

APPENDIX

A RELATED WORK

Low-Rank Adaptation. LoRA (Low-Rank Adaptation) (Hu et al., 2022) has emerged as a prominent technique for parameter-efficient fine-tuning (PEFT) (Li & Liang, 2021; Lester et al., 2021; Liu et al., 2021; Qiu et al., 2023; Liu et al., 2024b; 2022). By injecting lightweight, trainable low-rank decomposition matrices into frozen pre-trained weights, LoRA enables efficient task customization, especially in resource-constrained settings. Some LoRA variants (Liu et al., 2024a; Ding et al., 2023; Zi et al., 2023; Zhang et al., 2023b; Kalajdziewski, 2023) have been developed to enhance its generalization and robustness, while others (Zhou et al., 2024; Zhang et al., 2023a; Kopiczko et al., 2024; Azizi et al., 2024; Wang et al., 2024) address the increased memory overhead associated with scaling up model sizes. However, during training, these efficient LoRA variants still struggle with the substantial memory footprint of the original LLM parameters.

LoRA-related Compression. Model compression techniques like quantization (Han et al., 2015; Jacob et al., 2018; Nagel et al., 2019; Zhao et al., 2019; Yao et al., 2022; Park et al., 2022; Dettmers et al., 2022; Xiao et al., 2022; Frantar et al., 2022), sparsification (Molchanov et al., 2016; Liu et al., 2018; He et al., 2019; Hoefler et al., 2021; Frantar & Alistarh, 2023b; Liu et al., 2023; Bansal et al., 2022), and distillation (Hinton et al., 2015; Cho & Hariharan, 2019; Tang et al., 2019; Touvron et al., 2021; Hsieh et al., 2023; Gu et al., 2024b) have proven effective in reducing the memory footprint of LLM during training and inference. Naturally, the concept of compression has been adapted to LoRA to alleviate the substantial memory consumption dominated by pre-trained model parameters. In particular, LoRA-related quantization schemes (Dettmers et al., 2023; Xu et al., 2024; Li et al., 2024; Guo et al., 2024; Frantar et al., 2023; Chai et al., 2023) have been widely explored, but they still face the limitations of 1-bit precision, typically quantize weights to 4-bit to balance training efficiency with performance. Our work aims to push the boundaries of memory-efficient LoRA training by leveraging sparsification to achieve cost-effective performance improvements. Notably, existing LoRA-related sparsification works (Chen et al., 2023; Zhang et al., 2024a) focus on designing pruning algorithms to slim down models and use LoRA to recover the knowledge of pruned models, thereby producing compact but high-quality models. In contrast, LoRAM enables effective general pruning under high base-model sparsity, whereas (Gu et al., 2024a) focuses on task-specific LoRA sparsification with limited impact on base model memory reduction.

B EXPERIMENTAL DETAILS

Pre-train Corpus. To align the inconsistent knowledge between the pruned model during training and the original model during inference, we apply LoRAM to continual pre-training LLMs in a teacher-forcing manner (Bachmann & Nagarajan, 2024) on a mixed corpus of FineWeb (Penedo et al., 2024) and OpenWebMath (Paster et al., 2023). FineWeb, containing over 15TB of cleaned and deduplicated English web data from Common Crawl. OpenWebMath, extracted from over 200 billion HTML files on Common Crawl, provides high-quality mathematical text. Mixing these datasets enhances the pruned model’s capabilities in both general and mathematical domains.

Unless specified otherwise, we randomly sample 102,400 instances from both FineWeb and OpenWebMath to construct a mixed dataset with a sequence length of 512, yielding approximately 105 million tokens. The default training batch size is 128, allowing up to 1,600 update steps. We train without data repetition over a sufficiently large corpus to simulate a realistic pre-training scenario. Notably, this alignment process is a one-time, offline operation that model publishers can execute.

Fine-tuning Data. Following the fine-tuning scenario of LoRA (Hu et al., 2022), we primarily conduct supervised fine-tuning (SFT) on the OpenHermes-2.5 (Teknium, 2023) (referred to as OpenHermes). OpenHermes is a large-scale dataset constructed from synthetically generated instructions and chat samples, encompassing diverse sources such as Airoboros 2.2 (Wang et al., 2023), CAMELAI Domain Expert Dataset (Li et al., 2023), ChatBot Arena (GPT-4 Only) (Zheng et al., 2023), and more. To further demonstrate the general effectiveness of the LoRAM alignment process, we also evaluate LoRAM on the OpenOrca (Lian et al., 2023) dataset. OpenOrca is a widely used

instruction fine-tuning dataset where each data instance represents entries from the FLAN collection (Longpre et al., 2023), augmented by submitting the listed questions to either GPT-4 or GPT-3.5.

By default, we train SFT on the instruction dataset with a batch size of 128 and a sequence length of 512 for 400 steps, totaling approximately 26.2 million tokens. To effectively evaluate the overall fine-tuning performance, we assess the perplexity of the fine-tuned model on an out-of-domain test set. This out-of-domain test set is constructed by randomly sampling 2,000 instances from the Alpaca (Taori et al., 2023) test set, truncated to a sequence length of 512.

Downstream Task. We focus on the performance of LORAM in various downstream tasks, including mathematical reasoning, common sense reasoning, and code generation. All our downstream task evaluations are performed on lm-evaluation-harness⁵ and code-eval⁶ with VLLM⁷.

For mathematical reasoning, we benchmark the accuracy of baseline models using greedy decoding on MathQA (Amini et al., 2019) with a 1-shot setting and GSM8K (Grade School Math 8K) (Cobbe et al., 2021) with 8-shots, Chain of Thought (CoT) prompting and strict match MathQA is a large-scale dataset comprising 37k English multiple-choice math word problems, covering diverse math domains. It extends the AQuA-RAT dataset (Ling et al., 2017) by annotating problems with fully specified operational programs using a new representation language, building on the questions, options, rationale, and correct answers provided by AQuA-RAT. The GSM8K is a dataset of 8.5K high-quality, linguistically diverse grade school math word problems, designed to evaluate multi-step reasoning in basic arithmetic operations (+-x÷). We conduct evaluations on its 1.3K test set with *strict-match* to assess logical and mathematical reasoning in language models.

