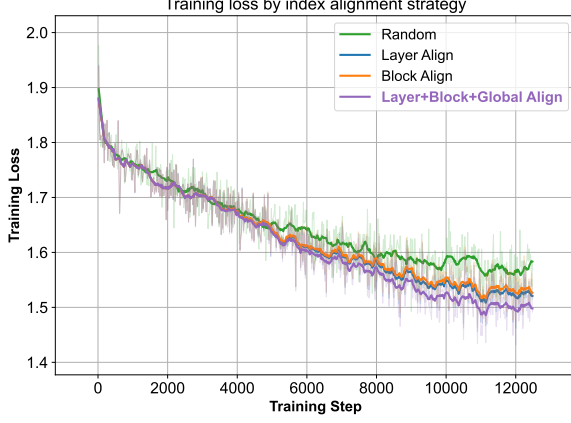


Figure 4: Comparison of training loss curves for index alignment strategies.

Table 1: Effect of index alignment on 5-shot MMLU accuracy.

Method	Hums.	STEM	Social.	Other	Avg.
Random	48.2	41.3	60.2	59.4	51.7
Layer Align	49.3	42.1	59.8	58.4	52.0
Block Align	49.4	41.5	59.9	57.9	51.8
Layer+Block+Global	49.3	42.1	60.3	59.1	52.3

shared residual index set J between the block input $x_J^{(b)}$ and output $x_J^{(b+1)}$.

Global alignment is obtained by reusing the same residual index set J across multiple residual-connected blocks. Define the within-block transformation (restricted to the aligned subspace) as $\mathcal{F}^{(b)}(\cdot)$, which maps $x_J^{(b)}$ to the residual update on J . Then the forward propagation through block b can be written as

$$x_J^{(b+1)} = x_J^{(b)} + \mathcal{F}^{(b)}(x_J^{(b)}) \quad (3)$$

$$\approx x_J^{(b)} + W_{J,I}^{(b,2)} W_{I,J}^{(b,1)} x_J^{(b)}, \quad (4)$$

and over m consecutive residual-connected blocks,

$$x_J^{(b+m)} = x_J^{(b)} + \sum_{j=0}^{m-1} \mathcal{F}^{(b+j)}(x_J^{(b+j)}). \quad (5)$$

These rules organize the selected updates into a depth-consistent support; they do not remove frozen coordinates from the full computation graph.

In the tested LLaMA2-7B setting, full alignment is associated with lower training loss (Figure 4) and higher final MMLU accuracy than random selection (Table 1). Multi-seed loss curves are provided in Appendix F.

Depthwise support continuity. RAPA preserves coordinate compatibility between consecutive trainable submatrices and residual endpoints, allowing

updates in connected projections to co-adapt within a common coordinate system across depth. Frozen pretrained weights remain active in the dense forward and backward computations; alignment organizes the direct coupling among trainable coordinates. Appendix C provides the detailed layer-, block-, and global-level derivations, while Appendices D and G extend the same construction to attention and MoE blocks.

We directly test this support-continuity property with a restricted-VJP (Vector-Jacobian Product) diagnostic on pretrained Qwen3-4B (Yang et al., 2025). Let B_b denote the full Jacobian of block b , including frozen weights and residual connections, and let P_{J_b} project onto $K = 384$ selected residual coordinates. Applying $g_{b-1} = P_{J_{b-1}} B_{b-1}^\top P_{J_b} g_b$ through all 36 blocks, the geometric mean of the per-block retained VJP fraction is 0.864 with a shared J and 0.385 with block-specific J_b , an absolute difference of 0.478 (95% CI [0.462, 0.495]); the shared construction is higher in all five paired selector seeds. This diagnostic indicates stronger depthwise continuity within the shared selected support. Separate masked-training ablations on LLaMA2-7B are reported in Figure 4 and Table 1, while Appendix K provides the full restricted-VJP protocol and analysis.

4 Experiments

We evaluate RAPA on MMLU (Hendrycks et al., 2021), eight commonsense reasoning tasks (Clark et al., 2018, 2019; Zellers et al., 2019; Mihaylov et al., 2018; Bisk et al., 2020; Sap et al., 2019; Sakaguchi et al., 2020), MT-Bench instruction tuning (Zheng et al., 2023), and task-specific code generation and mathematical reasoning on Qwen3-4B, with an additional code and efficiency evaluation on SmolLM3-3B-Base (Bakouch et al., 2025). Unless otherwise stated, experiments use a single NVIDIA A100 80GB GPU with 16-bit mixed precision (Mickiewicz et al., 2018) and AdamW (Loshchilov and Hutter, 2019). For the main LLaMA2/Mistral comparisons, we compare adapter-based baselines (LoRA (Hu et al., 2022), DoRA (Liu et al., 2024), MosLoRA (Wu et al., 2024)) and adapter-free baselines (PaCA (Woo et al., 2025), SMT (He et al., 2025), S²FT (Yang et al., 2024)) under matched trainable-parameter budgets, following the target modules of the original papers or released implementations when available. Detailed settings, module choices, ablations, and additional weight-

selection comparisons are provided in Appendix J and Appendix M.

4.1 Effects of Balanced Weight Selection and Alignment

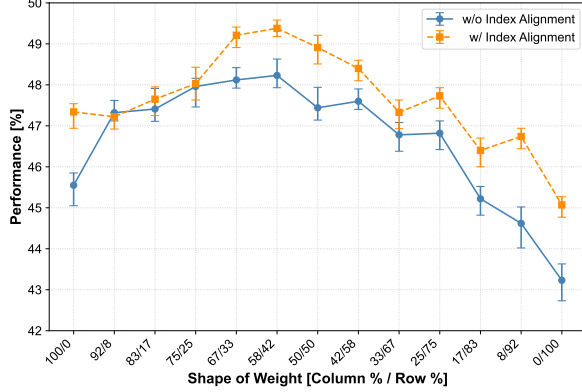


Figure 5: Effects of weight shape (row/column ratio) and index alignment on MMLU accuracy.

We investigated how the column-to-row ratio of trainable weights influences task accuracy and how this relationship changes after index alignment is applied. Figure 5 plots MMLU accuracy as a function of the selected column-to-row ratio on x-axis, comparing cases without index alignment (blue curve) and with index alignment (yellow curve). We fine-tuned the LLaMA2-7B model (Touvron et al., 2023) on the MMLU dataset for this analysis; the detailed experimental setup is provided in Appendix J (Table 13).

As shown in Figure 5, selecting a symmetric subset of weights yields higher accuracy compared to extreme cases where only row or column weights are selected. Focusing on the random selection case (blue curve), a balanced (50/50) ratio achieves an accuracy of 47.4%, outperforming the extreme column-only case (100/0) and row-only (0/100) configurations by 1.8% and 4.2%, respectively.

This trend remains consistent when the **index alignment method** is applied: balanced selection again achieves the highest accuracy. Moreover, the aligned variant consistently outperforms the random baseline across all settings. In the balanced case, index alignment improves performance to 48.9%, 1.5% higher than the random selection method at 47.4%. Even in the column-only setting, alignment increases accuracy from 45.6% to 47.3%, and in the row-only setting, from 43.2% to 45.1%, confirming that alignment enhances performance even under asymmetric or suboptimal weight configurations.

4.2 Fine-tuning for Downstream Tasks

We validate the task accuracy and efficiency of RAPA using the MMLU 5-shot benchmark, which evaluates complex reasoning across 57 academic subjects. All methods were evaluated on LLaMA2-7B under comparable trainable parameter budgets, using LoRA ranks $r=8$ and matching the other PEFT methods accordingly. Additional scalability experimental results using LLaMA2-13B are reported in Appendix L. The results from additional experiments across various rank settings are reported in Appendix H, and the detailed configurations are provided in Appendix J.

4.2.1 MMLU

On the MMLU benchmark, RAPA achieves the highest average accuracy among PEFT methods on LLaMA2-7B (Table 2). It outperforms the strongest adapter-free baseline, SMT, by 1.3 points while maintaining comparable or faster training time. RAPA also surpasses adapter-based baselines in average accuracy and trains substantially faster than DoRA. Thus, RAPA improves the accuracy-efficiency trade-off over both random/structure-based and calibrated adapter-free baselines.

4.2.2 Commonsense Reasoning

We further evaluated RAPA on eight canonical commonsense reasoning tasks, comparing it against the same adapter-based and adapter-free baselines. All experiments were conducted on the LLaMA2-7B model and we fine-tuned it on a commonsense reasoning dataset consisting of 170k samples. Detailed hyperparameters are listed in Appendix J.

Table 3 reports that RAPA achieves the highest macro-average accuracy of 69.7%, outperforming the strongest adapter-based baseline, MosLoRA (69.1%), by 0.6 points and the strongest adapter-free baseline, PaCA (68.2%), by 1.5 points. RAPA also improves over SMT (68.1%) and S²FT (67.8%) by 1.6 and 1.9 points, respectively.

Despite higher accuracy, RAPA keeps the computational footprint close to other adapter-free methods: 42.1 GB peak GPU memory and 4.1 hours of wall-clock time, comparable to PaCA (41.5 GB, 4.0h) and S²FT (42.0 GB, 4.2h). In contrast to SMT, RAPA is calibration-free, whereas SMT requires a 100-iteration calibration stage that adds about 28 minutes in our setup. Relative to adapter-based baselines, RAPA uses about 20% less memory than LoRA and MosLoRA and less than half the training time of DoRA.

Table 2: MMLU 5-shot results on LLaMA2-7B.

Type	Method	Params	Mem	Time	Accuracy (%)				
					Hums.	STEM	Social.	Other	Avg.
–	Baseline	–	–	–	43.6	37.1	51.4	53.0	45.9
AB	LoRA	20.0M	22.6G	4.1h	48.5	41.2	57.3	56.5	50.6
	DoRA	21.4M	28.0G	8.7h	48.7	42.3	58.3	57.6	51.3
	MosLoRA	20.0M	22.6G	4.3h	46.6	42.2	60.8	57.4	51.1
AF	PaCA	21.8M	19.5G	3.2h	48.7	41.7	58.7	57.6	51.2
	SMT [†]	20.0M	25.5G	3.6h	47.7	42.0	59.7	58.7	51.5
	S ² FT	19.8M	20.9G	3.2h	44.6	38.6	53.4	53.9	47.3
	RAPA (Ours)	20.1M	19.6G	3.2h	50.1	42.6	60.8	59.1	52.8

Note: AB and AF denote adapter-based and adapter-free PEFT methods, respectively.

[†] Method with an additional calibrated selection stage; Mem and Time include this stage.

Table 3: Commonsense reasoning results on LLaMA2-7B.

