LoRASuite: Efficient LoRA Adaptation Across Large Language Model Upgrades

Yanan Li[†,‡] **Fanxu Meng**[♮] **Muhan Zhang**[♮] **Shiai Zhu**[♯] **Shangguang Wang**[‡] **Mengwei Xu**[‡]
[†] CSCN [‡]Beijing University of Posts and Telecommunications
[♮]Peking University [♯]Unaffiliated
{YaNanLi,sgwang,mwx}@bupt.edu.cn
fxmeng@stu.pku.edu.cn, muhan@pku.edu.cn, shiaizhu2-c@my.cityu.edu.hk

Abstract

As Large Language Models (LLMs) are frequently updated, LoRA weights trained on earlier versions quickly become obsolete. The conventional practice of retraining LoRA weights from scratch on the latest model is costly, time-consuming, and environmentally detrimental, particularly as the diversity of LLMs and downstream tasks expands. This motivates a critical question: "How can we efficiently leverage existing LoRA weights to adapt to newer model versions?" To address this, we propose `LoRASuite`, a modular approach tailored specifically to various types of LLM updates. First, we compute a transfer matrix utilizing known parameters from both old and new LLMs. Next, we allocate corresponding layers and attention heads based on centered kernel alignment and cosine similarity metrics, respectively. A subsequent small-scale, skillful fine-tuning step ensures numerical stability. Experimental evaluations demonstrate that `LoRASuite` consistently surpasses small-scale vanilla LoRA methods. Notably, on backbone LLMs such as MiniCPM and Qwen, `LoRASuite` even exceeds the performance of full-scale LoRA retraining, with average improvements of +1.4 and +6.6 points on math tasks, respectively. Additionally, `LoRASuite` significantly reduces memory consumption by 5.5 GB and computational time by 78.23%.

1 Introduction

LoRA [1], a prominent parameter-efficient fine-tuning technique, has garnered significant attention for efficiently adapting pre-trained large language models (LLMs) to specialized downstream tasks using considerably fewer parameters than traditional full-parameter fine-tuning methods. A practical example is a mobile device employing an on-device LLM integrated with multiple app-specific LoRA modules to customize capabilities in mobile agent scenarios [2, 3].

However, frequent updates to LLM backbones, such as the periodic releases of new versions of Llama [4, 5] and Qwen [6, 7], quickly render previously trained LoRA weights obsolete. Consequently, developers face two problematic scenarios: (1) if the corresponding LoRA modules are not promptly updated, app-specific functionalities may degrade or fail; (2) if updated, the prevailing method typically involves retraining LoRA weights from scratch. This process is time-consuming, costly, and even environmentally unsustainable. For instance, experimental results from [8] indicate that fine-tuning a sub-billion-parameter model on a Google Pixel 7 Pro can take several hundred minutes. Similarly, research by [9] estimated that fine-tuning a sparse Mixtral model with two million queries on an NVIDIA H100 GPU costs approximately USD 3,460. Additionally, training a BERT model with 110 million parameters produces about 1,400 pounds of carbon dioxide equivalent—comparable to the emissions from a round-trip transcontinental flight in the United States

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

for one person [10]. This challenge is expected to escalate as the diversity of large models and the demand for various downstream tasks continue to increase. Thus, a natural question arises:

"How can we leverage the existing LoRA weights to adapt to the latest model version with less effort?"

To address this previously unexplored question, we first categorize the upgrades of mainstream LLMs into six explicit limitations: vocabulary size, hidden size, intermediate dimensions of up/down-projection layers, layer depth, attention head count, and attention type. These limitations hinder the direct reuse of prior LoRA weights. Next, we propose `LoRASuite`, a modular approach designed to efficiently resolve each of these limitations, in contrast to the current practice of retraining LoRA weights from scratch. For dimensional mismatches, `LoRASuite` utilizes known parameters from both the old and new LLM versions to compute a transfer matrix, enabling initial adaptation. To address differences in layer depth, we introduce a layer mapping algorithm that maps the LoRA weights from the i-th layer of the original model to the j-th layer of the upgraded model. Specifically, we apply Centered Kernel Alignment (CKA), a representational similarity method, to measure the similarities between layers, and then use dynamic programming to maximize the total sum of CKA similarities. To handle differences in attention head counts, we represent each attention head using input-independent interaction matrices, denoted as W^i_{QK} and W^i_{VO}. We then introduce a head mapping algorithm based on the Hungarian method, maximizing the aggregate sum of cosine similarities. However, the transformed LoRA parameters primarily result from matrix multiplications rather than backpropagation, which may cause numerical instability and reduced performance. To mitigate this, we introduce a small-scale, skillful fine-tuning stage to ensure the transformed LoRA parameters adapt effectively to the upgraded model while maintaining comparable performance.

We evaluated `LoRASuite` across multiple tasks, including commonsense reasoning and mathematical benchmarks, using various backbone LLMs. Experimental results show that `LoRASuite` consistently outperforms small-scale vanilla LoRA. Notably, for backbone LLMs such as MiniCPM and Qwen, `LoRASuite` even exceeds the performance of full-scale LoRA retraining, with average improvements of +1.4 and +6.6 points on math tasks, respectively. Additionally, `LoRASuite` significantly reduces memory usage by 5.5 GB and computational time by 78.23%.

Our key contributions are summarized as follows:

- To the best of our knowledge, this work is the first to systematically address the challenge of efficient LoRA weight transfer during LLM upgrades, analyzed from six explicit perspectives.
- We propose `LoRASuite`, a modular approach tailored to various types of LLM upgrades, including a layer-mapping strategy based on centered kernel alignment and an attention-head mapping approach utilizing the Hungarian algorithm.
- `LoRASuite` consistently outperforms small-scale vanilla LoRA across diverse tasks—including commonsense reasoning and mathematical evaluations—and, in certain scenarios, even surpasses full-scale LoRA retraining. Additionally, it significantly reduces both memory and time consumption.

2 Related Work

Interpretability of LLM. Mechanistic interpretability in deep neural networks has gained significant attention in recent years. Prior work has largely focused on analyzing hidden representations through techniques like probing [11], activation patching [12], and causal tracing [13], or on interpreting specific network weights by mapping model components to vocabulary space [14]. Our approach is based on the latter, and future work can further explore the integration of different methods. For a comprehensive discussion on mechanistic interpretability in transformer-based LLMs, see [15].