For commonsense reasoning (CSR), we report the average accuracy across six tasks—Arc Challenge & Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), and WinoGrande (Sakaguchi et al., 2021)—under 1-shot and greedy decoding settings. These benchmarks comprehensively assess the model’s ability to apply “common-sense” or world knowledge for reasoning, rather than relying on pattern recognition.

For code generation, we compare two pass rates, PASS@1 and PASS@10 (Kulal et al., 2019), on HumanEval (Chen et al., 2021) of each baseline in a zero-shot setting with sampling parameters of TEMPERATURE = {0.0, 0.2, 0.4, 0.6, 0.8}, and TOP_p = 0.95. The HumanEval dataset released by OpenAI consists of 164 handwritten Python programming problems, each with a function signature, docstring, body, and unit tests. Serving as a benchmark, HumanEval assesses models on a range of Python coding skills, from basic syntax to complex problem-solving, offering insights into their programming capabilities alongside language-focused tasks.

Sparsification & Quantization. LORAM incorporates two model compression techniques: sparsification, which generates a pruned model for low-rank matrix updates, and quantization, which forms QLORAM further to reduce the memory footprint of the pruned model. For sparsification, to validate the general effectiveness of LORAM, we benchmark its performance across various pruning strategies $P(\cdot)$. Specifically, we first establish a variant using randomly structured pruning and adapt LORAM to another three variants based on leading approaches: the structured pruning LLM-Pruner⁸ (Ma et al., 2023) and the non-structured (semi-structured & unstructured) pruning SparseGPT⁹ (Frantar & Alistarh, 2023a). These baselines are summarized below, with the corresponding configurations presented in Tables 4 to 6.

- **LORAM-RAND:** We adhere to the pruning settings of LORAM-STRU, modifying only by randomly removing weights instead of the original gradient-based pruning criterion.
- **LORAM-STRU:** We follow LLM-Pruner and employ a block-wise strategy for local structured pruning. Attention and MLP layers are treated as separate blocks, with non-critical coupling weights pruned based on gradient information at a uniform ratio. We retain the first four and last two layers of both blocks, focusing pruning on the intermediate layers.

⁵ <https://github.com/EleutherAI/lm-evaluation-harness> (MIT License).

⁶ <https://github.com/abacaj/code-eval> (MIT License).

⁷ <https://github.com/vllm-project/vllm> (Apache-2.0 license).

⁸ <https://github.com/horseee/LLM-Pruner> (Apache-2.0 license)

⁹ <https://github.com/IST-DASLab/sparsegpt> (Apache-2.0 license)

- **LoRAM-SEMI:** We utilize SparseGPT with a 4:8 semi-structured sparsity pattern to prune pre-trained weights across all model layers.
- **LoRAM-UNST:** We prune individual weights uniformly across layers using a predefined pruning ratio based on an unstructured version of SparseGPT.

For quantization $Q(\cdot)$, to further reduce memory usage during training, especially when dealing with models exceeding 70 billion parameters, we achieve QLoRAM by combining LoRAM with the LoRA-tailored quantization algorithm QLoRA (Dettmers et al., 2023). While LoRAM is compatible with the quantization of other customized LoRA methods (Xu et al., 2024; Li et al., 2024; Guo et al., 2024; Frantar et al., 2023; Chai et al., 2023), this falls outside the scope of this article.

Architecture & Hyperparameters. We adopt a LLaMA architecture with RMSNorm (Zhang & Sennrich, 2019) and SwiGLU activations (Shazeer, 2020; Zhao et al., 2022). We run all experiments with BF16 format to reduce memory usage. For all configurations, we default to a learning rate of $1e-3$. However, the downstream performance of models fine-tuned on OpenOrca is relatively sensitive to the learning rate. Therefore, in this evaluation, we tune the learning rates for each baseline within the range of $[1e-5, 1e-3]$ and report their respective optimal downstream scores. Specifically, we use $1e-5$ for the 7B LoRA and 13B & 70B LoRAM models, and $1e-4$ for the 13B LoRA model. All experiments run on NVIDIA A100-80GB GPUs with environments of CUDA 12.2, PyTorch 2.4.0, and Transformer 4.45.1. For LLaMA-2 herds, we set low-rank matrices \mathbf{B} and \mathbf{A} of rank $r = 8$ for \mathbf{W}_q , \mathbf{W}_k , \mathbf{W}_v , and \mathbf{W}_o in the attention layer, \mathbf{W}_{up} , \mathbf{W}_{gate} , and \mathbf{W}_{down} in the MLP layer, and the head embedding matrix $\mathbf{W}_{lm,head}$; for LLaMA-3 herds, we exclude the injection of the low-rank matrix of $\mathbf{W}_{lm,head}$.

Table 4: LoRAM configures on LLaMA-2-13B. Comparison of different pruning methods in terms of parameter reduction ratio (Reduction) and HBM footprint (GB) of pruned parameters (HBM), ignoring low-rank matrix overhead.

Method	#Orig. Params	Pruning Ratio	#Pruned Params	Reduction	HBM
LoRAM-Semi	13015864320	0.50	6738415616	$1.93\times$	12.55
LoRAM-Unst	13015864320	0.55	6037628912	$2.16\times$	11.25
LoRAM-Rand & Stru	13015864320	0.65	6005662720	$2.17\times$	11.19

Table 5: LoRAM configures on LLaMA-2-70B and LLaMA-3.1-70B with different pruning ratios.