Type	Method	Params	Mem	Time	ARC-c	ARC-e	BoolQ	Hella	OBQA	PIQA	SIQA	Wino	Avg.
–	Baseline	–	–	–	43.3	74.6	77.7	76.0	44.2	79.1	46.1	69.1	63.8
AB	LoRA	320M	53.2G	4.8h	43.0	72.6	80.6	67.8	44.6	76.6	54.9	74.3	64.3
	DoRA	321M	75.3G	8.8h	52.1	80.0	73.7	76.4	49.0	80.4	58.4	82.0	69.0
	MosLoRA	324M	53.4G	5.0h	53.0	81.2	70.7	76.3	50.2	80.8	58.8	82.1	69.1
AF	PaCA	348M	41.5G	4.0h	51.0	78.5	73.7	75.4	48.8	80.2	57.5	80.3	68.2
	SMT [†]	320M	38.6G	4.8h	48.8	78.3	78.0	75.5	47.8	79.3	57.4	79.8	68.1
	S ² FT	320M	42.0G	4.2h	50.5	76.4	75.2	74.2	47.4	80.1	58.0	80.4	67.8
	RAPA (Ours)	320M	42.1G	4.1h	53.1	81.1	76.4	77.1	49.8	80.4	58.9	80.8	69.7

Note: AB and AF denote adapter-based and adapter-free PEFT methods, respectively.

[†] Method with an additional calibrated selection stage; Mem and Time include this stage.

These results reinforce the MMLU findings: RAPA improves accuracy across diverse reasoning tasks while retaining the memory and runtime advantages of adapter-free fine-tuning. The gains suggest that structured square selection and cross-layer index alignment provide a more effective trainable subspace than random, structure-only, or calibrated selection under comparable parameter budgets.

4.2.3 Instruction Tuning

We evaluated RAPA on the Mistral-7B model, which was instruction-tuned for one epoch on the OASST1 dataset using a single NVIDIA A100 GPU. Performance was subsequently measured using the MT-Bench benchmark, which averages scores over two conversational turns. We used GPT-4o-mini as the evaluator for this benchmark (see Table 18 in Appendix J for further details).

As shown in Table 4, RAPA achieves the highest average MT-Bench score among all methods. This indicates that RAPA improves the accuracy limitations of prior adapter-free methods while retaining their training efficiency, and also surpasses strong adapter-based baselines.

4.2.4 Code Generation and Mathematical Reasoning

We evaluate task-specific code generation and mathematical reasoning on Qwen3-4B under matched trainable parameter budgets of approximately 88M (low) and 627M (high). RAPA, LoRA, and PaCA use the same seven target projection types and matched trainable parameter budgets. Code evaluation covers all 164 HumanEval (Chen et al., 2021) and 257 MBPP (Austin et al., 2021) problems with greedy decoding, while mathematical reasoning is evaluated on GSM8K (Cobbe et al., 2021) and MATH-500 (Lightman et al., 2024). Separately, we conduct an additional paired RAPA–LoRA comparison on SmoLM3-3B-Base to examine code-generation performance and end-to-end training efficiency. Exact parameter counts and profiling details are provided in Appendix I.1.

In the low-budget Qwen3-4B code setting, the prespecified fixed run gives a macro score of 69.53 for RAPA, compared with 65.36 for LoRA and 64.79 for PaCA. At the high budget, the corresponding single-run scores are 68.00, 65.42, and 66.51. For low-budget mathematical reasoning, RAPA obtains 63.38, compared with 61.41 for LoRA and

Table 4: MT-Bench instruction-tuning results on Mistral-7B (Jiang et al., 2023).

Type	Method	Params	Mem	Time	Hum.	STEM	Role	Extr.	Writ.	Reas.	Code	Math	Avg.
–	Baseline	–	–	–	1.85	3.20	2.95	3.15	2.50	1.70	2.30	1.40	2.38
AB	LoRA	168M	52G	47m	6.40	5.75	4.35	5.20	5.25	4.20	3.30	1.95	4.55
	DoRA	169M	69G	93m	6.70	5.60	4.30	4.40	5.75	4.50	3.85	1.95	4.63
	MosLoRA	169M	52G	48m	5.80	6.05	4.85	5.00	5.00	3.75	3.65	2.00	4.51
AF	PaCA	176M	47G	41m	5.35	5.45	4.15	4.50	4.55	2.55	3.20	1.95	3.96
	SMT [†]	170M	54G	45m	6.05	5.75	5.55	4.72	5.12	3.97	3.05	1.75	4.49
	S ² FT	170M	47G	38m	5.10	4.85	4.95	4.30	4.60	3.05	2.55	1.50	3.86
	RAPA (Ours)	168M	48G	42m	6.70	5.82	5.65	5.20	5.85	4.20	3.22	2.05	4.84

Note: AB and AF denote adapter-based and adapter-free PEFT methods, respectively.

[†] Method with an additional calibrated selection stage; Mem and Time include this stage.

Table 5: Code and math performance on Qwen3-4B (top), and code performance and efficiency on SmolLM3-3B-Base (bottom). Code averages HumanEval and MBPP, and Math averages GSM8K and MATH-500. Low/high correspond to approximately 88M/627M trainable parameters.

Setting	LoRA	PaCA	RAPA
Code (low)	65.36	64.79	69.53
Code (high)	65.42	66.51	68.00
Math (low)	61.41	62.66	63.38

Metric	LoRA	RAPA	Δ
Code $\uparrow$	54.94	55.19	+0.25
MBPP $\uparrow$	61.09	63.42	+2.33
Time (s) $\downarrow$	451.21	297.15	−34.14%
Memory (GiB) $\downarrow$	39.61	27.76	−29.92%

62.66 for PaCA. Table 5 states the run protocol for each row. A separate fixed-ledger Qwen3-4B comparison against 12 additional baselines gives RAPA the highest seed-42 Code Macro (68.45); Appendix I.2 reports the full protocol and panel.

In the SmolLM3-3B-Base comparison, RAPA attains a Code Macro of 55.19 and an MBPP score of 63.42, compared with 54.94 and 61.09 for LoRA. RAPA also reduces training time from 451.21 s to 297.15 s and peak memory from 39.61 GiB to 27.76 GiB. Thus, RAPA improves both code performance and training efficiency over LoRA.

4.3 Architecture-level Ablation: MoE Compatibility

Finally, we examine whether RAPA’s alignment principle extends beyond dense Transformer blocks. We conduct a controlled ablation on DeepSeek-MoE-16B (Dai et al., 2024), comparing square-random and square-aligned index selection under the same trainable parameter budget with fixed router parameters. As shown in Table 6, square-aligned selection reduces PPL from 4.34 to 3.74 and improves 5-shot MMLU accuracy from 50.5 to 51.9. Figure 6 further shows lower training loss,

Table 6: MoE compatibility ablation on DeepSeek-MoE-16B. PPL is training perplexity; MMLU reports 5-shot accuracy.

Method	PPL $\downarrow$	MMLU Avg.
Baseline	8.71	44.8
Square-random	4.34	50.5
Square-aligned (RAPA)	3.74	51.9

MoE Training Loss: Random vs Aligned Index Selection

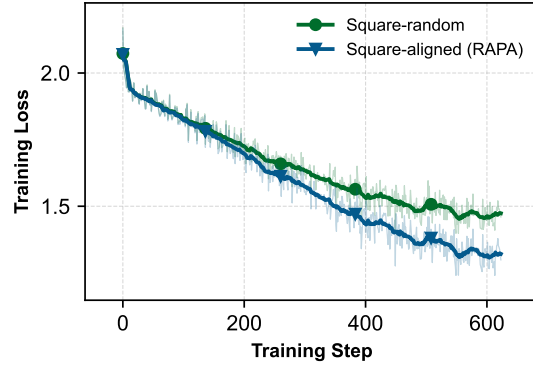


Figure 6: MoE training loss under square-random and square-aligned index selection.

indicating improved optimization as well as final accuracy. The full MoE derivation is provided in Appendix G.

5 Related work

Adapter-based PEFT Adapter-based PEFT methods freeze pretrained weights and train modules or low-rank branches. Early adapter methods insert bottleneck modules into each layer (Houlsby et al., 2019; Pfeiffer et al., 2020), while LoRA (Hu et al., 2022) and its variants (Liu et al., 2024; Wu et al., 2024) learn low-rank updates that can be merged into the base model at inference. These methods are effective, but the additional adapter path can increase activation storage, kernel launches, and training-time overhead.

Adapter-free and sparse PEFT. Adapter-free and sparse PEFT methods restrict adaptation to selected pretrained parameters or sparse task-specific updates. Existing approaches mainly differ in the criterion used to choose trainable weights. PaCA (Woo et al., 2025) uses random partial connections, which keeps selection overhead negligible but does not explicitly coordinate selected coordinates across layers. S²FT (Yang et al., 2024) applies structured sparsity over organized units, such as projection components or channels, to make sparse updates more compatible with efficient dense computation. FISH Mask (Sung et al., 2021) and SMT (He et al., 2025) use task-dependent Fisher or gradient signals to construct fixed supports; PaFi (Liao et al., 2023) instead uses a task-agnostic pretrained magnitude criterion, and SPEFT (Liu et al., 2025) systematically compares salience metrics and static versus dynamic masking. LT-SFT (Ansell et al., 2022) identifies large-change coordinates through preliminary fine-tuning and rewinding, while Diff Pruning (Guo et al., 2021) jointly learns a sparse task-specific difference vector. SpIEL (Ansell et al., 2024) periodically prunes and regrows sparse parameter deltas, whereas LISA (Pan et al., 2024) periodically resamples the intermediate layers optimized in full rank. RAPA instead constructs a fixed, depth-consistent trainable support by recursively reusing row and column indices across connected projections and residual endpoints throughout depth, avoiding the additional overhead of task-dependent weight search or calibration.

6 Conclusion

We presented RAPA, an adapter-free PEFT method that increases capacity and stabilizes optimization without adding modules or extra calibration passes. RAPA reallocates a parameter budget to balanced square submatrices and aligns their indices across depth, forming coherent trainable pathways and yielding more expressive updates under the same budget. Across MMLU, commonsense reasoning, MT-Bench, and code/math tasks, RAPA achieves strong accuracy under comparable parameter budgets while retaining adapter-free training efficiency. Controlled ablations further show that balanced selection and index alignment improve optimization and final accuracy. We summarize broader impacts in Appendix O.

Limitations

Although we evaluate RAPA across representative models, tasks, and training configurations, the observed accuracy–efficiency trade-offs may vary with model architecture, downstream data distribution, parameter budget, and target module choices. As with other PEFT methods, further validation under broader settings would help better characterize its generality. Future work will explore more adaptive pathway alignment strategies and extend the analysis to a wider range of PEFT scenarios.

Ethical Considerations

RAPA is a general-purpose PEFT method evaluated on standard public benchmarks and does not introduce new user data, annotation, or deployment-specific applications. While lower-cost adaptation can improve accessibility, it may also reduce the barrier to unintended or harmful model adaptation; we therefore limit our claims to benchmark-level accuracy and efficiency rather than deployment-level safety.

Artifact use and licenses. We use publicly released models, datasets, and evaluation tools only for research and reproducibility purposes, following their standard benchmark protocols and original licenses or terms of use. We do not redistribute original model weights, datasets, or derived data; released code is intended for reproducing the experiments and should be used together with the licenses and access conditions of the corresponding upstream artifacts.