Efficient Fine-tuning. This work builds on LoRA, a classical and efficient fine-tuning method. Beyond LoRA, several other parameter-efficient fine-tuning (PEFT) techniques have been proposed. Adapter tuning [16] inserts lightweight trainable modules into the frozen backbone. Prompt tuning [17] appends trainable soft tokens to the input sequence, while prefix tuning extends this idea by inserting soft tokens into each layer's hidden representations. Hidden state tuning [18], exemplified by $(IA)^3$, rescales attention keys and values as well as activations in the feed-forward layers. Bias

Table 1: Summary of LLM upgrades with the same architecture (selected mainstream models). Highlighted cells indicate differences introduced during upgrades.

Architecture	Model	Vocab. Size	Hidden size	Interm. size	# of layers	# of heads	Attention
`LlamaForCausalLM`	Yi-6B	64k	4096	11008	32	32	GQA
	Yi-1.5-9B	64k	4096	11008	48	32	GQA
`GPTNeoXForCausalLM`	pythia-1b	50k	2048	8192	16	8	MHA
	pythia-1.4b	50k	2048	8192	24	16	MHA
`BloomForCausalLM`	bloom-560m	250k	1024	4096	24	16	MHA
	bloomz-1b1	250k	1536	6144	24	16	MHA
`LlamaForCausalLM`	Llama-2-7b	32k	4096	11008	32	32	MHA
	Llama-3-8b	128k	4096	14336	32	32	GQA
`Qwen2ForCausalLM`	Qwen-1.5-1.8B	152k	2048	5504	24	16	MHA
	Qwen-2.5-3B	152k	2048	11008	36	16	GQA
`MiniCPMForCausalLM`	MiniCPM-S-1B	73k	1536	3840	52	24	GQA
	MiniCPM-2B	123k	2304	5760	40	36	MHA

tuning [19] updates only the model's bias terms or a selected subset. Masked weight learning [20] applies a fixed sparse mask to the model's parameters. Input tuning [21] introduces an adapter at the embedding layer to adjust input representations.

Optimization for LoRA. LoRA [1], one of the most effective methods for parameter-efficient fine-tuning, has gained significant attention in recent years, leading to numerous variants and optimizations. For example, AdaLoRA [22] dynamically learns the required rank size for each model layer, LoSparse [23] integrates LoRA to prevent pruning from removing too many expressive neurons, DoRA [24] adapts LoRA based on magnitude and direction, and PiSSA [25] tunes the essential low-rank components while freezing high-rank, nonessential parts. LoRA+ [26] applies different learning rates to the A and B modules. Our work is orthogonal to these approaches, and future research can explore more efficient combinations of our method with these variants.

Knowledge Transfer between LLMs. Prior research on knowledge transfer between pre-trained language models has primarily focused on transferring prompt-based modules [27, 28, 29]. These methods typically align overlapping vocabularies between models, which is conceptually similar to our vocabulary and hidden-size alignment via linear combination described in Section 3.1. While several studies examine the transferability of LoRA or PEFT modules, they often rely on modifying the original fine-tuning process or introducing additional training steps, rather than enabling direct and efficient transfer. For example, Trans-LoRA [30] improves LoRA transferability through synthetic data generation, Trans-PEFT [31] enhances PEFT transfer via modified fine-tuning procedures, and MUSCLE [32] employs knowledge distillation to transfer adapters across models. Furthermore, some works [33, 34] investigate continual pre-training as a means of incremental model updating—analogous to implicit dataset upgrades in our setting—where direct transfer proves to be the most effective approach.

3 LoRASuite

Given a pre-trained weight matrix $W_o \in \mathbb{R}^{d_o \times d_{o_t}}$ in the original LLM, LoRA trained on a specific downstream corpus comprises two low-rank matrices, $A_o \in \mathbb{R}^{r \times d_{o_t}}$ and $B_o \in \mathbb{R}^{d_o \times r}$, where $r \ll min(d_o, d_{o_t})$. Typically, matrix B_o is initialized with zeros, and A_o is with Kaiming Uniform [35], ensuring $B_o A_o = 0$ initially.

Upon upgrading the LLM backbone to a newer version, as summarized in Table 1, six explicit factors—vocabulary size, hidden size, intermediate dimension of up/down-projection layers, layer depth, attention head numbers, and attention type—prevent direct reuse of existing matrices A_o and B_o for the new weight matrix $W_n \in \mathbb{R}^{d_n \times d_{n_t}}$.

Therefore, this section introduces several methods to adapt existing LoRA weights A_o and B_o to new matrices A_n and B_n without retraining from scratch when upgrading from W_o to W_n on the same corpus. We specifically focus on upgrades within LLMs of identical architecture, as defined in their respective `config.json` files. For instance, Phi-1.5 and Phi-2 share the same `PhiForCausalLM`

architecture, while Phi-3 has a different `Phi3ForCausalLM` architecture. Consequently, the activation function remains unchanged in our scenario. Exploring the impact of varying activation functions represents a promising direction for future research. In addition to the six explicit upgrade factors, we recognize that implicit upgrades, such as changes in pre-training datasets and post-training methods (e.g., RLHF), also influence model adaptation. Future research could further investigate the effects of these implicit factors.

3.1 Methodology

Vocabulary Size and Hidden Size. Vocabulary size and hidden dimension directly influence the embedding layer weights of a model, with hidden dimension mismatches notably restricting the direct reuse of existing LoRA weights. Based on variations in vocabulary size and hidden dimension, upgrades can be classified into three scenarios, as detailed in Table 2. In the simplest scenario—where only the hidden dimension changes—the transformation matrix W_h can be directly computed using the embedding weights of both the original and upgraded models. Assuming that all parameters from both models are available, the transfer matrix is specifically computed as $W_h = E_o^{-1}E_n$, where E_o and E_n denote the embedding weights of the original and updated models, respectively. For scenarios involving simultaneous changes in vocabulary size and hidden dimension, an additional intersection step is required to filter shared tokens, after which the transformation matrix is calculated using the filtered embedding weights.

Table 2: Classification for the upgrades concerning vocabulary size (VS) and hidden size (HS).

FROM	TO	VS	HS
PHI-1.5	PHI-2	—	✓
LLAMA-2-7B	LLAMA-3-8B	✓	—
MINICPM-1B	MINICPM-2B	✓	✓

Intermediate Size. When LoRA target modules contain up and down projection layers, mismatches in intermediate dimensions pose another significant challenge. Following a similar strategy, we leverage known parameters from both models to compute the transformation matrix W_i. Specifically, W_i is calculated as $W_i = W_o^{-1}W_hW_n$, where W_o and W_n represent the weights of the original and updated up/down projection layers, respectively.