Method	#Orig. Params	Pruning Ratio	#Pruned Params	Reduction	HBM
LoRAM-Rand & Stru	68976648192	0.65	28099436544	$2.45\times$	52.34
LoRAM-Rand & Stru	68976648192	0.75	21488738304	$3.21\times$	40.03
LoRAM-Rand & Stru	68976648192	0.85	16272924672	$4.24\times$	30.31
LoRAM-Rand & Stru	68976648192	0.95	9662226432	$7.14\times$	18.00
LoRAM-Rand & Stru	70553706496	0.85	17849982976	$3.95\times$	33.25

Table 6: QLoRAM configures on LLaMA-2-70B and LLaMA-3.1-70B with , demonstrating more aggressive parameter compression.

Method	#Orig. Params	Pruning Ratio	#Pruned Params	Reduction	HBM
QLoRAM-Rand & Stru	68976648192	0.65	7024859136	$9.82\times$	13.08
QLoRAM-Rand & Stru	68976648192	0.75	5372184576	$12.84\times$	10.01
QLoRAM-Rand & Stru	68976648192	0.85	4068231168	$16.95\times$	7.58
QLoRAM-Rand & Stru	68976648192	0.95	2415556608	$28.56\times$	4.50
QLoRAM-Rand & Stru	70553706496	0.85	4462495744	$15.81\times$	8.31

C VISUALIZATION OF DIMENSION EVOLUTION

To clearly illustrate the evolution of weight matrix dimensions across the multiple stages in the proposed scheme, we take LLM-Pruner (Ma et al., 2023) as an example in (e.g., LORAM-STRU) in Fig. 9, visualizing the transformation from $\mathbf{W}_0 \Rightarrow \mathbf{W}_0^P$, $\mathbf{W}_\Delta \Rightarrow \mathbf{W}_\Delta^P$, and $\mathbf{W}_\Delta^{P^*} \Rightarrow \mathbf{W}_\Delta^{R^*}$ under LoRAM with structured pruning. For LoRAM variants employing non-structured pruning, the parameter dimensionality remains unchanged during training due to the use of a mask matrix. Therefore, these visualizations are omitted.

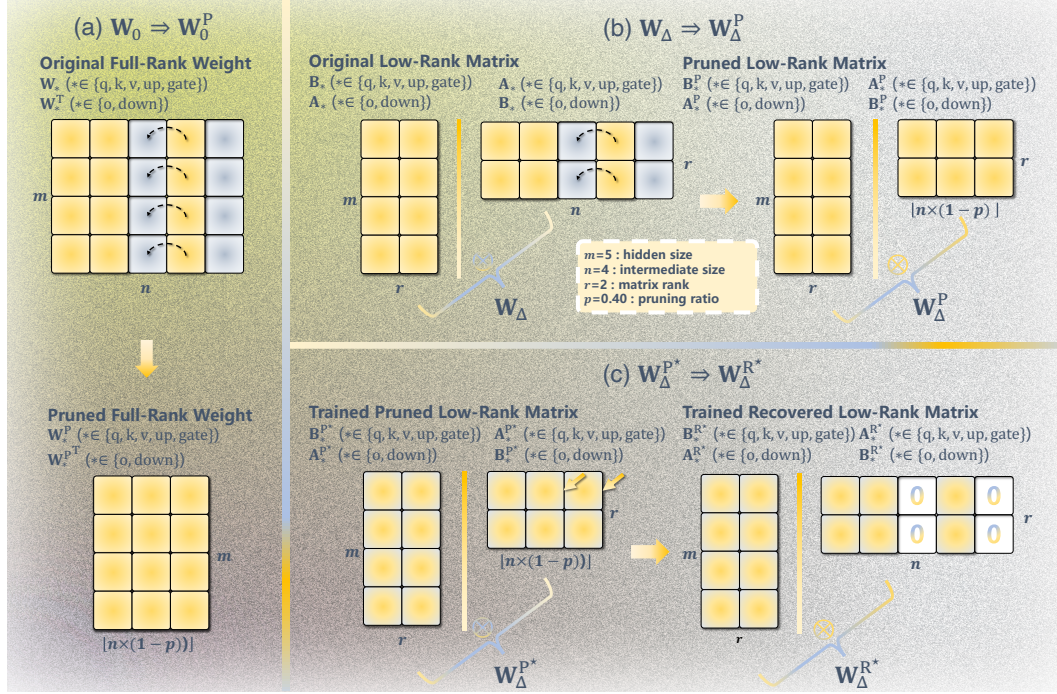


Figure 9: Dimensional evolution of the weight matrices: $\mathbf{W}_0 \Rightarrow \mathbf{W}_0^P$ (a), $\mathbf{W}_\Delta \Rightarrow \mathbf{W}_\Delta^P$ (b), and $\mathbf{W}_\Delta^{P^*} \Rightarrow \mathbf{W}_\Delta^{R^*}$ (c) during LORAM-STRU training. This includes updates for \mathbf{W}_q , \mathbf{W}_k , \mathbf{W}_v , and \mathbf{W}_o in the attention layer, as well as \mathbf{W}_{up} , \mathbf{W}_{gate} , and \mathbf{W}_{down} in the MLP layer.

D VISUALIZATION OF LOW-RANK MATRICES

In this section, we utilize the L_2 -norm to evaluate variations in low-rank matrices trained with different LORAM variants. This metric facilitates the visualization of captured features and allows for an analysis of LORAM’s effectiveness. Specifically, we examine the updated low-rank matrices in the self-attention and MLP layers of LLaMA-2-13B and LLaMA-2-70B, trained with LORAM variants on OpenHermes.

D.1 HEAD-WISE NORM OF ATTENTION

For the low-rank matrices in the attention layer, denoted as \mathbf{W}_{Δ^*} where $\Delta^* \in \{q, k, v, o\}$, we compute the L_2 norms for each attention head. Let H^* represent the number of heads. The L_2 norms for each head h (where $h = 0, 1, \dots, H^* - 1$) are defined as follows:

$$\|\mathbf{W}_{\Delta^*}^{(h)}\|_2 = \begin{cases} \|\mathbf{W}_{\Delta^*}[h, :]\|_2 & \text{if } \Delta^* \in \{q, k, v\} \\ \|\mathbf{W}_{\Delta^*}[:, h]\|_2 & \text{if } \Delta^* = o \end{cases}. \quad (10)$$

The results are visualized through heatmaps in Figs. 10 and 12, effectively illustrating the distribution of features captured by different attention heads.