Acknowledgements

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation under Grant RS-2021-II211343, Grant IITP-2026-RS-2023-00256081, and Grant RS-2024-00347394, and in part by the National Research Foundation of Korea under Grant RS-2024-00408040 and Grant RS-2022-00144419.

References

- Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796.

- Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M. Ponti. 2024. [Scaling sparse fine-tuning to large language models](#). *Preprint*, arXiv:2401.16405.
- Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. [Program synthesis with large language models](#). *Preprint*, arXiv:2108.07732.
- Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Noumane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, and 1 others. 2025. [Smollm3: smol, multilingual, long-context reasoner](#). *Hugging Face Blog*.
- Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. [PIQA: Reasoning about physical commonsense in natural language](#). In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 7432–7439, New York, NY, USA. AAAI Press.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2023. Palm: scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. [BoolQ: Exploring the surprising difficulty of natural yes/no questions](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. [Think you have solved question answering? try arc, the ai2 reasoning challenge](#). *arXiv preprint arXiv:1803.05457*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. [Training verifiers to solve math word problems](#). *Preprint*, arXiv:2110.14168.
- Damai Dai, Chengqi Deng, Chenggang Zhao, R.X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y.K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. [DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297, Bangkok, Thailand. Association for Computational Linguistics.
- DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *CoRR*, abs/2501.12948.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 82 others. 2024. [The llama 3 herd of models](#). *CoRR*, abs/2407.21783.
- Demi Guo, Alexander Rush, and Yoon Kim. 2021. [Parameter-efficient transfer learning with diff pruning](#). 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)*, pages 4884–4896, Online. Association for Computational Linguistics.
- Haoze He, Juncheng B Li, Xuan Jiang, and Heather Miller. 2025. [SMT: Fine-tuning large language models with sparse matrices](#). In *The Thirteenth International Conference on Learning Representations*.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. [Deep Residual Learning for Image Recognition](#). In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Los Alamitos, CA, USA. IEEE Computer Society.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. [Identity mappings in deep residual net-](#)

- works. In *Computer Vision – ECCV 2016*, pages 630–645, Cham. Springer International Publishing.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. [Measuring massive multitask language understanding](#). In *International Conference on Learning Representations*.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *International Conference on Learning Representations (ICLR)*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, L  lio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timoth  e Lacroix, and William El Sayed. 2023. [Mistral 7b](#). *Preprint*, arXiv:2310.06825.
- Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, L  lio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, and 7 others. 2024. [Mixtral of experts](#). *CoRR*, abs/2401.04088.
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. [Scaling laws for neural language models](#). *Preprint*, arXiv:2001.08361.
- Baohao Liao, Yan Meng, and Christof Monz. 2023. Parameter-efficient fine-tuning without introducing new latency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4242–4260.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. [Let’s verify step by step](#). In *The Twelfth International Conference on Learning Representations*.
- Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. [DoRA: weight-decomposed low-rank adaptation](#). In *Proceedings of the 41st International Conference on Machine Learning*, ICML’24.
- Xinxin Liu, Aaron Thomas, Cheng Zhang, Jianyi Cheng, Yiren Zhao, and Xitong Gao. 2025. [Refining salience-aware sparse fine-tuning strategies for language models](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 31932–31945, Vienna, Austria. Association for Computational Linguistics.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. [Mixed precision training](#). In *International Conference on Learning Representations*.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. [Can a suit of armor conduct electricity? a new dataset for open book question answering](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024. [LISA: Layerwise importance sampling for memory-efficient large language model fine-tuning](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 57018–57049.
- Jonas Pfeiffer, Andreas R  ckl  , Clifton Poth, Aishwarya Kamath, Ivan Vuli  , Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. [AdapterHub: A framework for adapting transformers](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. [Winogrande: An adversarial winograd schema challenge at scale](#). In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 8732–8740, New York, NY, USA. AAAI Press.
- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. [Social IQa: Commonsense reasoning about social interactions](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the*

- 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4463–4473, Hong Kong, China. Association for Computational Linguistics.
- Yang Shi, U. N. Niranjan, Animashree Anandkumar, and Cris Cecka. 2016. [Tensor contractions with extended blas kernels on cpu and gpu](#). In *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, pages 193–202.
- Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. Training neural networks with fixed sparse masks. In *Proceedings of the 35th International Conference on Neural Information Processing Systems, NIPS '21*, Red Hook, NY, USA. Curran Associates Inc.
- Lintang Sutawika, Hailey Schoelkopf, Leo Gao, Baber Abbasi, Stella Biderman, Jonathan Tow, ben fatori, Charles Lovering, farzanehnakhaee70, Jason Phang, Anish Thite, Fazz, Thomas Wang, Niklas Muennighoff, Aflah, sdtblk, nopperl, gakada, tttyun-tian, and 11 others. 2024. [Eleutherai/lm-evaluation-harness: v0.4.2](#).
- Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, Soroosh Mariooryad, Yifan Ding, Xinyang Geng, Fred Alcober, Roy Frostig, Mark Omernick, Lexi Walker, Cosmin Paduraru, Christina Sorokin, and 1118 others. 2024. [Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context](#). Preprint, arXiv:2403.05530.
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). Preprint, arXiv:2307.09288.
- Sunghyeon Woo, Sol Namkung, Sunwoo Lee, Inho Jeong, Beomseok Kim, and Dongsuk Jeon. 2025. [PaCA: Partial connection adaptation for efficient fine-tuning](#). In *The Thirteenth International Conference on Learning Representations*.
- Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. [Mixture-of-subspaces in low-rank adaptation](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 7880–7899. Association for Computational Linguistics.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. [Qwen3 technical report](#). Preprint, arXiv:2505.09388.
- Xinyu Yang, Jixuan Leng, Geyang Guo, Jiawei Zhao, Ryumei Nakada, Linjun Zhang, Huaxiu Yao, and Beidi Chen. 2024. [S²ft: Efficient, scalable and generalizable LLM fine-tuning by structured sparsity](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. [HellaSwag: Can a machine really finish your sentence?](#) In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.
- Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in neural information processing systems*, 32.
- Justin Zhao, Timothy Wang, Wael Abid, Geoffrey Angus, Arnav Garg, Jeffery Kinnison, Alex Sherstinsky, Piero Molino, Travis Addair, and Devvret Rishi. 2024. [Lora land: 310 fine-tuned llms that rival gpt-4, A technical report](#). CoRR, abs/2405.00732.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. [Judging LLM-as-a-judge with MT-bench and chatbot arena](#). In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

A Robustness of Fine-tuning vs. Initial Weight Sensitivity

Table 7: MMLU 5-shot accuracy comparison. The rows with the lowest (blue) and highest (red) average scores for each method are highlighted.

Method	Setting	Accuracy (%)				
		Hums.	STEM	Social.	Other	Avg.
Baseline	-	43.6	37.1	51.4	53.0	45.9
Pruning to Zero	Seed #1	34.0	31.8	38.5	42.8	36.4
	Seed #2	39.9	35.9	48.3	49.9	43.0
	Seed #3	39.5	38.3	47.4	49.1	43.1
	Seed #4	40.4	37.5	49.6	50.6	44.0
	Mean $\pm$ Std. (Avg.): 41.6 $\pm$ 3.5					
Finetuning	Seed #1	50.2	44.1	63.2	60.4	54.0
	Seed #2	51.3	43.2	62.6	60.9	54.1
	Seed #3	51.2	42.6	62.1	60.5	53.7
	Seed #4	51.2	44.3	62.8	60.6	54.3
	Mean $\pm$ Std. (Avg.): 54.0 $\pm$ 0.3					

To determine whether the initial weight sensitivity of a model to a downstream task could predict its final fine-tuning performance, we used the performance degradation from our "Pruning to Zero" experiment as a proxy for sensitivity (Table 7). The results revealed a high variance in sensitivity depending on the random seed; for instance, the accuracy drop from the baseline ranged from a minimal 1.9 % (Seed #4 from the previous table) to a substantial 9.5 % (Seed #1), indicating different sets of weights with varying importance were pruned.

Notably, we observed no correlation between this initial weight sensitivity and the final fine-tuning accuracy. Despite the wide disparity in the pruning outcomes, all fine-tuning runs converged to a narrow and high-performance bracket (53.7 % to 54.3 %). This suggests that the fine-tuning process is highly robust and overriding the initial weight state of the model to find a consistent solution.

B Overhead Ablation

To verify that the overhead pattern observed in the main text is not specific to a particular model width, we repeat our profiling on two decoder-only Transformer models with different hidden dimensions: a smaller model with $h_{\text{dim}} = 1024$ and a wider model with $h_{\text{dim}} = 8192$. For each model, we compare a frozen baseline, an adapter-based (AB) method, and an adapter-free (AF) counterpart under the same benchmarking protocol as in our main-text overhead analysis.

Figures 7(a)–(d) report the absolute number of GEMM kernel launches and the corresponding

forward-time overhead for the two widths. In both regimes, the AB method consistently triggers approximately $3\times$ more GEMM kernels than the baseline, translating to a 35–45% forward-time overhead. By contrast, the AF method closely tracks the baseline in both kernel count and run-time, incurring only a few percent overhead.

Figures 7(e)–(h) further break down the kernel-count and matrix-multiplication time *ratios* across a range of batch sizes, sequence lengths, and adapter ranks r . The overhead of AB is largely insensitive to r and h_{dim} , but grows with batch size and sequence length due to the structural cost of executing additional small kernels. In all settings, AF remains near parity with the baseline. These results confirm that the structural inefficiency of adapter modules is robust across model width, and that our adapter-free design can retain high efficiency even for wider models.

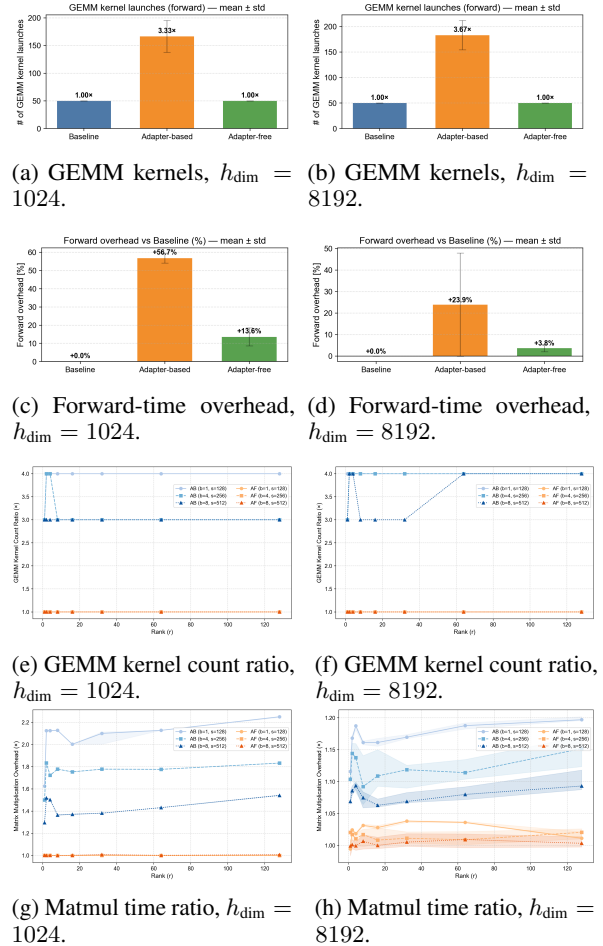


Figure 7: Overhead ablation for adapter-based (AB) and adapter-free (AF) methods on two model widths. (a, b) GEMM kernel counts; (c, d) forward-time overhead; (e, f) kernel-count ratios; (g, h) matmul-time ratios, all relative to the frozen baseline.