Layer Depth. Inspired by prior studies on representational similarity, we employ centered kernel alignment (CKA) to quantify the similarity between corresponding layers of two LLMs. Following [36], we adopt a minibatch implementation of CKA to reduce memory usage. Based on the computed CKA similarities, we propose a novel dynamic programming-based layer mapping algorithm, which sequentially aligns corresponding layers to maximize the total similarity, subject to a predefined maximum offset constraint.

Algorithm 1 CKA-based Layer Mapping

Input: CKA similarity matrix $S \in \mathbb{R}^{l_o \times l_n}$, Original and upgrade layer depth l_o and l_n.
Output: $path[i][j]$ with the highest total CKA.

```
/# Store the maximum sum
Initialize dp[i][j] ← −∞ for all i, j
/# Store path information
Initialize path[i][j] ← 0 for all i, j
/# Limit maximum offset
for j = 0 to |l_n − l_o| do
  dp[0][j] ← S[0][j]
end for

for i = 1 to l_o − 1 do
  /# Limit maximum offset
  for j = i to i + |l_n − l_o| do
    max_v ← −∞
    max_i ← −1
    for k = i − 1 to j − 1 do
      /# Transfer equation
      if dp[i−1][k] + S[i][j] > max_v then
        max_v ← dp[i − 1][k] + S[i][j]
        max_i ← k
      end if
    end for
    dp[i][j] ← max_v
    path[i][j] ← max_i
  end for
end for
```

CKA [37] provides a robust method for quantifying neural network representations by measuring the similarity between pairs of activation tensors. Let $X \in \mathbb{R}^{m \times d_1}$ and $Y \in \mathbb{R}^{m \times d_2}$ denote the activations of two layers, with d_1 and d_2 neurons, respectively, across the same set of m examples. The elements of the Gram matrices $K = XX^T$ and $L = YY^T$ represent the pairwise similarities between examples based on X and Y. Using the centering matrix $H = I_n - \frac{1}{n}11^T$, the centered Gram matrices $K' = HKH$ and $L' = HLH$ remove mean effects. HSIC measures their similarity by flattening the matrices and computing the dot product:

$\text{HSIC}_0(K, L) = \text{vec}(K') \cdot \text{vec}(L')/(m-1)^2$. HSIC is invariant to orthogonal transformations and neuron permutations but remains sensitive to scaling. CKA normalizes HSIC to yield a similarity index between 0 and 1 that is invariant to isotropic scaling.

$$\text{CKA}(K, L) = \frac{\text{HSIC}_0(K, L)}{\sqrt{\text{HSIC}_0(K, K)\text{HSIC}_0(L, L)}} \tag{1}$$

However, computing CKA naively requires storing activations for the entire dataset, which is impractical for wide and deep LLMs. To reduce memory usage, [36] proposes estimating CKA by averaging HSIC scores across k minibatches $\text{HSIC}_0(K, L) \Rightarrow \frac{1}{k}\sum_{i=1}^{k}\text{HSIC}_1(X_i X_i^T, Y_i Y_i^T)$. In place of HSIC_0, which is a biased estimator of HSIC, the usage of an unbiased estimator of HSIC_1 [38] makes the value of CKA independent of batch size:

$$\text{HSIC}_1(K, L) = \frac{1}{n(n-3)}\left(\text{tr}(\tilde{K}\tilde{L}) + \frac{1^T\tilde{K}11^T\tilde{L}1}{(n-1)(n-2)} - \frac{2}{n-2}1^T\tilde{K}\tilde{L}1\right) \tag{2}$$

where $\tilde{K}$ and $\tilde{L}$ are obtained by setting the diagonal entries of similarity matrices K and L to zero. This minibatch-based HSIC estimation is equivalent to the bagging block HSIC method in [39] and converges to the full-dataset HSIC, as proven in [36].

Using the minibatch CKA defined above, we construct a similarity matrix S, where each element $S_{i,j}$ represents the CKA similarity between the i-th layer of the original model and the j-th layer of the upgraded model. Inspired by dynamic programming, we propose a layer-mapping algorithm that maximizes the total similarity in S by optimally aligning corresponding layers. To further guide the alignment, we impose an ordered mapping constraint, restricting the layer assignment within a threshold Δ_{layer}, which accounts for differences in layer depth between the original and upgraded models. The detailed procedure is outlined in Algorithm 1.

Head Number. Recall that the attention projection matrices W_Q, W_K, and W_V can be split along the column axis to H parts, denoted as W_Q^i, W_K^i, and $W_V^i \in \mathbb{R}^{d\times d/H}$, for $1 \le i \le H$. Similarly, the output projection matrix W_O is split along the row axis into H heads, with $W_O^i \in \mathbb{R}^{d/H\times d}$. For each head, we define two input-independent interaction matrix: $W_{QK}^i := W_Q^i {W_K^i}^T \in \mathbb{R}^{d\times d}$ and $W_{VO}^i := W_V^i {W_O^i}^T \in \mathbb{R}^{d\times d}$. Intuitively, W_{QK}^i captures the attention strength between token pairs, while W_{VO}^i models how attending to specific tokens influences the subsequent hidden state.

To address differences in attention head counts, we characterize each head using its interaction matrices and compute a similarity matrix via cosine similarity between heads from the old and new models, at the layers specified above. When interaction matrix dimensions differ, we align them by transforming to a common hidden size d_n (typically the new model's hidden size) using the embedding-based transformation W_h. We then determine the optimal head mapping by maximizing the sum of similarities, applying the Hungarian algorithm with the similarity matrix treated as a cost matrix. Further algorithmic details and theoretical analysis are provided in Appendix A.1.

Changes in attention mechanisms primarily affect the number of W_K and W_V heads. For instance, transitioning from GQA to MHA in MiniCPM-1B to MiniCPM-2B increases the number of K and V heads from 8 to 36. As in the forward pass, we replicate the K and V heads to match the number of Q heads before applying the head mapping algorithm.

Head Dimension. Following the attention head granularity used in the model's forward pass, we also adapt LoRA weights to the attention head level. As shown in Table 3, the hidden size and number of attention heads jointly determine the dimension of each head. Similar to earlier steps, we leverage known parameters from both the original and new models to adapt the LoRA weights. Specifically, we first compute the

Table 3: Classification for the upgrades concerning hidden size (HS), head number (HN), and their ratio head dimension (HD = HS / HN).