D.2 LAYER-WISE NORM OF MLP

For the low-rank matrices in the MLP layers, denoted as \mathbf{W}_{Δ^*} where $\Delta^* \in \{\text{up}, \text{gate}, \text{down}\}$, we denote the number of layers as L . The average L_2 norm for a specific layer l (where $l = 0, 1, \dots, L - 1$) is computed as follows, excluding elements equal to zero using a mask, ensuring that only active parameters contribute to the average:

$$\|\mathbf{W}_{\Delta^*}^{(l)}\|_2 = \begin{cases} \frac{1}{m} \sum_{i=0}^{m-1} \|\mathbf{W}_{\Delta^*}^{(l)}[i, :]\|_2 \cdot \mathbb{I}(\mathbf{W}_{\Delta^*}^{(l)}[i, :] \neq 0) & \text{if } \Delta^* \in \{\text{up}, \text{gate}\} \\ \frac{1}{n} \sum_{j=0}^{n-1} \|\mathbf{W}_{\Delta^*}^{(l)}[:, j]\|_2 \cdot \mathbb{I}(\mathbf{W}_{\Delta^*}^{(l)}[:, j] \neq 0) & \text{if } \Delta^* = \text{down} \end{cases}. \quad (11)$$

Here, $\mathbb{I}(\cdot)$ denotes the indicator function, which returns 1 only when the corresponding element is non-zero, effectively excluding zero elements from the average calculation. The average norms for the MLP layers are visualized in Figs. 11 and 13, clearly depicting the trends in updating amplitudes across the various projections.

D.3 ATTENTION UPDATE PATTERNS

Layer Update Patterns in LORAM and LoRA. Figs. 10 and 12 reveal that both LoRA and LORAM display similar layer update behaviors. In any low-rank matrix \mathbf{W}_{Δ^*} where $\Delta^* \in \{q, k, v, o\}$, deeper colors predominantly concentrate in either shallow or deep layers, while middle layers receive relatively few updates. This suggests that training primarily focuses on optimizing the shallow layers to capture semantic information, with deeper layers refining this knowledge, rendering middle layers somewhat redundant.

More Uniform Projection Updates in LORAM. Figs. 10 and 12 further indicates that updates in the LoRA-trained low-rank matrices, particularly for \mathbf{W}_{Δ^*} , are relatively uniform, exhibiting substantial deep colors across multiple heads. In contrast, other matrices emphasize specific rows and heads. For instance, in the 70B model’s \mathbf{W}_{Δ^k} , only the heads in the uppermost layers experience significant updates, while lower layers show minimal changes. This suggests that the unpruned model retains rich knowledge, requiring only minor adjustments to a few heads in certain layers for task adaptation. Conversely, LORAM demonstrates a more uniform distribution of deep colors across each low-rank matrix, indicating that the pruned model must effectively utilize every limited neuron to capture knowledge, thereby enhancing downstream performance.

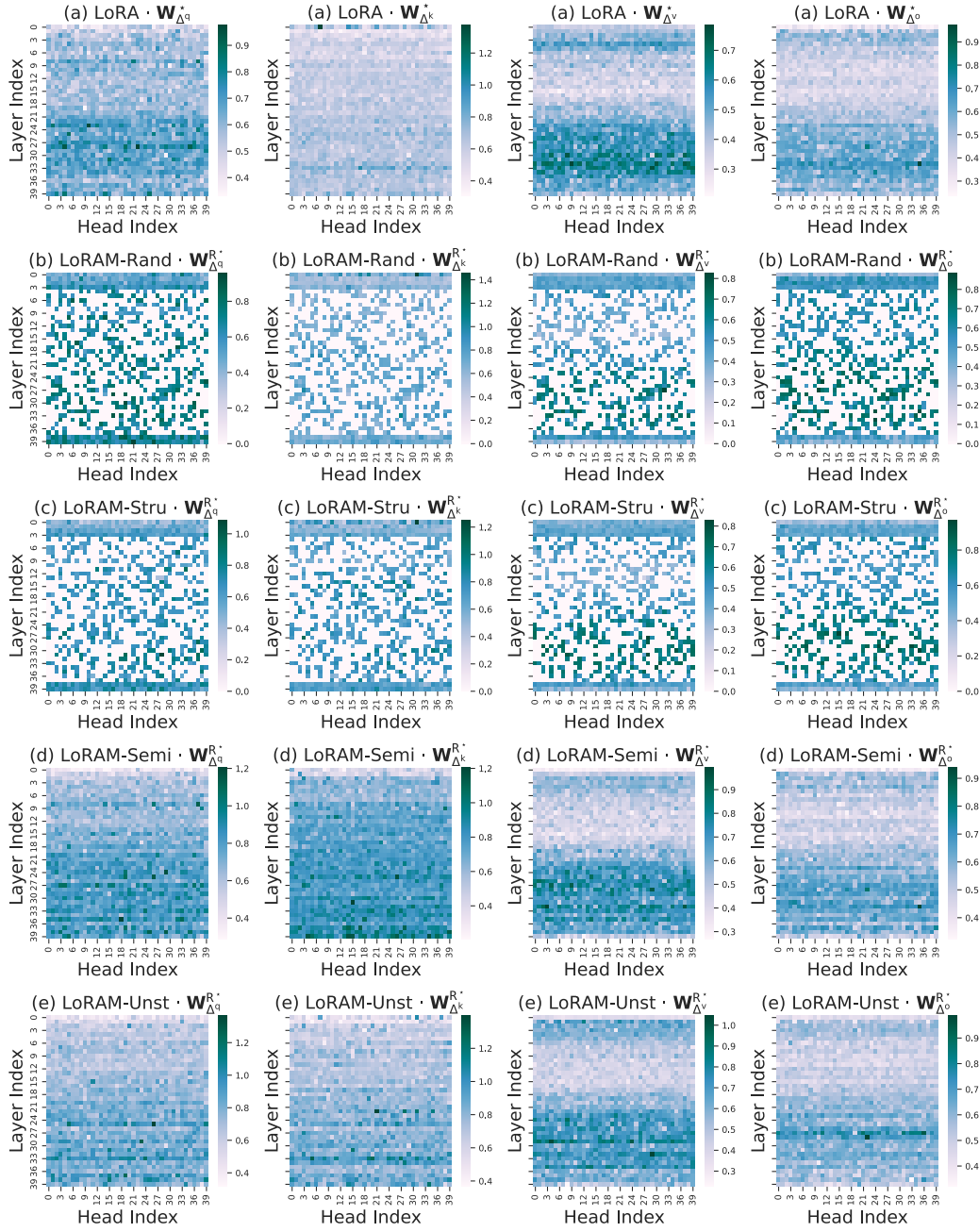
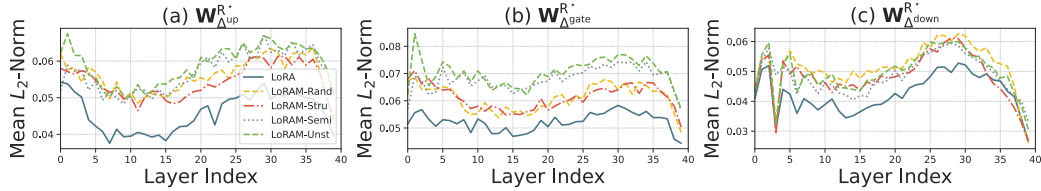