C Gradient Analysis of Index Alignment

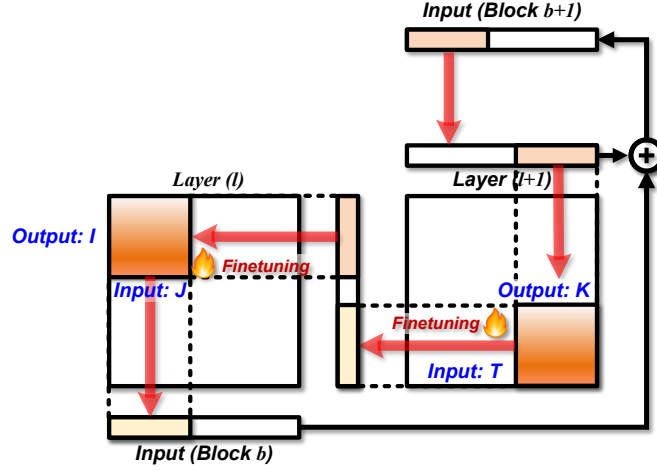


Figure 8: Example of disrupted gradient flow in the absence of index alignment

This section isolates the direct coupling between consecutively selected trainable submatrices. The full Transformer computation remains dense: frozen pretrained weights and residual paths continue to propagate activations and gradients outside this restricted selected-support component. We use I (intermediate) and J (residual) as the aligned index sets of size k , consistent with Section 3.2.2. To illustrate failure cases, we first write a *misaligned* two-linear residual block using arbitrary sets $K, T \subset \{1, \dots, d\}$ (also of size k). The aligned case is recovered by setting $T = I$ (layer-wise) and $K = J$ (block-wise).

$$x_I^{(l+1)} = W_{I,J}^{(l)} x_J^{(l)}, \quad (6)$$

$$x_K^{(l+2)} = x_K^{(l)} + W_{K,T}^{(l+1)} x_T^{(l+1)}. \quad (7)$$

C.1 Layer-wise Alignment: Aligning Output-Input Indices across Layers

In layer-wise alignment, the output indices of a layer are matched with the input indices of the next layer to enhance gradient propagation. Based on the forward propagation defined in Eqs. 6 and 7, the loss gradient with respect to the trainable weight $W_{K,T}^{(l+1)}$ at layer $l+1$ is given by

$$\frac{\partial \mathcal{L}}{\partial W_{K,T}^{(l+1)}} = \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}} \frac{\partial x_K^{(l+2)}}{\partial W_{K,T}^{(l+1)}}. \quad (8)$$

The weights corresponding to the selected subspace of layer $l+1$ are updated during back-propagation as follows:

$$W_{K,T}^{(l+1)} \leftarrow W_{K,T}^{(l+1)} - \eta \frac{\partial \mathcal{L}}{\partial W_{K,T}^{(l+1)}}, \quad (9)$$

where η is the learning rate.

By applying the chain rule, we can similarly compute the gradient of the selected subspace weights at layer l , $W_{I,J}^{(l)}$, as

$$\frac{\partial \mathcal{L}}{\partial W_{I,J}^{(l)}} = \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}} \frac{\partial x_K^{(l+2)}}{\partial x_I^{(l+1)}} \frac{\partial x_I^{(l+1)}}{\partial W_{I,J}^{(l)}}. \quad (10)$$

Key subtlety (misalignment). Eq. 7 uses only $x_T^{(l+1)}$ as input to the next linear layer. Therefore, if the upstream output indices I and the downstream input indices T are not aligned, only their overlap contributes to the direct selected-to-selected coupling isolated here. Let

$$U := I \cap T$$

denote the overlapping index set. Then $x_U^{(l+1)}$ is the only part of $x_I^{(l+1)}$ that is both influenced by $W_{I,J}^{(l)}$ and consumed by the next layer. Accordingly, the middle Jacobian in Eq. 10 effectively becomes

$$\frac{\partial x_K^{(l+2)}}{\partial x_I^{(l+1)}} = \frac{\partial x_K^{(l+2)}}{\partial x_U^{(l+1)}} = W_{K,U}^{(l+1)},$$

(and it is zero on $I \setminus T$), while

$$\frac{\partial x_U^{(l+1)}}{\partial W_{U,J}^{(l)}} = x_J^{(l)}.$$

Therefore, the gradient for the *effective* trainable portion $W_{U,J}^{(l)}$ becomes

$$\frac{\partial \mathcal{L}}{\partial W_{U,J}^{(l)}} = \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}} W_{K,U}^{(l+1)} x_J^{(l)}. \quad (11)$$

Eq. 11 shows that the gradient coupling from layer $(l+1)$ to layer l is governed by the overlap U : when $T \neq I$, $|U|$ can be small (or even $U = \emptyset$), weakening (or eliminating) this direct selected-to-selected term.

Aligned case. When the index sets are **aligned** ($T = I$), we have $U = I$, and Eq. 11 reduces to

$$\frac{\partial \mathcal{L}}{\partial W_{I,J}^{(l)}} = \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}} W_{K,I}^{(l+1)} x_J^{(l)}. \quad (12)$$

Crucially, under alignment ($T = I$), the connector weights $W_{K,I}^{(l+1)}$ are themselves contained in the selected trainable subspace of layer $(l+1)$ (since the trainable block is indexed by (K, T) and $T = I$). In contrast, when $T \neq I$, only the overlap $U = I \cap T$ participates in this two-layer gradient path: coordinates in $I \setminus T$ influence $x_I^{(l+1)}$ but are not consumed by the next layer, so their contribution to $\frac{\partial x_K^{(l+2)}}{\partial x_I^{(l+1)}}$ is zero. Equivalently, the effective coupling term reduces from $W_{K,I}^{(l+1)}$ to $W_{K,U}^{(l+1)}$ (Eq. 11), which can be substantially weaker when $|U|$ is small. As a result, alignment makes the gradient coupling between consecutive tuned subspaces more coherent, accelerating convergence as discussed in Section 3.2.2.

C.2 Block-wise Alignment: Aligning within Residual-connected Layers

The residual connection within the block transmits both activations and gradients directly across layers. Such residual paths preserve gradient magnitude and stabilize the training of deep neural networks (He et al., 2016b).

Two gradient branches (linear vs. residual). We focus on a single residual block whose final layer index is $l+2$. Recall the (un-aligned) forward definition:

$$x_I^{(l+1)} = W_{I,J}^{(l)} x_J^{(l)}, \quad (13)$$

$$x_K^{(l+2)} = W_{K,T}^{(l+1)} x_T^{(l+1)} + x_K^{(l)}. \quad (14)$$

Let $g_K := \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}}$. Back-propagation to the block input $x_J^{(l)}$ has two additive contributions:

$$\frac{\partial \mathcal{L}}{\partial x_J^{(l)}} = g_J^{\text{linear}} + g_J^{\text{residual}}.$$

(1) Linear branch. The second linear layer consumes $x_T^{(l+1)}$, while the first layer produces $x_I^{(l+1)}$. Hence, only the *overlap*

$$U := I \cap T$$

can transmit gradients through the two-layer path. Concretely,

$$\frac{\partial x_K^{(l+2)}}{\partial x_U^{(l+1)}} = W_{K,U}^{(l+1)}, \quad \frac{\partial x_U^{(l+1)}}{\partial x_J^{(l)}} = W_{U,J}^{(l)}.$$

Therefore,

$$g_J^{\text{linear}} = \frac{\partial \mathcal{L}}{\partial x_K^{(l+2)}} \frac{\partial x_K^{(l+2)}}{\partial x_U^{(l+1)}} \frac{\partial x_U^{(l+1)}}{\partial x_J^{(l)}} = g_K W_{K,U}^{(l+1)} W_{U,J}^{(l)}. \quad (15)$$

(2) Residual branch. The shortcut adds $x_K^{(l)}$ directly to $x_K^{(l+2)}$, so

$$\frac{\partial x_K^{(l+2)}}{\partial x_K^{(l)}} = \mathbf{I}.$$

However, we are differentiating w.r.t. $x_J^{(l)}$, not $x_K^{(l)}$. Thus, only coordinates in the intersection $J \cap K$ receive this identity gradient. Let $\Pi_{J \cap K}$ denote the projection that keeps entries in $J \cap K$ and zeros out the rest. Then

$$g_J^{\text{residual}} = \Pi_{J \cap K} g_K, \quad (16)$$

which is equivalent to the element-wise statement

$$(g_J^{\text{residual}})_r = \mathbf{1}[r \in J \cap K] (g_K)_r \quad \text{for } r \in J.$$

Combined gradient. Combining Eqs. 15 and 16, we obtain

$$\frac{\partial \mathcal{L}}{\partial x_J^{(l)}} = g_K \left(W_{K,U}^{(l+1)} W_{U,J}^{(l)} \right) + \Pi_{J \cap K} g_K, \quad U = I \cap T. \quad (17)$$

Aligned (block-wise) case. Block-wise alignment enforces $K = J$, so $\Pi_{J \cap K} = \mathbf{I}$ and the residual contribution becomes a full identity term on the tuned subspace. If we additionally apply layer-wise alignment inside the block ($T = I$), then $U = I$ and Eq. 17 simplifies to

$$\frac{\partial \mathcal{L}}{\partial x_J^{(l)}} = \frac{\partial \mathcal{L}}{\partial x_J^{(l+2)}} \left(\mathbf{I} + W_{J,I}^{(l+1)} W_{I,J}^{(l)} \right), \quad (18)$$

highlighting the explicit $\mathbf{I}$ (identity) shortcut term that appears only when the same index set is used on both sides of the residual connection ($K = J$). This direct term shortens the effective gradient route and stabilizes training, matching the empirical convergence improvements in Figure 4.

C.3 Global Alignment: Index Alignment across Residual Blocks

Residual connections benefit optimization not only within a single block but also across a sequence of blocks. To exploit this, we fix a single index set $J \subset \{1, \dots, d\}$ with $|J| = k$ and reuse it across all consecutive residual blocks. As a result, the forward (and backward) dynamics of the selected pathway evolve consistently in the same k -dimensional subspace, i.e., $x_J^{(t)} \in \mathbb{R}^k$. We denote by $\mathbf{I}_k \in \mathbb{R}^{k \times k}$ the identity matrix on this aligned subspace.