FROM	TO	HS	HN	HD
PYTHIA-1B	PYTHIA-1.4B	—	✓	✓
PHI-1.5	PHI-2	✓	—	✓
LLAMA-2-7B	LLAMA-2-13B	✓	✓	—
PYTHIA-410M	PYTHIA-1B	✓	✓	✓

Algorithm 2 Pseudocode of `LoRASuite` in PyTorch-like style.

```
/# Load the original LoRA weights.
B_o, A_o = load_peft_weights(ORIGINAL_LORA_PATH)
/# Calculate the hidden size transformation matrix.
W_h = embedding_transform(ORIGINAL_TOKENIZER, NEW_TOKENIZER)
/# CKA-based Layer Mapping, where S_{i,j} represents the CKA similarity
between the i-th layer of the original model and the j-th layer of the
upgraded model.
L_dict = cka_layer_mapping(S)
for each i and j in L_dict do
  ΔW_{o,i} = B_{o,i} A_{o,i}
  /* Compute the Hungarian-based head mapping between the ith projection layer of
  the original model W_{o,i} and the jth layer of the new model W_{n,j}.*/
  H_dict = hungarian_head_mapping(W_{o,i}, W_{n,j})
  for each h_o and h_n in H_dict do
    /# Head-level transformation based on Equation (3).
    ΔW_{n,j}^{h_n} = W_h^T ΔW_{o,i}^{h_o} W_{o,i}^{h_o T} W_h W_{n,j}^{h_n}
  end for
  B_{n,j}, A_{n,j} = SVD(ΔW_{n,j})
end for
```

weight update ΔW_o for each projection matrix by multiplying the corresponding B_o and A_o, then split it along the specified axis to extract updates for individual heads. Let $(W_Q^i)_o$ and $(\Delta W_Q^i)_o$ denote the original weights and weight updates for the i-th head in the original model. Based on the Hungarian algorithm, the weight update for the j-th head in the new model is computed as:

$$(\Delta W_Q^j)_n = W_h^T \cdot (\Delta W_Q^i)_o \cdot (W_Q^i)_o^T \cdot W_h \cdot (W_Q^j)_n \tag{3}$$

where, W_h, derived from the embedding weights of both models, is omitted if the hidden size remains unchanged. Incorporating original weights improves numerical stability, allowing the transformed LoRA to be directly applied in some cases. Finally, after all head mappings are completed, a single SVD step decomposes $(\Delta W_Q)_n$ into its low-rank B_n and A_n components.

We employ different strategies for layer and head mapping due to their distinct structural roles. For layer mapping, we adopt a dynamic programming approach inspired by studies on CNNs, where different layers capture different levels of abstraction, and higher layers build upon the representations of lower layers [40]. In contrast, attention heads within the same layer operate in parallel and independently [41], so we formulate head mapping as a bipartite matching problem and solve it using the Hungarian algorithm to obtain the optimal one-to-one correspondence.

3.2 Put All Together: A Final Recipe

Integrating the five components described above, the pseudocode of `LoRASuite` is presented in Algorithm 2. The CKA-based layer similarity and head similarity between the original and upgraded models can be precomputed offline, eliminating the need for repeated calculation during each adaptation.

Since the transformed LoRA parameters are generated via matrix multiplication rather than back-propagation, they may introduce numerical instability and degrade performance. To mitigate this, we introduce a lightweight fine-tuning step to help the upgraded model better adapt. Unlike standard Transformer `Trainer` that uses a linear learning-rate scheduler with warm-up to stabilize optimization from random initialization [42], our model is initialized with transformed parameters. Therefore, we omit the warm-up phase and increase the learning rate to compensate.

3.3 Complexity Analysis

The complexity of `LoRASuite` primarily arises from three key processes: layer mapping using the CKA method, attention head mapping utilizing the Hungarian algorithm, and the final SVD. As illustrated in Algorithm 1, the CKA-based layer mapping has a time complexity of $O(n_{layer}\Delta_{layer}^2)$,

Table 4: Performance on *math* tasks during the LLM upgrade from MiniCPM-S-1B to MiniCPM-2B. LFT denotes additional lightweight fine-tuning using small-scale data. Numbers in parentheses indicate the size of the fine-tuning datasets. "-" represents the performance of the vanilla model. Bold and underlined entries indicate the best and second-best results, respectively.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
MiniCPM-S-1B	-	23.80	5.17	23.62	7.43	17.72	20.17	16	16.27
	LoRA (10k)	29.62	54.67	31.5	12.13	8.27	31.93	18.2	26.62
MiniCPM-2B	-	23.29	48	27.56	12.28	14.57	22.27	19	23.85
	LoRASuite w/o LFT	23.04	47	28.15	12.36	14.17	22.27	20.7	23.96
	LoRA (100)	20.76	40.33	28.54	11.98	<u>15.35</u>	23.95	18.8	22.82
	LoRASuite w LFT (100)	**54.18**	<u>75.5</u>	**56.69**	**18.73**	14.17	**50.84**	**36.5**	**43.80**
	LoRA (10k)	<u>47.59</u>	**84.67**	<u>47.44</u>	<u>17.97</u>	**19.69**	<u>47.48</u>	<u>31.9</u>	<u>42.39</u>

Table 5: Performance on *commonsense* tasks when LLM upgrades from MiniCPM-S-1B to MiniCPM-2B. LFT denotes additional lightweight fine-tuning using small-scale data. The number in parentheses represents the scale of fine-tuning datasets. "-" represents the performance of the vanilla model. **Bold** and <u>underlined</u> entries indicate the best and second-best results, respectively.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
MiniCPM-S-1B	-	51.9	33.68	13.46	12.09	39.78	9.56	12.16	8.8	22.68
	LoRA (10k)	58.17	27.48	53.74	20.63	59.67	27.39	35.23	40.6	40.36
MiniCPM-2B	-	53.55	22.8	39.66	14.5	<u>52.33</u>	20.14	27.19	31.6	32.72
	LoRASuite w/o LFT	53.64	22.69	40.17	14.61	52.09	20.56	26.77	31	32.69
	LoRA (100)	**63.18**	<u>31.88</u>	42.37	15.77	50.99	28.5	<u>35.27</u>	40.8	38.60
	LoRASuite w LFT (100)	61.28	**43.91**	<u>48.21</u>	**24**	51.07	**32.42**	**39.02**	<u>38.8</u>	**42.34**
	LoRA (10k)	<u>62.35</u>	26.71	**50.61**	<u>21.17</u>	**60.77**	<u>27.73</u>	34.64	**42.8**	<u>40.85</u>

where n_{layer} denotes the number of layers in the original model, and Δ_{layer} represents the difference in the number of layers between the original and upgraded models. According to [43], the Hungarian algorithm exhibits a time complexity of $O(n_{head}^3)$, where n_{head} indicates the smaller number of attention heads present either in the original or upgraded model. Based on[44], the complexity of the SVD process to obtain the $B_{n,j}, A_{n,j}$ in j-th layer is $O(n_{hidden}^3)$. Combining these factors, the total computational complexity becomes $O(n_{layer}({\Delta_{layer}}^2 + {n_{head}}^3 + n_{hidden}^3))$. Since $n_{hidden} >> n_{head}, \Delta_{layer}$, the overall complexity is effectively dominated by the SVD and simplifies to $O(n_{layer} n_{hidden}^3)$.