Figure 10: Visualization of low-rank matrices in the attention layers of LLaMA-2-13B.

Figure 11: Average L_2 norms of low-rank matrices in the MLP layers of LLaMA-2-70B.

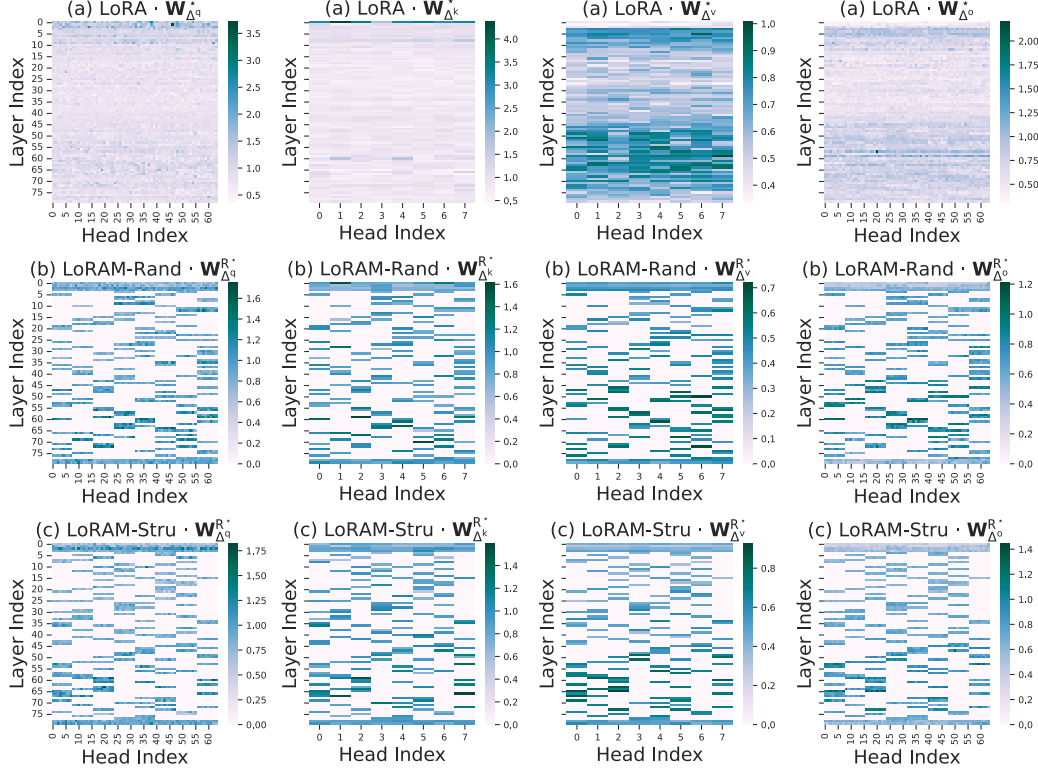
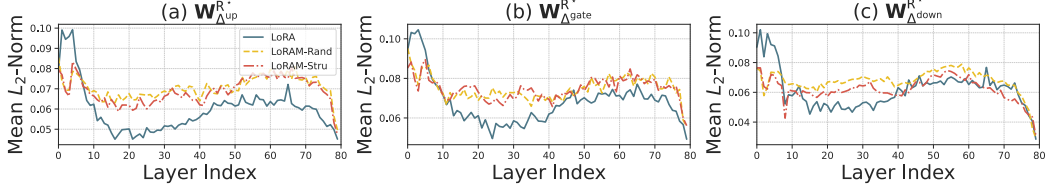


Figure 12: Visualization of low-rank matrices in the attention layers of LLaMA-2-70B.

Figure 13: Average L_2 norms of low-rank matrices in the MLP layers of LLaMA-2-70B.

D.4 MLP UPDATE PATTERNS

LoRAM Exhibits Greater Update Amplitude than LoRA. For both the 13B and 70B models, LoRAM consistently exhibits a greater update amplitude across each layer compared to LoRA, as shown in Figs. 11 and 13. This increased amplitude indicates that LoRAM is more effective in adjusting the weights in all layers, thus enhancing the adaptability and overall performance.

Distinct Update Trends in Layer Amplitudes. The amplitude changes reveal a distinct pattern in Figs. 11 and 13: first decreasing, then increasing, and finally decreasing again. Shallow layers (0-3) and deeper layers (25-35 for the 13B model and 50-75 for the 70B model) undergo intensive updates. This behavior indicates that model prioritizes foundational feature extraction in shallow layers and the refinement of complex representations in deeper layers. Such a strategic update distribution optimizes the learning process, ensuring effective capture of basic and advanced features.