For a residual block indexed by b (cf. Eq. 3 in Section 3.2.2), the aligned-subspace forward propagation can be written as

$$x_J^{(b+1)} = x_J^{(b)} + \mathcal{F}^{(b)}(x_J^{(b)}), \quad (19)$$

where $\mathcal{F}^{(b)}(\cdot)$ denotes the block transformation restricted to the J -subspace (e.g., the two-layer layers induced by $W_{J,I}^{(b+1)}$ and $W_{I,J}^{(b)}$). Unrolling Eq. 19 for m successive blocks yields

$$x_J^{(b+m)} = x_J^{(b)} + \sum_{i=0}^{m-1} \mathcal{F}^{(b+i)}(x_J^{(b+i)}). \quad (20)$$

Back-propagation with a direct shortcut. Let $g_J^{(t)} := \frac{\partial \mathcal{L}}{\partial x_J^{(t)}}$ denote the gradient on the aligned subspace. By the chain rule for each residual block, we have

$$g_J^{(b)} = g_J^{(b+1)} \left(\mathbf{I}_k + \frac{\partial \mathcal{F}^{(b)}(x_J^{(b)})}{\partial x_J^{(b)}} \right), \quad (21)$$

which can be equivalently written as the *additive* decomposition

$$g_J^{(b)} = \underbrace{g_J^{(b+1)}}_{\text{direct shortcut}} + \underbrace{g_J^{(b+1)} \frac{\partial \mathcal{F}^{(b)}(x_J^{(b)})}{\partial x_J^{(b)}}}_{\text{via learned weights}}. \quad (22)$$

The first term in Eq. 22 comes purely from the residual shortcut and does not traverse intermediate weights.

Applying Eq. 21 repeatedly across m blocks yields

$$\frac{\partial \mathcal{L}}{\partial x_J^{(b)}} = \frac{\partial \mathcal{L}}{\partial x_J^{(b+m)}} \overleftarrow{\prod}_{i=0}^{m-1} \left(\mathbf{I}_k + \frac{\partial \mathcal{F}^{(b+i)}(x_J^{(b+i)})}{\partial x_J^{(b+i)}} \right), \quad (23)$$

where we define the reverse-ordered product as $\overleftarrow{\prod}_{i=0}^{m-1} A_i := A_{m-1} A_{m-2} \cdots A_0$. Expanding the product in Eq. 23 always contains the identity contribution (the shortcut-only path), which retains an explicit shortcut contribution throughout the J -projected recurrence. This direct gradient path:

- Shortens the effective back-propagation path length.
- Strengthens early-layer updates.
- Mitigates vanishing gradients even across many blocks.

D Attention-layer index alignment

In this section, we clarify (i) which attention parameters are selected by RAPA, (ii) why hidden-dimension index alignment still matters after forming the attention scores $S = QK^T/\sqrt{d_h}$, and (iii) how this interacts with the multi-head structure and the low-rank nature of QK^T .

D.1 Which attention parameters are selected?

In each multi-head attention block, we write the projections as

$$Q = xW_Q, \quad K = xW_K, \quad V = xW_V, \quad z = AV, \quad y = x + zW_O,$$

where $x \in \mathbb{R}^{d_{\text{model}}}$ is the residual stream and A is the attention weight matrix.

The core RAPA update always includes W_V and W_O , using the same input/output index sets as the residual stream and FFN layers. As an extension, we also align the index sets of W_Q and W_K with those of W_V/W_O . This additional Q/K alignment is not required for RAPA to function, but it further improves gradient flow, as shown by the ablation in Section D.4.

D.2 Why hidden index alignment still matters

After forming

$$S = \frac{QK^T}{\sqrt{d_h}} \in \mathbb{R}^{T \times T},$$

the hidden dimension is no longer explicitly visible: each entry S_{ij} is a scalar score between a query token i and a key token j . However, each score is still a bilinear function of the hidden coordinates:

$$S_{ij} = \frac{1}{\sqrt{d_h}} \langle q_i, k_j \rangle = \frac{1}{\sqrt{d_h}} \sum_{\ell=1}^{d_h} q_i[\ell] k_j[\ell].$$

RAPA’s index alignment acts *before* this contraction over the head dimension. Let $I \subset \{1, \dots, d_h\}$ be the aligned index set shared by W_Q, W_K, W_V, W_O and the FFN blocks, and decompose

$$q_i = (q_i^I, q_i^{\bar{I}}), \quad k_j = (k_j^I, k_j^{\bar{I}}).$$

RAPA only trains the slices of W_Q and W_K that produce q_i^I and k_j^I ; the remaining coordinates $q_i^{\bar{I}}, k_j^{\bar{I}}$ stay essentially frozen. For a small update, the change in the score is then

$$\Delta S_{ij} = \frac{1}{\sqrt{d_h}} \left(\langle \Delta q_i^I, k_j^I \rangle + \langle q_i^I, \Delta k_j^I \rangle \right),$$

because the non-aligned coordinates $\bar{I}$ do not move.

In other words: (i) only the aligned hidden coordinates I contribute to the *trainable* part of the score change ΔS , and (ii) those same coordinates I are exactly where W_V, W_O , and the FFN are updated. Thus, even though S lives in token–token space, the directions along which we are allowed to change S are controlled by which hidden indices we make trainable in W_Q and W_K . Aligning Q/K indices with V/O/FFN indices ensures that changes in the attention scores are produced by the same hidden channels that later carry the value and residual updates.

D.3 Gradient view and projection

The above intuition is reflected in a gradient-based view. For a single head, with

$$S = \frac{QK^T}{\sqrt{d_h}}, \quad A = \text{softmax}(S), \quad z = AV, \quad y = x + zW_O,$$

a small change in W_Q leads to

$$\Delta \mathcal{L}_Q = \frac{1}{\sqrt{d_h}} \langle G_Q, \Delta W_Q \rangle,$$

where the effective gradient on W_Q factors as

$$G_Q = x^T \mathcal{J}_{\text{sm}}(S)^T G(VW_O)^T K.$$

Here $G = \partial \mathcal{L} / \partial y$ and $\mathcal{J}_{\text{sm}}(S)$ is the Jacobian of the row-wise softmax.

RAPA restricts updates to a projected subspace,

$$\Delta W_Q \leftarrow P_{\text{in}}^{QK} \Delta W_Q P_{\text{out}}^{QK},$$

so the useful loss decrease is governed by

$$-\Delta \mathcal{L}_Q \propto \left\| P_{\text{in}}^{QK} G_Q P_{\text{out}}^{QK} \right\|.$$

Choosing P_{in}^{QK} and P_{out}^{QK} to share the same indices as the projections used for W_V, W_O , and the FFN means that the gradient directions that matter for Q/K and for V/O/FFN coincide. This is precisely the pathway alignment principle extended from FFNs to attention.

D.4 Empirical effect of Q/K alignment

We now make the empirical role of Q/K alignment explicit via an ablation on LLaMA2-7B (5-shot MMLU). We compare: (i) a variant that aligns only V/O and FFN indices, and (ii) a variant that additionally aligns Q/K indices.

Table 8: Q/K-alignment ablation on LLaMA2-7B (5-shot MMLU). PPL denotes training-loss perplexity.

Alignment	PPL ↓	Hum.	STEM	Soc.	Other	Avg.
V/O+FFN	4.62	49.1	42.1	60.4	59.0	52.2
V/O+FFN+Q/K	4.49	50.0	42.4	60.8	59.1	52.7

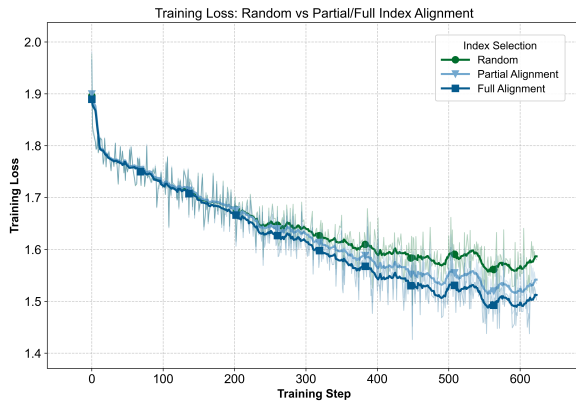


Figure 9: Training loss for LLaMA2-7B under three index-selection schemes. Random uses random indices. Partial Alignment uses aligned indices for V/O and FFN only. Full Alignment uses aligned indices for V/O, FFN, and Q/K. Thin curves show raw per-step training loss, and thick curves show a moving-average smoothed loss over steps.

On LLaMA2-7B, aligning only V/O and FFN indices yields 52.2% average MMLU accuracy, while additionally aligning Q/K indices increases this to 52.7% (+0.5 points) with a slightly lower perplexity (Table 8). Figure 9 shows the corresponding training-loss trajectories for the three index-selection schemes. As suggested by the analysis above, Q/K alignment has a smaller effect than aligning FFN and V/O (since Q/K influence the loss more indirectly through attention scores), but it still provides consistent gains on top of the core RAPA pathway alignment.

E Implementation Details

Algorithm 1 instantiates *all three* alignment rules in a single pass over the residual blocks.

Algorithm 1 Index Alignment of Trainable Weights (RAPA)

```

1: Input: residual blocks ( $\mathcal{B}_1, \dots, \mathcal{B}_L$ ), hidden size  $d$ , block size  $k$ , seed  $s$ 
2:  $J \leftarrow \text{RANDOMPICK}(d, k, s)$  // global residual indices
3:  $I \leftarrow \text{RANDOMPICK}(d, k, s+1)$  // shared intermediate indices
4: for  $b = 1$  to  $L$  do
5:   // first linear: train  $W_{I,J}^{(b,1)}$ , freeze others
6:   // second linear: train  $W_{J,I}^{(b,2)}$ , freeze others
7:   // enforce: block input/output indices are  $J$  (block-wise + global alignment)
8: end for
9: return aligned index sets  $(I, J)$  for all blocks

```

F Detailed Training Loss Curves

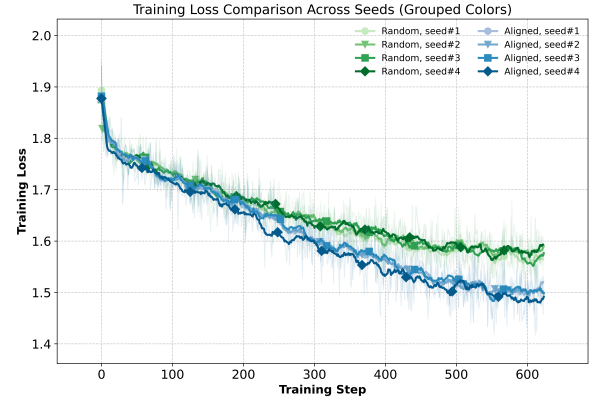


Figure 10: Detailed training loss curves for aligned vs. random index selection across four different random seeds. The grouped colors (blue hues for aligned, green hues for random) and unique markers identify each individual run. This demonstrates that the faster convergence of the aligned method is consistent and robust across various initializations.