4 Experiments

Implementation Details. Our implementation[1] builds on the publicly available Huggingface `transformers` and `peft` libraries, with modifications to support initialization from transformed weights. All experiments were conducted on a Linux server equipped with 80 Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz cores, 500 GB RAM, and 8 NVIDIA V100 GPUs.

Experimental Settings. For commonsense reasoning tasks, we evaluate BoolQ [45], PIQA [46], SIQA [47], HellaSwag [48],WinoGrande [49], ARC-e, ARC-c [50], and OBQA [51]. For math tasks, we evaluate AQuA [52], GSM8K [53], MAWPS [54], and SVAMP [55].

To ensure reproducibility, we use the same training and evaluation datasets as LLM-Adapters [56]. Both the original and upgraded models' LoRA (10k) are trained on the full dataset with 10k samples, whereas LoRA (100) and `LoRASuite` with LFT (100) are trained on identical, randomly selected subsets of the LoRA (10k) dataset with 100 samples.

Hyperparameter Settings. Unless otherwise specified, we use the default LoRA settings: rank 32, alpha 32, and dropout 0. A sensitivity analysis of these parameters is provided in Section 4.2. By default, the target modules include `q_proj`, `k_proj`, `v_proj`, and `o_proj`, with performance for other modules detailed in the Appendix A.3.

[1] https://github.com/YananLi18/LoRASuite

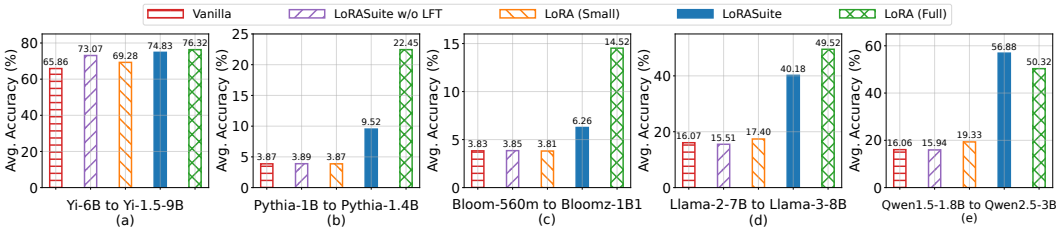

Figure 2: Average performance comparison on math tasks for different types of LLM upgrades.

4.1 Overall Performance

As shown in Table 1, upgrading from MiniCPM-S-1B to MiniCPM-2B involves all six explicit aspects. Thus, we primarily use MiniCPM to demonstrate LoRASuite's overall performance.

Accuracy. Tables 4 and 5 present performance results for math and commonsense tasks, respectively. "LoRASuite w/o LFT" denotes LoRASuite's performance using only the transformed MiniCPM-S-1B LoRA without further fine-tuning with small-scale data. "LoRASuite w LFT (100)" indicates LoRASuite with additional fine-tuning at a data scale of 100. For MiniCPM-2B, "LoRA (10k)" represents LoRA retraining using the complete dataset, while "LoRA (100)" serves as a baseline for retraining at the same scale. Under the default setting with rank 32 for MiniCPM-2B, the number of trainable parameters for LoRA (100), LoRASuite with LFT (100), and LoRA (10k) is the same—23.59M, which is approximately 0.78% of all model parameters. From the tables, we observe: First, LoRASuite achieves a notable average score of 43.80 on math tasks, nearly doubling the performance compared to same-scale fine-tuning (22.82) and surpassing full dataset retraining (42.39). Similarly, in commonsense tasks, LoRASuite outperforms both same-scale LoRA fin-tuning and full dataset retraining.

Additionally, "LoRASuite w/o LFT" exhibits performance nearly identical to the vanilla MiniCPM-2B model on both math and commonsense tasks. This result supports the hypothesis that transformation relying solely on matrix multiplication without backpropagation leads to numerical instability. It further validates the effectiveness of additional small-scale fine-tuning.

Memory and time consumption. Figure 1 demonstrates that LoRASuite significantly outperforms retraining a new LoRA model, achieving memory savings of 5.5GB and reducing time consumption by 78.23%. Although LoRASuite requires a small-scale fine-tuning stage, it yields a modest memory reduction of 5.5 GB due to three factors. First, data pipeline overhead and auxiliary states (e.g., DataLoader prefetching, dataset shuffling, and caching) scale with both the number of samples and training steps. Second, preloading the full dataset or pre-tokenized storage directly increases host DRAM usage. Third, for variable-length inputs, larger datasets increase the likelihood of long-sequence batches, resulting in excessive padding and transient memory spikes from dynamic allocation. For "LoRASuite w/o LFT," the memory reduction is more substantial, reaching 36.36% (15.34GB). This memory efficiency stems from loading only the parameters of the new and old models for matrix operations, thereby eliminating the overhead associated with optimizer states and gradients. The notable decrease in time consumption occurs because LoRASuite avoids the need for loading extensive datasets, performing backpropagation, or training for multiple epochs from scratch. Even with small-scale fine-tuning, the additional time is only approximately two minutes.

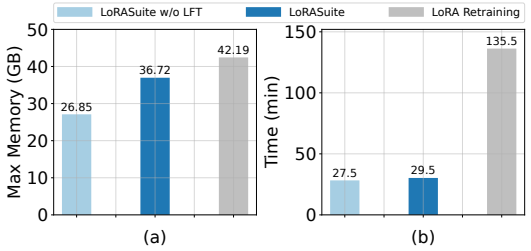

Figure 1: Memory and time comparison between LoRASuite and LoRA retraining.