D.5 ANALYSIS OF UNCHANGED WEIGHTS

Here, we try to analyze the unchanged weights to support the motivation of LoRAM.

Fine-Grained Visualizations. As the above visualization, we conducted detailed visualizations comparing the updated magnitudes of pruned and unpruned weights across layers. The results demonstrate that unpruned weights in both attention and MLP layers exhibit consistently smaller updates during fine-tuning as shown in Fig. 12, indicating their critical role in preserving the model’s capacity for inference.

Theoretical Perspective. The phenomenon can be explained by the gradient-based importance of these weights, which prioritize parameters with minimal updates but high sensitivity during recovery. These weights stabilize inference outputs, making them indispensable despite their limited fine-tuning updates.

Quantitative Evidence Our analysis reveals a strong correlation between weight update magnitudes and downstream performance. Pruning weights with smaller updates significantly degrades performance, highlighting their importance for inference and validating our intuition.

Impact on Large Models The selective pruning strategy shows notable benefits in larger models such as LLaMA-2-70B, where it outperforms random pruning by a substantial margin. Retaining critical parameters ensures effective task adaptation and generalization across diverse domains.

E PERFORMANCE OF SUB-TASKS IN CSR

We report the performance of six sub-tasks in CSR, with Figs. 14 and 15 showcasing the results for LoRAM-trained LLaMA-2-13B and LLaMA-2-70B, respectively. Our findings indicate that various LoRAM variants outperform core competitive benchmarks: for the 13B model, LoRAM surpasses both the untrained 13B and the LoRA-trained 7B, while for the 70B model, it exceeds the untrained 70B and the LoRA-trained 13B. This demonstrates that LoRAM consistently achieves performance gains across models of different scales while effectively reducing memory usage. Furthermore, selective weight contributions in the 70B model significantly enhance performance, as evidenced by LoRAM-STRU’s marked improvement, particularly in the challenging Arc Challenge multi-choice question-answering task. This suggests that LoRAM-STRU effectively identifies and leverages weight differences, focusing on the most trainable weights compared to LoRAM-RAND.

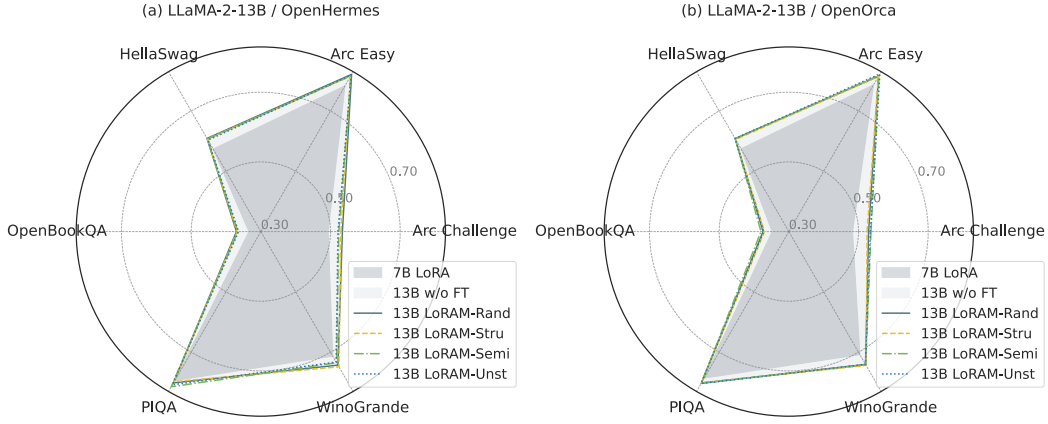


Figure 14: Performance of six CSR sub-tasks on the trained LLaMA-2-13B using LoRAM.

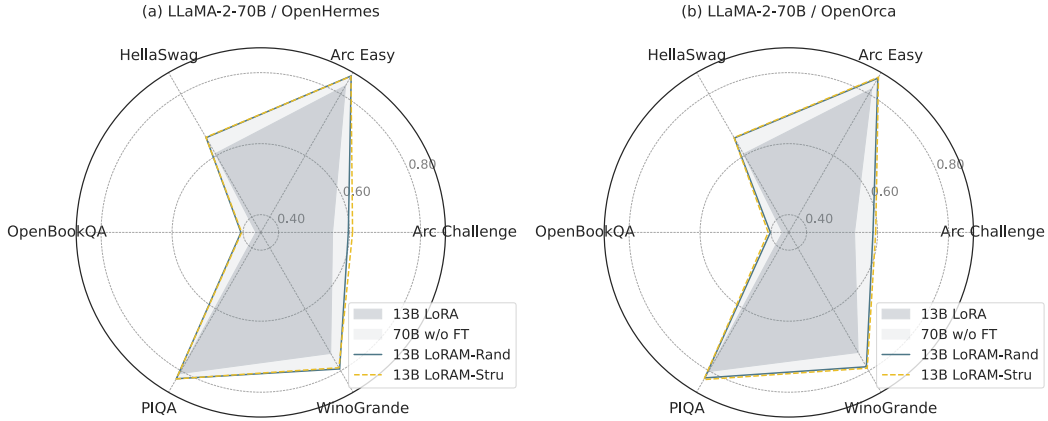


Figure 15: Performance of six CSR sub-tasks on the trained LLaMA-2-70B using LoRAM.

F ALGORITHM OF LORAM

Here, we present the complete algorithm of LORAM in Algorithm 1.