G Extension of RAPA to MoE Residual Blocks

We show that the *index alignment* mechanism for dense Transformers extends to MoE architectures. By rewriting MoE residual blocks in an aligned subspace and applying the same residual-path analysis, we obtain an analogous “identity + aligned update” structure for expert FFNs, while the router acts as a smooth reweighting factor.

G.1 MoE residual blocks still admit the same pathway structure.

Notation. Each expert is a two-layer FFN parameterized by weights $W^{(b,e)}$, and we omit the weight argument for brevity: $\mathcal{F}_e^{(b)}(x) \equiv \mathcal{F}_e^{(b)}(W^{(b,e)}, x)$. For $J \subset \{1, \dots, d\}$ with $|J| = k$, let $x_J \in \mathbb{R}^k$ denote the restriction of $x \in \mathbb{R}^d$ to coordinates in J .

MoE residual block. At layer b ,

$$x^{(b+1)} = x^{(b)} + \sum_{e=1}^E p_e^{(b)} \left(x^{(b)} \right) \mathcal{F}_e^{(b)} \left(x^{(b)} \right),$$

where $p_e^{(b)}(\cdot)$ are router probabilities.

Aligned subspace restriction (trainable update confined to J). Choose an aligned index set $J \subset \{1, \dots, d\}$ with $|J| = k$ (and optionally $I \subset \{1, \dots, d_{\text{ff}}\}$ with $|I| = k$). We restrict each expert so that the *trainable* residual update is confined to J . Concretely, for a two-layer FFN expert

$$\mathcal{F}_e^{(b)}(x) = W^{(b,e,2)} \phi \left(W^{(b,e,1)} x \right),$$

we enforce trainability only on the aligned blocks $W_{I,J}^{(b,e,1)}$ and $W_{J,I}^{(b,e,2)}$ (shared (I, J) within the layer). Define the induced expert output on the J -subspace:

$$\mathcal{F}_{e,J}^{(b)}(x_J) := W_{J,I}^{(b,e,2)} \phi \left(W_{I,J}^{(b,e,1)} x_J \right).$$

Then the J -projected residual update is captured by the induced MoE update

$$\tilde{\mathcal{F}}^{(b)}(x) := \sum_{e=1}^E p_e^{(b)}(x) \mathcal{F}_{e,J}^{(b)}(x_J),$$

so that

$$x_J^{(b+1)} = x_J^{(b)} + \tilde{\mathcal{F}}^{(b)} \left(x^{(b)} \right).$$

Here, $\tilde{\mathcal{F}}^{(b)}(x)$ depends on x_J through $\mathcal{F}_{e,J}^{(b)}(x_J)$ and on the full x through $p_e^{(b)}(x)$.

Exact residual-stack Jacobian (chain rule). Stacking m such blocks from layers $b, \dots, b+m-1$ gives

$$x_J^{(b+t+1)} = x_J^{(b+t)} + \tilde{\mathcal{F}}^{(b+t)} \left(x^{(b+t)} \right), \quad t = 0, \dots, m-1.$$

Define the J -subspace Jacobian of a single block (partial Jacobian w.r.t. x_J):

$$A^{(b+t)} := \frac{\partial x_J^{(b+t+1)}}{\partial x_J^{(b+t)}} = I + \frac{\partial \tilde{\mathcal{F}}^{(b+t)} \left(x^{(b+t)} \right)}{\partial x_J^{(b+t)}}.$$

By the chain rule over the composition of residual blocks,

$$\frac{\partial x_J^{(b+m)}}{\partial x_J^{(b)}} = A^{(b+m-1)} A^{(b+m-2)} \dots A^{(b)}.$$

Equivalently, for any loss $\mathcal{L}$ depending on $x_J^{(b+m)}$,

$$\frac{\partial \mathcal{L}}{\partial x_J^{(b)}} = \frac{\partial \mathcal{L}}{\partial x_J^{(b+m)}} \left(A^{(b+m-1)} A^{(b+m-2)} \dots A^{(b)} \right).$$

Thus each block contributes an identity shortcut on x_J plus an aligned update confined to the same J -subspace.

G.2 Router gradients and stability.

Router-induced Jacobian terms. Using $\tilde{\mathcal{F}}^{(b)}(x) = \sum_{e=1}^E p_e^{(b)}(x) \mathcal{F}_{e,J}^{(b)}(x_J)$, the product rule yields

$$\frac{\partial \tilde{\mathcal{F}}^{(b)}(x^{(b)})}{\partial x_J^{(b)}} = \sum_{e=1}^E p_e^{(b)}(x^{(b)}) \frac{\partial \mathcal{F}_{e,J}^{(b)}(x_J^{(b)})}{\partial x_J^{(b)}} + \sum_{e=1}^E \mathcal{F}_{e,J}^{(b)}(x_J^{(b)}) \left(\nabla_{x_J^{(b)}} p_e^{(b)}(x^{(b)}) \right)^\top.$$

The first term is the usual expert Jacobian scaled by router weights, and the second term is the router-gradient contribution when $p_e^{(b)}$ depends on $x_J^{(b)}$. Importantly, $A^{(b)} = I + \partial \tilde{\mathcal{F}}^{(b)}(x^{(b)}) / \partial x_J^{(b)}$ still retains the “identity + aligned update” structure in the aligned J -subspace.

Stability in practice. In our experiments, we *do not* update router parameters: the router is kept fixed to the MoE baseline setting (i.e., no router training and no RAPA-specific constraint on router weights). We did not observe any instability attributable to RAPA under this setting. Hence, the core residual-alignment argument (identity + aligned MoE updates in J) continues to hold, while the router acts only as a smooth, subspace-preserving reweighting mechanism.

H 5-Shot MMLU Accuracy Comparisons under Various Ranks

Table 9: 5-shot accuracy (%) on MMLU dataset by rank and method.

Method	r	Param.	Accuracy (%)				
			Hum.	STEM	Soc.	Oth.	Avg.
Baseline	-	-	43.6	37.1	51.4	53.0	45.9
LoRA	N/A	-	-	-	-	-	-
DoRA	N/A	-	-	-	-	-	-
MosLoRA	N/A	-	-	-	-	-	-
PaCA	1	1.4M	44.6	38.1	54.0	53.8	47.2
SMT	-	1.4M	45.6	40.0	53.8	56.0	48.4
S ² FT	-	1.4M	43.7	36.9	51.5	52.8	45.9
RAPA	60	1.4M	46.7	40.6	56.3	56.4	49.6
LoRA	1	2.5M	46.7	42.6	57.9	57.6	50.6
DoRA	1	3.9M	47.5	41.8	58.8	58.0	51.0
MosLoRA	1	2.5M	48.2	40.9	59.8	58.6	51.4
PaCA	2	2.7M	46.5	39.8	55.7	54.7	48.8
SMT	-	2.5M	45.3	40.4	54.1	55.5	48.4
S ² FT	-	2.5M	46.0	40.3	55.6	55.1	48.8
RAPA	80	2.5M	47.6	41.8	58.8	58.4	51.1
LoRA	8	20.0M	48.5	41.2	57.3	56.5	50.6
DoRA	8	21.4M	48.7	42.3	58.3	57.6	51.3
MosLoRA	8	20.0M	46.6	42.2	60.8	57.4	51.1
PaCA	16	21.8M	48.7	41.7	58.7	57.6	51.2
SMT	-	20.0M	47.7	42.0	59.7	58.7	51.5
S ² FT	-	19.8M	44.6	38.6	53.4	53.9	47.3
RAPA	228	20.1M	50.1	42.6	60.8	59.1	52.8
LoRA	64	160M	51.2	43.1	61.7	60.1	53.6
DoRA	64	161M	51.0	43.6	61.8	60.1	53.7
MosLoRA	64	161M	50.4	42.8	61.5	59.5	53.1
PaCA	128	174M	50.7	43.3	63.5	60.1	53.9
SMT	-	160M	48.9	41.4	59.1	57.4	51.3
S ² FT	-	160M	50.2	43.3	62.0	60.2	53.4
RAPA	644	160M	51.2	44.3	62.8	60.6	54.3
LoRA	128	320M	50.6	43.3	61.9	59.5	53.4
DoRA	128	321M	51.8	43.1	61.9	60.1	53.9
MosLoRA	128	324M	50.7	42.4	60.5	58.2	52.6
PaCA	256	348M	50.7	43.3	63.5	60.1	53.9
SMT	-	320M	47.7	40.6	57.8	56.4	50.2
S ² FT	-	320M	51.5	43.6	62.4	60.8	54.1
RAPA	910	320M	51.4	44.4	64.3	60.2	54.6

I Additional Qwen3-4B Experiments

I.1 Controlled Full-Model Profiling

We profile the complete Qwen3-4B training graph rather than an isolated linear layer. RAPA, LoRA, and PaCA target the same seven projection types and run in fresh processes on one H100 80GB GPU with bf16 AdamW and zero weight decay. The fixed replay uses 256 examples from MMLU auxiliary training data; each method processes the same

two warm-up batches and 20 timed batches, totaling 14,255 non-padding tokens. The replay seed is 31, the batch size is 2, and the maximum sequence length is 512. A loss-equivalence gate requires the profiled and control runs to agree within 10^{-6} .

At the low budget, RAPA provides 9.53% higher throughput and 17.76% lower peak allocated memory than LoRA; at the high budget, the corresponding margins are 29.53% and 12.94%. The comparison with PaCA is mixed, with all observed throughput and memory differences between RAPA and PaCA remaining within 4.2% and 1.6%, respectively.

For five predeclared detailed steps, we partition RAPA device time into mutually exclusive CUDA-event leaves. The entries in Table 11 sum to 100% at each budget up to rounding. Selected-weight scatter is the largest RAPA-specific component, and its share increases with the trainable-parameter budget.

The FP32 trainable parameter footprint is 0.328/2.334 GiB at the low/high budgets, with 0.656/4.668 GiB of corresponding AdamW state. A complete corrected 6,248-example code-training pass yields 390.2 s, 7186.1 tokens/s, and 36.216 GiB for RAPA; 559.2 s, 5013.7 tokens/s, and 49.915 GiB for LoRA; and 371.3 s, 7552.0 tokens/s, and 35.275 GiB for PaCA. Since concurrent cluster load differed across these complete passes, we treat them as supporting measurements and base the quantitative comparison on the controlled replay above.

I.2 Expanded Fixed-Ledger Code Comparison

This panel is distinct from the task-specific Qwen3-4B rows in Table 5. It compares a broader set of selection and sparse-tuning methods under one fixed data ledger and training order. Starting from 50,002 code-training examples, we remove 585 examples with exact or 13-gram overlap with HumanEval or MBPP, leaving 49,417 examples. A deterministic SHA-256 ordering selects one fixed 6,248-example ledger, which every final training run consumes exactly once.