Multiple LLM backbones. Figures 2 and 3 present the average performance comparisons across math and commonsense tasks, respectively, for different LLM backbones. The required amount of additional fine-tuning data for LoRASuite varies across different LLM backbones. Henceforth, we mainly use "LoRASuite" to refer to the variant with lightweight fine-tuning (LoRASuite w LFT (Small)), which also applies to the comparison method LoRA (Small). Our observations from these figures are as follows: First, LoRASuite consistently achieves performance comparable to or exceeding full-scale LoRA retraining in several scenarios. Second, LoRASuite significantly

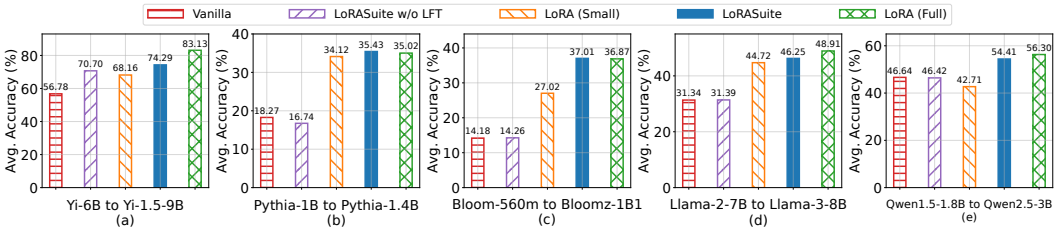

Figure 3: Average performance comparison on common tasks for different types of LLM upgrades.

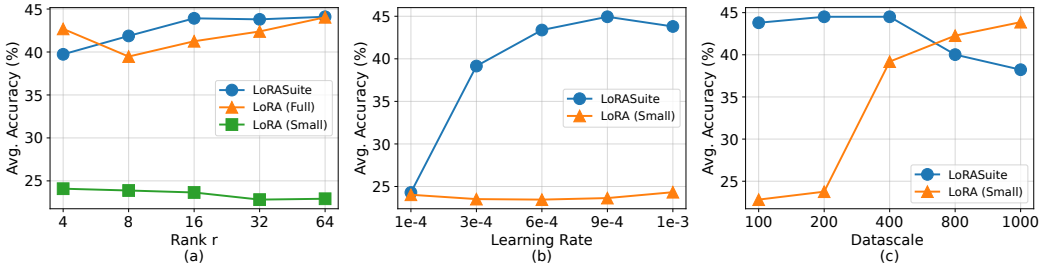

Figure 4: Average performance on math tasks under different settings for the MiniCPM-S-1B to MiniCPM-2B upgrade.

outperforms same-scale LoRA fine-tuning (LoRA (Small)) across all cases, with a notably pronounced improvement—nearly threefold—in math tasks when upgrading from Qwen-1.5-1.8B to Qwen-2.5-3B. Additionally, LoRASuite consistently surpasses "LoRASuite w/o LFT," highlighting lightweight small-scale fintuing's critical role in enhancing numerical stability. Finally, "LoRASuite w/o LFT" occasionally achieves performance similar to or slightly better than the vanilla model and "LoRA (Small)," suggesting that numerical stability may not be a significant concern for certain models. Future research could focus on developing more robust algorithms that avoid additional fine-tuning and accommodate diverse model update scenarios.

4.2 Sensitive Analysis

Different rank. Figure 4(a) presents the performance comparison between LoRASuite and baseline methods on math tasks across varying LoRA ranks. LoRASuite consistently outperforms vanilla LoRA in both small-scale and full-scale fine-tuning scenarios. Notably, at a LoRA rank of 4, LoRASuite achieves 93% of the performance obtained from training with 10K samples using only 100 samples. At a LoRA rank of 16, LoRASuite achieves its greatest improvement over full-scale LoRA, with an average increase of approximately 2.67 points.

Different LFT learning rate. The experimental results depicted in Figure 4(b) indicate that LoRASuite is highly sensitive to changes in the learning rate compared to vanilla LoRA (Small). Specifically, when the learning rate is 1e-4, LoRASuite marginally outperforms LoRA (Small) by only 0.27 percentage points on average. However, at a learning rate of 9e-4, LoRASuite substantially outperforms LoRA (Small) by an average of 21.30 points, representing approximately a twofold improvement in performance. This sensitivity likely arises because LoRASuite leverages previously trained LoRa weights to quickly identify crucial task-specific features, creating a steeper optimization landscape and greater dependence on the chosen learning rate.

Different LFT data scale. When trained on small datasets, LoRASuite maintains stable performance and consistently outperforms LoRA (Small). However, its performance declines with increasing dataset size, likely due to LoRASuite's reliance on historically trained LoRA weights, making it more susceptible to overfitting.

4.3 Application to Other PEFTs

Table 6 and Table 7 report the average performance of two PEFT methods, AdaLoRA and DoRA, on math tasks when upgrading the LLM from MiniCPM-S-1B to MiniCPM-2B. As shown, LoRASuite yields a substantial improvement for DoRA adaptation: while the small-scale vanilla LoRA achieves only 23.54% accuracy, LoRASuite improves this by more than 1.8×, reducing the performance

Table 6: Performance on *math* tasks during the LLM upgrade from MiniCPM-S-1B to MiniCPM-2B with AdaLoRA. Numbers in parentheses indicate the size of the fine-tuning datasets. Bold and underlined entries indicate the best and second-best results, respectively.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
MiniCPM-S-1B	-	23.80	5.17	23.62	7.43	17.72	20.17	16	16.27
	AdaLoRA (10k)	48.61	76.67	58.66	27.29	17.32	52.1	36.2	45.26
MiniCPM-2B	-	23.29	48	27.56	12.28	14.57	22.27	19	23.85
	LoRASuite w/o LFT	42.28	67.17	50	25.63	13.78	33.19	34.2	38.04
	AdaLoRA (1k)	52.41	71.67	59.66	28.81	<u>19.29</u>	52.94	35.8	45.80
	LoRASuite w LFT (1k)	**59.75**	**82.67**	<u>65.35</u>	<u>31.46</u>	**20.47**	<u>56.3</u>	**41.20**	**51.03**
	AdaLoRA (10k)	<u>55.44</u>	<u>82.17</u>	**65.55**	**31.69**	14.57	**63.03**	<u>40.80</u>	<u>50.46</u>