Algorithm 1 LORAM (Memory-Efficient LoRA Training)

Require: original full-rank pre-trained weight \mathbf{W}_0 , alignment corpus \mathcal{D}_A , and flags $\mathcal{F}^P, \mathcal{F}^A, \mathcal{F}^Q, \mathcal{F}^R$.

```

1: Offline  $\mathbf{W}_0^*$  Process Stage:
2: if  $\mathcal{F}^P$  then
3:    $\mathbf{W}_0^P = P(\mathbf{W}_0) = \mathbf{W}_0 \circ \mathbf{M}^P$  ▷ Pruned Full-Rank Weight Generation.
4:   if  $\mathcal{F}^A$  then
5:      $\mathbf{W}_0^{P,A} \leftarrow \text{argmin } \mathcal{L}_A(\mathcal{D}_A; \mathbf{W}_0^P)$  ▷ Pruned Full-Rank Weight Alignment.
6:     if  $\mathcal{F}^Q$  then
7:        $\mathbf{W}_0^{P,A,Q} = Q(\mathbf{W}_0^{P,A})$  ▷ Pruned Full-Rank Weight Quantization.
8:     end if
9:   else if  $\mathcal{F}^Q$  then
10:     $\mathbf{W}_0^{P,Q} = Q(\mathbf{W}_0^P)$ 
11:   end if
12: else if  $\mathcal{F}^Q$  then
13:    $\mathbf{W}_0^Q = Q(\mathbf{W}_0)$  ▷ Standard Quantization for LoRA
14: end if
15: Record the processing result of  $\mathbf{W}_0$  as  $\mathbf{W}_0^*, * \in \{\text{NULL}, P, Q, (P, Q), (P, A), (P, A, Q)\}$ .
16:
17: Online  $\mathbf{W}_\Delta^*$  Training Stage:
18: if  $\mathcal{F}^P$  then ▷ Pruned Low-Rank Matrix Generation.
19:    $\mathbf{W}_\Delta^P = \mathbf{B}^P \mathbf{A}^P = P(\mathbf{W}_\Delta) = \mathbf{W}_\Delta \circ \mathbf{M}^P = \mathbf{B} \mathbf{A} \circ \mathbf{M}^P$ 
20:   while TRAINING do ▷ Pruned Low-Rank Matrix Training.
21:     Update low-rank matrix via objective  $\mathcal{L}_{\text{SFT}}$  with the forward pass  $\mathbf{h} = \mathbf{x}\mathbf{W}_0^* + \mathbf{x}\mathbf{W}_\Delta^P$ .
22:     Return trained low-rank matrix  $\mathbf{W}_\Delta^{P*} = \mathbf{B}^{P*} \mathbf{A}^{P*}$ .
23:   end while
24:   if  $\mathcal{F}^R$  then ▷ Recovered Low-Rank Matrix Generation.
25:      $\mathbf{W}_\Delta^{R*} = \mathbf{B}^{R*} \mathbf{A}^{R*} = R(\mathbf{W}_\Delta^{P*}) = \mathbf{W}_\Delta^{P*} \circ (1 - \mathbf{M}^P)$  ▷ Structured LORAM
26:   else ▷ Non-structured LORAM
27:      $\mathbf{W}_\Delta^{R*} = \mathbf{B}^{R*} \mathbf{A}^{R*} = \mathbf{B}^{P*} \mathbf{A}^{P*}$ 
28:   end if
29: else
30:   while TRAINING do ▷ Standard LoRA Training.
31:     Update low-rank matrix via objective  $\mathcal{L}_{\text{SFT}}$  with the forward pass  $\mathbf{h} = \mathbf{x}\mathbf{W}_0^* + \mathbf{x}\mathbf{W}_\Delta$ .
32:     Return trained low-rank matrix  $\mathbf{W}_\Delta^* = \mathbf{B}^* \mathbf{A}^*$ .
33:   end while
34: end if
35: Record the trained low-rank matrix as  $\mathbf{W}_\Delta^*, * \in \{R^*, *\}$ .
36:
37: Online  $\mathbf{W}_0, \mathbf{W}_\Delta^*$  Inference Stage:
38: while INFERENCE with  $*$  is  $R^*$  do ▷ Recovered Low-Rank Matrix Inference.
39:   Perform inference with the forward pass  $\mathbf{h} = \mathbf{x}(\mathbf{W}_0 + \mathbf{W}_\Delta^{R*}) = \mathbf{x}(\mathbf{W}_0 + \mathbf{B}^{R*} \mathbf{A}^{R*})$ .
40: end while
41: while INFERENCE with  $*$  is  $*$  do ▷ Standard LoRA Inference.
42:   Perform Inference with the forward pass  $\mathbf{h} = \mathbf{x}(\mathbf{W}_0 + \mathbf{W}_\Delta^*) = \mathbf{x}(\mathbf{W}_0 + \mathbf{B}^* \mathbf{A}^*)$ .
43: end while

```

G TUNING OF LEARNING RATE

We provide additional details on the learning rate tuning process for full LoRA applied to LLaMA-2-7B and LLaMA-2-13B models, trained on the OpenHermes dataset. These experiments in Fig. 16 demonstrate that a learning rate of $1e-3$ consistently achieves the best perplexity across both in-domain and out-of-domain datasets, further validating the reliability of our comparison.

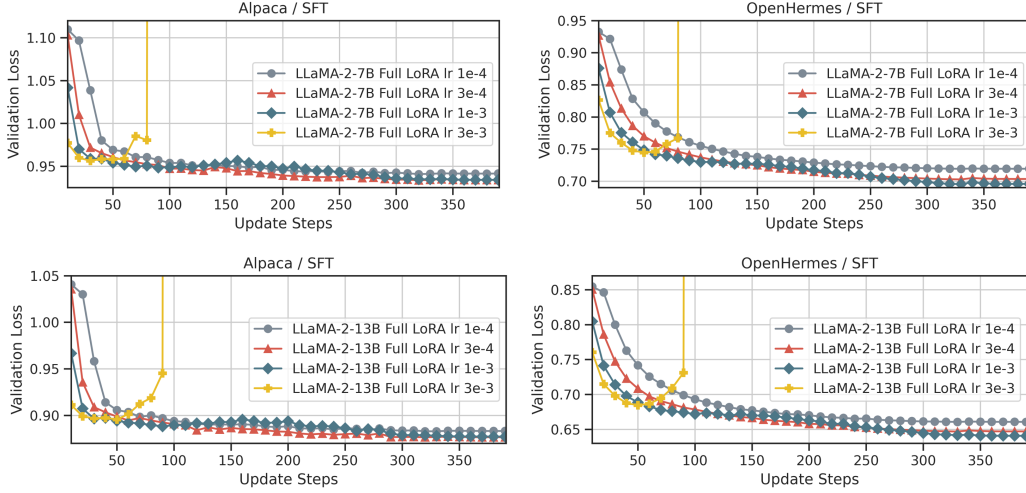


Figure 16: Learning rate tuning for LLaMA-2-7B and LLaMA-2-13B on OpenHermes using LoRA.