All runs use Qwen3-4B, bfloat16 AdamW, zero weight decay, gradient clipping at 1.0, dropout 0, a cosine schedule with 20 warm-up updates and a minimum learning-rate ratio of 0.1, and sequence/response limits of 1024/512. The physical and effective batch sizes are both 4, yielding 1,562 optimizer updates. The target modules

Table 10: Controlled full-model replay on Qwen3-4B. Low and high denote approximately 88M and 626M trainable-parameter budgets. Peak memory is peak allocated GPU memory.

Budget	Method	Trainable params	Throughput (tok/s)	Peak mem. (GiB)
Low	RAPA	88,100,352	3223.77	15.585
	LoRA	88,768,512	2943.15	18.950
	PaCA	88,473,600	3109.07	15.345
High	RAPA	626,577,408	2936.68	22.236
	LoRA	627,572,736	2267.21	25.541
	PaCA	625,950,720	3063.22	21.948

Table 11: RAPA device-time breakdown in the controlled Qwen3-4B replay (percent of measured device time).

Component	Low	High
Selected-weight scatter	32.127	57.870
Scale/cast	0.638	0.678
Dense projection forward	9.040	3.325
Activation gather	2.354	1.225
Custom forward residual	2.315	0.609
Dense input-gradient GEMM	8.515	5.590
Output-gradient gather	1.997	1.651
Selected-weight-gradient GEMM	4.433	3.812
Custom backward residual	2.057	1.327
AdamW	1.692	4.837
Base/other remainder	34.833	19.076

are q_proj , k_proj , v_proj , o_proj , $gate_proj$, up_proj , and $down_proj$. Seed-42 rows share the identical flattened example order. RAPA, LoRA, and PaCA use $(r, \alpha, LR) = (384, 192, 2e-4)$, $(43, 22, 3e-4)$, and $(80, 80, 2e-4)$, respectively, with approximately 88.1M trainable parameters in each case.

Evaluation uses greedy decoding on all 164 HumanEval and 257 MBPP problems (do_sample=false, one beam, and 512 generated tokens). A completion is correct only when it passes all provided unit tests, with a 10-second subprocess timeout. Code Macro is the unweighted mean of HumanEval and MBPP accuracy.

Table 12: Fixed-order Qwen3-4B code comparison. Search denotes examples consumed in a separate support-selection pass before final training. Baseline rows use seed 42; cross-method differences are descriptive.

Method	Search	HumanEval	MBPP	Macro
RAPA	0	66.46	70.43	68.45
LISA	0	67.07	69.65	68.36
LT-SFT	6,248	67.07	69.26	68.17
PaCA	0	65.24	68.09	66.67
SPEFT-Mag.	0	65.24	67.32	66.28
SpIEL-AG	0	63.41	68.48	65.95
SPEFT-Rand.	0	60.37	68.87	64.62
PaFi	0	59.76	66.54	63.15
LoRA	0	57.32	68.48	62.90
RMSNorm-only	0	57.32	67.32	62.32
SPEFT-Grad.	1,024	55.49	63.42	59.46
Diff Pruning	6,248	58.54	59.92	59.23
Fisher	1,024	45.12	54.86	49.99

RAPA’s three predeclared training seeds yield a Code Macro of 67.70 ± 1.60 (sample standard deviation). The baseline methods use only seed 42, so Table 12 is a common-seed, common-order point comparison rather than a paired training-seed significance test. LISA and RMSNorm-only auxiliary (Zhang and Sennrich, 2019) retain their natural instantaneous parameter budgets.

J Experimental Details

Table 13: Hyperparameters for the row/column weight-ratio sweep on LLaMA2-7B.

Fixed hyperparameters			
Method	RAPA		
Precision	16-bit mixed		
Trainable weights	4,096 / layer		
Optimizer	AdamW		
LR scheduler	Cosine		
Batch size	8		
Seq. length	512		
Epochs	1		
Target modules	Q, K, V, O		
Sweep configuration			
Col. rank	Row rank	Col.:row	α
1	4096	100/0	4
2	2048	92/8	8
		$\vdots$	
64	64	50/50	64
		$\vdots$	
2048	2	8/92	8192
4096	1	0/100	16384

Column rank is swept over powers of two from 1 to 4096, with row rank $\times$ column rank fixed to 4096. Representative settings are shown.

Table 14: Target module sets for different PEFT methods.

Index	Target modules
1	Q, K, V, O, Up, Down, Gate
2	Q, K, V
3	O, Down
4	Down

Table 15: Hyperparameters for fine-tuning LLaMA2-7B on MMLU. Common settings apply to all methods.

Common settings					
Precision	16-bit mixed				
Optimizer	AdamW				
LR	1e-4, 2e-4, 3e-4				
Scheduler	cosine				
Batch / Accum.	2 / 4				
Seq. length	512				
Warmup / Epochs	100 / 1				
Method-specific settings					
Method	Rank	α	Drop.	Target	Calib.
LoRA	8	32	0.1	1	–
DoRA	8	32	0.1	1	–
MosLoRA	8	32	0.1	1	–
PaCA	16	64	–	1	–
S ² FT	–	–	–	3	–
SMT	–	–	–	2	100
RAPA	228	912	–	1	–

Target refers to the module-set index in Table 14; Calib. denotes extra calibration steps.

Table 17: Hyperparameters for fine-tuning LLaMA2-7B on commonsense reasoning. Common settings apply to all methods.

Common settings					
Precision	16-bit mixed				
Optimizer	AdamW				
LR	2e-4, 3e-4				
Scheduler	cosine				
Batch / Accum.	8 / 1				
Seq. length	512				
Warmup / Epochs	100 / 1				
Method-specific settings					
Method	Rank	α	Drop.	Target	Calib.
LoRA	128	64	0.1	1	–
DoRA	128	64	0.1	1	–
MosLoRA	128	64	0.1	1	–
PaCA	256	128	–	1	–
S ² FT	–	–	–	3	–
SMT	–	–	–	2	100
RAPA	910	455	–	1	–

Target refers to the module-set index in Table 14; Calib. denotes extra calibration steps.

Table 16: Hyperparameters for fine-tuning LLaMA2-13B on MMLU. Common settings apply to all methods.

Common settings					
Precision	16-bit mixed				
Optimizer	AdamW				
LR	1e-4, 2e-4, 3e-4				
Scheduler	cosine				
Batch / Accum.	4 / 2				
Seq. length	512				
Warmup / Epochs	100 / 1				
Method-specific settings					
Method	Rank	α	Drop.	Target	Calib.
LoRA	128	64	0.1	1	–
DoRA	128	64	0.1	1	–
MosLoRA	128	64	0.1	1	–
PaCA	256	128	–	1	–
S ² FT	–	–	–	3	–
SMT	–	–	–	2	100
RAPA	1018	509	–	1	–

Target refers to the module-set index in Table 14; Calib. denotes extra calibration steps.

Table 18: Hyperparameters for instruction tuning Mistral-7B-v0.3 on OASST1. Common settings apply to all methods.

Common settings					
Precision	16-bit mixed				
Optimizer	AdamW				
LR	1e-3, 5e-4				
Scheduler	linear				
Batch / Accum.	2 / 8				
Seq. length	1024				
Warmup / Epochs	61 / 1				
Method-specific settings					
Method	Rank	α	Drop.	Target	Calib.
LoRA	64	1	0.1	1	–
DoRA	64	1	0.1	1	–
MosLoRA	64	1	0.1	1	–
PaCA	128	1	–	1	–
S ² FT	–	–	–	4	–
SMT	–	–	–	2	100
RAPA	602	8	–	1	–

Target refers to the module-set index in Table 14; Calib. denotes extra calibration steps.

Table 19: Restricted-VJP diagnostic on Qwen3-4B. Each selector seed is evaluated on four validation batches; entries aggregate the five paired selector seeds.

Metric	Shared J	Block-specific J_b
Final selected-cotangent norm	1.129586	2.603×10^{-14}
Geometric retention / block	0.863530	0.385449
Mean log-norm AUC	-0.316583	-15.362204

K Selected-Support Continuity Analysis

K.1 Restricted-VJP diagnostic

Diagnostic graph. Let $h_b = f_b(h_{b-1})$ denote Transformer block b , and let $B_b = \partial h_b / \partial h_{b-1}$ be its full Jacobian. Thus, B_b includes the dense pretrained weights, nonlinearities, and residual connection of the block. For a selected residual-coordinate set J_b , let P_{J_b} denote the corresponding orthogonal coordinate projector. Starting from a unit-normalized cotangent g_L at the final block, the diagnostic recursively applies

$$\tilde{g}_{b-1} = B_b^\top P_{J_b} g_b, \quad g_{b-1} = P_{J_{b-1}} \tilde{g}_{b-1}, \quad (24)$$

for $b = L, \dots, 1$. The projectors define the diagnostic recurrence; RAPA training retains the full dense forward and backward graph.

Protocol and metrics. We run the diagnostic on pretrained Qwen3-4B with $L = 36$ blocks and $K = 384$ selected residual coordinates. We compare (i) a shared set, $J_b = J$ for all blocks, and (ii) independently sampled block-specific sets J_b , each with cardinality K . Model weights, four validation batches, selected width, terminal cotangent, and selector seeds $\{7, 42, 123, 2024, 31415\}$ are paired; only the coordinate topology is changed.

At transition b , the retained fraction of the full block VJP is

$$r_b = \frac{\|P_{J_{b-1}} \tilde{g}_{b-1}\|_2}{\|\tilde{g}_{b-1}\|_2}. \quad (25)$$

We report the final selected-cotangent norm $\|g_0\|_2$, the geometric mean $\exp\left(L^{-1} \sum_{b=1}^L \log r_b\right)$, and the mean log-norm AUC over depth.

Results. The shared construction is higher for all five paired selector seeds. For the final norm, the paired shared-minus-block-specific difference has a 95% confidence interval of $[1.010355, 1.248818]$. The geometric retention difference is 0.478081, with a 95% interval of $[0.461618, 0.494545]$, and the mean log-norm-AUC difference is 15.045621,

with a 95% interval of $[14.654831, 15.436411]$. These results show that reusing one coordinate set preserves substantially more of each block VJP within a common selected support across depth. The corresponding masked-training behavior is evaluated separately in Figure 4, Table 1, and Appendix F.