Table 7: Performance on *math* tasks during the LLM upgrade from MiniCPM-S-1B to MiniCPM-2B with DoRA. Numbers in parentheses indicate the size of the fine-tuning datasets. Bold and underlined entries indicate the best and second-best results, respectively.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
MiniCPM-S-1B	-	23.80	5.17	23.62	7.43	<u>17.72</u>	20.17	16	16.27
	DoRA (10k)	31.90	60.67	31.89	12.59	12.60	32.35	20.80	28.97
MiniCPM-2B	-	23.29	48	27.56	12.28	14.57	22.27	19	23.85
	LoRASuite w/o LFT	22.03	47	29.13	12.51	16.14	23.11	19.30	24.17
	DoRA (100)	22.53	44	27.76	11.90	15.75	23.53	19.30	23.54
	LoRASuite w LFT (100)	**55.44**	<u>71.33</u>	**55.91**	**19.33**	11.81	**49.16**	**37.20**	<u>42.88</u>
	DoRA (10k)	47.09	**87**	<u>48.43</u>	<u>17.66</u>	**20.47**	<u>47.06</u>	<u>34.60</u>	**43.19**

gap with full-scale LoRA to less than 1%. In contrast, the improvement offered by LoRASuite over LoRA (Small) is less pronounced for AdaLoRA, although it still surpasses full-scale LoRA retraining. This discrepancy is likely due to rank incompatibilities between corresponding layers of the original and upgraded models.

5 Conclusion and Discussion

We propose LoRASuite, a modular framework designed to handle diverse types of LLM upgrades. To the best of our knowledge, this is the first work to systematically address the challenge of efficient LoRA weights transfer during LLM upgrades across six explicit perspectives. LoRASuite consistently outperforms small-scale vanilla LoRA across diverse tasks and, in certain scenarios, even surpasses full-scale LoRA retraining with both memory and time reduction.

Limitations. Currently, LoRASuite requires an additional small-scale fine-tuning step to achieve superior performance. Future research could investigate strategies to eliminate this step without compromising performance, further minimizing memory usage. Additionally, our study primarily addresses explicit upgrade aspects of LLMs; future investigations could extend to implicit upgrades, such as variations in pre-training datasets and post-training methods, with knowledge editing methods [13, 57, 58]. Lastly, exploring LoRA adaptation methods applicable to different architectures remains an open avenue for future work.

6 Acknowledgments and Disclosure of Funding

The authors thank the anonymous reviewers for their insightful feedback and Longxi Gao for assistance with complementary experiments. This work was supported by the National Natural Science Foundation of China (No.62425203, No.62032003) and partly by the Supercomputing Platform of Beijing University of Posts and Telecommunications. Fanxu Meng and Muhan Zhang are supported by the National Key R&D Program of China (2022ZD0160300). Mengwei Xu was also supported by the National Natural Science Foundation of China (No.62522202) and the Beijing Natural Science Foundation (No.L253005). Mengwei Xu is the corresponding author.

References

[1] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[2] Google, "Gemini nano with the google ai edge sdk," 2025. [Online]. Available: https://developer.android.com/ai/gemini-nano

[3] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, and Y. Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 543–557.

[4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[5] Meta, "The llama 4 herd: The beginning of a new era of natively multimodal ai innovation," 2025. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[6] A. Yang, B. Yang, B. Hui *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[7] A. Yang, B. Yang, B. Zhang *et al.*, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[8] M. Xu, D. Cai, Y. Wu, X. Li, and S. Wang, "Fwdllm: Efficient federated finetuning of large language models with perturbed inferences," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 579–596.

[9] Y. Xia, J. Kim, Y. Chen, H. Ye, S. Kundu, C. Hao, and N. Talati, "Understanding the performance and estimating the cost of llm fine-tuning," *arXiv preprint arXiv:2408.04693*, 2024.

[10] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 09, 2020, pp. 13 693–13 696.

[11] A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni, "What you can cram into a single vector: Probing sentence embeddings for linguistic properties," *arXiv preprint arXiv:1805.01070*, 2018.

[12] A. Conmy, A. Mavor-Parker, A. Lynch, S. Heimersheim, and A. Garriga-Alonso, "Towards automated circuit discovery for mechanistic interpretability," *Advances in Neural Information Processing Systems*, vol. 36, pp. 16 318–16 352, 2023.

[13] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 359–17 372, 2022.

[14] G. Dar, M. Geva, A. Gupta, and J. Berant, "Analyzing transformers in embedding space," *arXiv preprint arXiv:2209.02535*, 2022.

[15] D. Rai, Y. Zhou, S. Feng, A. Saparov, and Z. Yao, "A practical review of mechanistic interpretability for transformer-based language models," *arXiv preprint arXiv:2407.02646*, 2024.

[16] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*. PMLR, 2019, pp. 2790–2799.

[17] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," *arXiv preprint arXiv:2104.08691*, 2021.

[18] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. A. Raffel, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[19] E. B. Zaken, S. Ravfogel, and Y. Goldberg, "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," *arXiv preprint arXiv:2106.10199*, 2021.

[20] Y.-L. Sung, V. Nair, and C. A. Raffel, "Training neural networks with fixed sparse masks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 24 193–24 205, 2021.

[21] S. An, Y. Li, Z. Lin, Q. Liu, B. Chen, Q. Fu, W. Chen, N. Zheng, and J.-G. Lou, "Input-tuning: Adapting unfamiliar inputs to frozen pretrained models," *arXiv preprint arXiv:2203.03131*, 2022.

[22] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, and T. Zhao, "Adaptive budget allocation for parameter-efficient fine-tuning," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=lq62uWRJjiY

[23] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, and T. Zhao, "Losparse: Structured compression of large language models based on low-rank and sparse approximation," in *International Conference on Machine Learning*. PMLR, 2023, pp. 20 336–20 350.

[24] S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen, "Dora: Weight-decomposed low-rank adaptation," *arXiv preprint arXiv:2402.09353*, 2024.

[25] F. Meng, Z. Wang, and M. Zhang, "Pissa: Principal singular values and singular vectors adaptation of large language models," *arXiv preprint arXiv:2404.02948*, 2024.

[26] S. Hayou, N. Ghosh, and B. Yu, "Lora+: Efficient low rank adaptation of large models," *arXiv preprint arXiv:2402.12354*, 2024.

[27] Y. Su, X. Wang, Y. Qin, C.-M. Chan, Y. Lin, H. Wang, K. Wen, Z. Liu, P. Li, J. Li *et al.*, "On transferability of prompt tuning for natural language processing," *arXiv preprint arXiv:2111.06719*, 2021.

[28] B. Lester, J. Yurtsever, S. Shakeri, and N. Constant, "Reducing retraining by recycling parameter-efficient prompts," *arXiv preprint arXiv:2208.05577*, 2022.