H PERFORMANCE OF DOMAIN-SPECIFIC TASK

To assess the effectiveness of LoRAM in domain-specific tasks, we conducted experiments on GSM8K (using the training set for tuning and the test set for evaluation), a mathematical reasoning benchmark known for its sensitivity to sparsification. Specifically, we trained LLaMA-3.1-70B using QLoRAM under various configurations.

The results, summarized in Table 7, highlight that LoRAM achieves excellent performance in this domain-specific setting. Notably, LoRAM-based models maintain high accuracy with substantial parameter reduction ratios, showcasing their robustness and efficiency in domain-specific tasks. These findings emphasize LoRAM’s broad applicability beyond general-purpose instruction fine-tuning.

Table 7: Evaluation of LoRAM on the GSM8K dataset for domain-specific fine-tuning. Results show accuracy (%) and parameter reduction ratios for different configurations.

LLaMA-3.1	GSM8K	Parameter Reduction Ratio
8B w/o Fine-Tuning	55.27	8.79×
8B LoRA (OpenHermes 400)	55.80	8.79×
70B w/o Fine-Tuning	75.28	1.00×
70B QLoRAM-Stru 400 (OpenHermes 400)	80.36	15.81×
70B QLoRAM-Stru 400 (GSM8K 100)	77.18	15.81×
70B QLoRAM-Stru 400 (GSM8K 200)	79.15	15.81×
70B LoRA (OpenHermes 400)	80.74	1.00×

I ANALYSIS OF LoRAM COST

Identifying the costs of LoRAM is indeed important, which is why we report both the number of training tokens used during the alignment phase and the parameter reduction ratios in the low-rank training phase. Below, we clarify the two stages of LoRAM:

Offline Knowledge Alignment Phase. The offline phase is task-agnostic and can be conducted by the model publisher prior to deployment, making its cost negligible for end users. To quantify the offline cost, we measured the number of training tokens (as in [Xia et al. \(2024\)](#)) rather than end-to-end latency, which can vary based on hardware configurations. As shown in Figure 5, LoRAM achieves significant performance gains using only 13 million tokens, demonstrating the efficiency of the alignment phase.

Online Low-Rank Matrix Training Phase. For the online phase, the memory and latency costs are primarily determined by the size of the base model parameters, which dominate resource consumption during training. To avoid redundancy in reporting, we focused on parameter reduction ratios instead of absolute time or memory usage.

Comparative Metrics for Online Training. Here, we provide additional metrics, including memory and latency comparisons for the online training phase. We conducted experiments using a workload of 1024 samples (batch size 128, micro-batch size 4, sequence length 512) randomly selected from OpenHermes. The results in Table 8 demonstrate that LoRAM with a structured pruning ratio of $2.17\times$ ($13\text{B} \rightarrow 6\text{B}$) achieves comparable peak memory, latency, and throughput to 7B LoRA, with only minor trade-offs. These differences arise due to the larger layer count in 13B LoRAM, introducing more non-GEMM operations, slightly affecting latency and throughput.

These results underscore the advantages of LoRAM’s design in achieving substantial resource efficiency without significant trade-offs in memory or latency.

Table 8: Comparison of peak memory (MiB), latency (s), and throughput (samples/s) during the online training phase for LoRAM and LoRA models. Results are based on a workload of 1024 samples (batch size 128, micro-batch size 4, sequence length 512).

LLaMA-2	#Model Params	Reduction Ratio	Memory	Latency	Throughput
7B LoRA	6.73B	$1.93\times$	30,517	134.27	7.626
13B LoRA	13.02B	$1.00\times$	51,661	206.07	4.969
13B LoRAM-Stru	6.01B	2.17 \times	29,799	147.86	6.925

J ANALYSIS OF CHANGES IN PERFORMANCE TRENDS

We analyze performance at two stages: after fine-tuning but before recovery, and after both fine-tuning and recovery.

After Fine-Tuning but Before Recovery. At this stage, the results of LoRAM align with prior work (e.g., SparseGPT, Wanda, and LLM-Pruner). Unstructured and semi-structured pruning consistently outperform structured pruning (see Fig. 6, solid lines). This trend holds true across both aligned and unaligned settings, with the performance order as follows: $\text{LORAM-SEMI} < \text{LORAM-UNST} < \text{LORAM-STRU} < \text{LORAM-RAND}$. The slight advantage of LORAM-SEMI over LORAM-UNST can be attributed to its smaller pruning ratio, which retains more parameters and mitigates performance degradation.

After Fine-Tuning and Recovery. Post-recovery results show that structured pruning outperforms unstructured pruning. This can be explained by two factors:

- **Preserved Structure for Recovery:** Structured pruning maintains the organization of the pruned weights into coherent structures (e.g., rows and columns in MLP layers, attention heads in attention layers), ensuring that activations after recovery are aligned with those of the original model. This alignment improves the recovery process.
- **Pruned Weight Quality:** The quality of pruned weights influences the recovery effectiveness. Structured pruning tends to remove less critical weights, leaving more recoverable parameters. In contrast, unstructured pruning can remove weights that are more difficult to recover, which negatively impacts performance post-recovery.

These results highlight the interplay between pruning and recovery, suggesting that structured pruning, despite initial performance disadvantages, facilitates more effective recovery.