K.2 Restricted second-moment recursion

The projector-defined graph in Eq. 24 admits a compact recursive interpretation. Let ξ_b be a zero-mean stochastic perturbation and define

$$A_b = P_{J_{b-1}} B_b^\top P_{J_b}, \quad \zeta_b = P_{J_{b-1}} \xi_b. \quad (26)$$

The restricted cotangent follows

$$g_{b-1} = A_b g_b + \zeta_b, \quad \mathbb{E}[\xi_b \mid \mathcal{F}_b] = 0. \quad (27)$$

Let

$$v_b = \mathbb{E}\|g_b\|_2^2, \quad \rho_b = \|A_b\|_2, \quad \nu_b = \mathbb{E}\|\zeta_b\|_2^2. \quad (28)$$

When the perturbation is conditionally uncorrelated with g_b , submultiplicativity gives

$$v_{b-1} \leq \rho_b^2 v_b + \nu_b. \quad (29)$$

Unrolling over L blocks yields

$$v_0 \leq \left(\prod_{b=1}^L \rho_b^2 \right) v_L \quad (30)$$

$$+ \sum_{i=1}^L \left(\prod_{b=1}^{i-1} \rho_b^2 \right) \nu_i. \quad (31)$$

The product term summarizes depthwise signal retention in the selected support, while the sum accumulates projected perturbations. Under an isotropic-noise approximation and a rank- K coordinate projector,

$$\nu_b \approx \frac{K}{d_{b-1}} \mathbb{E}\|\xi_b\|_2^2. \quad (32)$$

Because the shared and block-specific diagnostics use the same K , this approximation assigns them the same expected projected-noise scale. Their difference is therefore expressed through the restricted operators A_b and the realized retention measured in Table 19. The restricted-VJP experiment directly supplies this comparison by measuring the realized selected-support retention in the pretrained network.

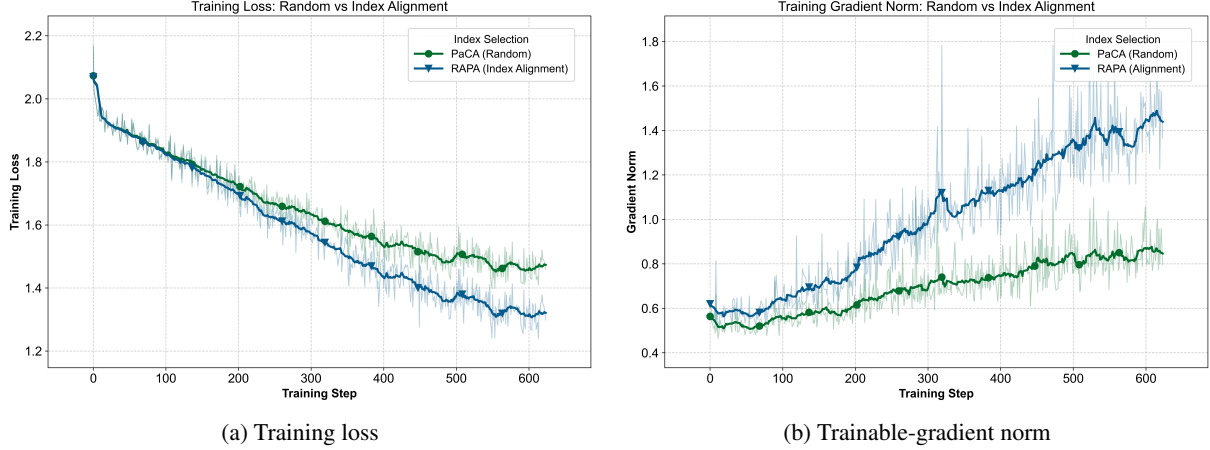


Figure 11: Complementary training dynamics under random and recursively aligned selected supports. The gradient norm is reported as a descriptive training statistic alongside the loss and accuracy ablations.

Connection to masked training. Figure 11 shows lower training loss and a larger trainable-gradient norm for aligned selection in the reported run. Appendix F repeats the loss comparison across four seeds, and Table 1 reports the corresponding accuracy ablation. The restricted VJP identifies the depthwise continuity of the selected support, while these masked-training results measure its observed optimization outcome.

L Scalability to LLaMA2-13B on MMLU

We further evaluate RAPA on LLaMA2-13B under matched trainable-parameter budgets. As shown in Table 20, RAPA achieves the highest average accuracy among the compared PEFT methods. Compared with the strongest adapter-free baseline, PaCA, RAPA improves the average accuracy by 0.6 points while maintaining comparable training time.

M Performance Comparison of Weight Selection in Adapter-Free Fine-Tuning

In this section, we compare three weight-selection strategies under a comparable trainable-parameter budget: (i) random selection, (ii) SMT-style gradient-calibrated selection over all target layers, and (iii) our aligned selection. This appendix is a diagnostic all-layer comparison of selection criteria, and is not intended as a reproduction of the original SMT setting. In the SMT paper, the main experiments allocate trainable parameters only to Q, K, and V vectors in attention layers.

Random selection. We uniformly sample trainable weights at random under a fixed parameter

budget. This approach is calibration-free and introduces no search overhead, but the selected subspace can be fragmented across layers.

SMT-style all-layer calibration. We apply SMT’s gradient-calibrated block scoring to all target weight matrices. Specifically, each target matrix is partitioned into fixed 256×256 square sub-blocks, and a 100-iteration full fine-tuning calibration is used to accumulate gradient magnitudes. We then select the top-ranked sub-blocks until the trainable-parameter budget matches the other methods, and fine-tune only those selected blocks. This all-layer variant intentionally stresses the calibration stage: compared with SMT’s QKV-only main setting, scoring all target weight matrices greatly expands the candidate space and exposes the memory and runtime overhead of calibrated selection.

Aligned selection (Ours). We select the same number of trainable weights but enforce *cross-layer index alignment*, so that the tuned coordinates form a consistent channel subspace along residual pathways. This alignment is *calibration-free*: it does not require any warm-up full fine-tuning to score weights, thereby avoiding precomputation overhead while maintaining adapter-free training efficiency.

Tables 21 and 22 compare runtime, peak memory, and downstream accuracy under comparable trainable-parameter budgets on a single NVIDIA H200 GPU. Random and Alignment require no search stage, thus matching the standard adapter-free fine-tuning cost. In contrast, calibration (SMT) adds a 100-iteration warm-up, increasing peak memory from 43 GB to 253 GB ($\sim 5.9\times$) and incurring approximately 10% runtime overhead. In

Table 20: MMLU 5-shot results on LLaMA2-13B. Type denotes adapter-based (AB) and adapter-free (AF) PEFT. [†] indicates methods with an additional calibrated selection stage; Mem and Time include this stage when applicable.

Type	Method	Params	Mem	Time	Accuracy (%)				
					Hums.	STEM	Social.	Other	Avg.
–	Baseline	–	–	–	53.2	44.2	62.8	60.8	55.0
AB	LoRA	501M	60.3G	5.9h	56.4	47.5	67.9	64.2	58.6
	DoRA	503M	80.3G	12.0h	57.0	47.8	67.6	64.6	58.9
	MosLoRA	505M	60.6G	6.1h	55.5	46.0	67.4	64.2	57.9
AF	PaCA	545M	52.7G	5.1h	57.2	47.5	67.5	65.1	59.0
	SMT [†]	501M	52.5G	7.3h	50.7	45.3	64.5	60.4	54.7
	S ² FT	501M	53.2G	5.3h	56.4	47.3	67.0	64.3	58.6
	RAPA (Ours)	501M	52.9G	5.2h	57.9	48.1	67.6	65.9	59.6

Table 21: All-layer adapter-free weight-selection comparison on MMLU (LLaMA2-7B). Time and memory report fine-tuning + search.

Method	Params	Mem	Time	Hums.	STEM	Social.	Other	Avg.
Random	320M	41G	78m	50.1	43.0	63.7	60.4	53.8
All-layer calibration	320M	253G	88m + 10m	49.3	42.8	60.6	58.8	52.4
Alignment (Ours)	320M	41G	78m	51.4	44.4	64.3	60.2	54.6

Table 22: All-layer adapter-free weight selection results on commonsense reasoning (LLaMA2-7B). Time and memory report fine-tuning + search.

Method	Params	Mem	Time	ARC-C	ARC-E	BoolQ	Hella	OBQA	PIQA	SIQA	Wino	Avg.
Random	320M	41G	116m	55.0	82.2	77.8	77.5	50.0	81.6	58.4	83.5	70.8
All-layer calibration	320M	253G	130m+10m	52.9	80.4	81.9	75.3	51.6	81.2	57.4	80.6	70.2
Alignment (Ours)	320M	41G	116m	55.2	81.8	86.5	77.6	51.8	82.0	58.4	82.0	71.9

Dataset	Train	Val.	Test	Avg. chars
MMLU FT corpus	99,842	0	0	1,688.5
CSR FT corpus	147,580	0	0	218.1
OASST1/Guanaco	9,846	0	518	1,447.9

Table 23: Statistics of the datasets used for fine-tuning. CSR denotes the commonsense reasoning corpus. OASST1 reports an available test split, but only the training split was used for fine-tuning.

addition, Alignment achieves the best accuracy on average: 54.6 on MMLU (vs. 53.8 Random / 52.4 Calibration) and 71.9 on commonsense reasoning (vs. 70.8 / 70.2).

N Statistics for Data

Table 23 summarizes the datasets used for fine-tuning. For MMLU and commonsense reasoning, we used the LMFlow text_only formatted corpora. The training script used the full training corpus without creating an internal validation split. For instruction tuning, we used the timdettmers/openassistant-guanaco version of OASST1.

For evaluation, we used lm-evaluation-harness (Sutawika et al., 2024). MMLU was

Benchmark	Train	Dev/Val.	Test	Eval ex.
MMLU	99,842	285 / 1,531	14,042	14,042
BoolQ	9,427	3,270	3,245	3,270
PIQA	16,113	1,838	3,084	1,838
Social IQA	33,410	1,954	–	1,954
HellaSwag	39,905	10,042	10,003	10,042
WinoGrande	40,398	1,267	1,767	1,267
ARC-Easy	2,251	570	2,376	2,376
ARC-Chal.	1,119	299	1,172	1,172
OpenBookQA	4,957	500	500	500

Table 24: Evaluation benchmark statistics. MMLU is evaluated in the 5-shot setting; commonsense benchmarks are evaluated in the zero-shot setting. For datasets without public test labels in the harness, the validation split is used as the evaluation split.

evaluated as a 5-shot multiple-choice benchmark over 57 subjects using the official test split. Commonsense reasoning was evaluated in the zero-shot setting over eight standard tasks: BoolQ, PIQA, Social IQA, HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge, and OpenBookQA.

O Broader Impacts

RAPA may help reduce the cost of maintaining many task-specific model variants. Unlike LoRA-style methods that introduce a separate adapter

branch, RAPA stores updates on a small square-aligned subset of pretrained weights, avoiding an additional inference path and reducing the per-task update footprint. This property is relevant to multi-tenant serving, on-device adaptation, and cost-sensitive deployment, although full serving-system evaluation remains future work.

Our DeepSeek-MoE-16B ablation further shows that the square-aligned selection principle is compatible with routed expert architectures: under a fixed router and matched trainable-parameter budget, square-aligned selection improves both training perplexity and 5-shot MMLU accuracy over square-random selection. This suggests that RAPA can serve as a lightweight mechanism for specializing expert modules without separate adapter branches. At the same time, deployment at scale requires further study of scheduling, storage formats, and interference among many task-specific updates. Since lower-cost adaptation may also reduce the barrier to adapting models for unintended or harmful uses, we limit our claims to standard public benchmarks and do not claim deployment-level safety.