[29] Z. Wu, Y. Wu, and L. Mou, "Zero-shot continuous prompt transfer: Generalizing task semantics across language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=26XphugOcS

[30] R. Wang, S. Ghosh, D. Cox, D. Antognini, A. Oliva, R. Feris, and L. Karlinsky, "Trans-lora: towards data-free transferable parameter efficient finetuning," *Advances in Neural Information Processing Systems*, vol. 37, pp. 61 217–61 237, 2024.

[31] N. Gu, P. Fu, X. Liu, K. Ma, Z. Lin, and W. Wang, "Adapt once, thrive with updates: Transferable parameter-efficient fine-tuning on evolving base models," *arXiv preprint arXiv:2506.06844*, 2025.

[32] J. Echterhoff, F. Faghri, R. Vemulapalli, T.-Y. Hu, C.-L. Li, O. Tuzel, and H. Pouransari, "Muscle: A model update strategy for compatible llm evolution," *arXiv preprint arXiv:2407.09435*, 2024.

[33] Y. Qin, C. Qian, X. Han, Y. Lin, H. Wang, R. Xie, Z. Liu, M. Sun, and J. Zhou, "Recyclable tuning for continual pre-training," *arXiv preprint arXiv:2305.08702*, 2023.

[34] R. Shahroz, P. Li, S. Yun, Z. Wang, S. Nirjon, C.-W. Wong, and T. Chen, "PortLLM: Personalizing evolving large language models with training-free and portable model patches," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=gyHoR6uFhU

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[36] T. Nguyen, M. Raghu, and S. Kornblith, "Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=KJNcAkY8tY4

[37] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *International conference on machine learning*. PMLR, 2019, pp. 3519–3529.

[38] L. Song, A. Smola, A. Gretton, J. Bedo, and K. Borgwardt, "Feature selection via dependence maximization," *The Journal of Machine Learning Research*, vol. 13, pp. 1393–1434, 2012.

[39] M. Yamada, Y. Umezu, K. Fukumizu, and I. Takeuchi, "Post selection inference with kernels," in *International conference on artificial intelligence and statistics*. PMLR, 2018, pp. 152–160.

[40] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[42] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[43] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[44] X. Li, S. Wang, and Y. Cai, "Tutorial: Complexity analysis of singular value decomposition and its variants," *arXiv preprint arXiv:1906.12085*, 2019.

[45] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," *arXiv preprint arXiv:1905.10044*, 2019.

[46] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[47] M. Sap, H. Rashkin, D. Chen, R. LeBras, and Y. Choi, "Socialiqa: Commonsense reasoning about social interactions," *arXiv preprint arXiv:1904.09728*, 2019.

[48] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[49] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[50] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[51] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," *arXiv preprint arXiv:1809.02789*, 2018.

[52] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," *arXiv preprint arXiv:1705.04146*, 2017.

[53] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[54] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, "Mawps: A math word problem repository," in *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, 2016, pp. 1152–1157.

[55] A. Patel, S. Bhattamishra, and N. Goyal, "Are nlp models really able to solve simple math word problems?" *arXiv preprint arXiv:2103.07191*, 2021.

[56] Z. Hu, Y. Lan, L. Wang, W. Xu, E.-P. Lim, R. K.-W. Lee, L. Bing, and S. Poria, "Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models," *arXiv preprint arXiv:2304.01933*, 2023.

[57] P. Wang, Z. Li, N. Zhang, Z. Xu, Y. Yao, Y. Jiang, P. Xie, F. Huang, and H. Chen, "Wise: Rethinking the knowledge memory for lifelong model editing of large language models," *Advances in Neural Information Processing Systems*, vol. 37, pp. 53 764–53 797, 2024.

[58] Y. Yao, N. Zhang, Z. Xi, M. Wang, Z. Xu, S. Deng, and H. Chen, "Knowledge circuits in pretrained transformers," *Advances in Neural Information Processing Systems*, vol. 37, pp. 118 571–118 602, 2024.

[59] Q. Sun, M. Pickett, A. K. Nain, and L. Jones, "Transformer layers as painters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 24, 2025, pp. 25 219–25 227.

[60] A. Morcos, M. Raghu, and S. Bengio, "Insights on representational similarity in neural networks with canonical correlation," *Advances in neural information processing systems*, vol. 31, 2018.

[61] L. Yu, B. Yu, H. Yu, F. Huang, and Y. Li, "Language models are super mario: Absorbing abilities from homologous models as a free lunch," in *Forty-first International Conference on Machine Learning*, 2024.

[62] P. Yadav, D. Tam, L. Choshen, C. Raffel, and M. Bansal, "Resolving interference when merging models," *arXiv preprint arXiv:2306.01708*, vol. 1, 2023.

Algorithm 3 Hungarian Algorithm

Require: Cost matrix C of size $m \times n$
Ensure: Optimal assignment permutation
Phase 1: Matrix Preparation
Pad matrix with zeros to make it square ($n \leftarrow \max(m, n)$)
$C \leftarrow$ resulting $n \times n$ matrix
# Phase 2: Row/Column Reduction
for each row i **do**
  $r_{\min} \leftarrow \min(C[i, :])$
  $C[i, :] \leftarrow C[i, :] - r_{\min}$
end for
for each column j **do**
  $c_{\min} \leftarrow \min(C[:, j])$
  $C[:, j] \leftarrow C[:, j] - c_{\min}$
end for
# Phase 3: Initial Matching
Find maximum matching M using zeros
if matching M is perfect **then**
  return M
end if
# Phase 4: Iterative Adjustment
repeat
  Cover all zeros with minimum lines
  $k \leftarrow$ number of covering lines
  if $k = n$ **then**
    return current matching
  end if
  $\delta \leftarrow \min\{C[i, j] \mid \text{uncovered elements}\}$
  for all uncovered elements $C[i, j]$ **do**
    $C[i, j] \leftarrow C[i, j] - \delta$
  end for
  for all doubly covered elements $C[i, j]$ **do**
    $C[i, j] \leftarrow C[i, j] + \delta$
  end for
  Update matching M
until perfect matching found

A Appendix

A.1 Details of Hungarian Algorithm

The Hungarian algorithm, also known as the Kuhn-Munkres algorithm, is a combinatorial optimization technique for solving the assignment problem. It guarantees to find the optimal one-to-one assignment that minimizes the total cost in a bipartite graph. As shown in the above algorithm, the procedure mainly consists of four phases: (1) **Matrix Preparation**: Convert the cost matrix to a square matrix by zero-padding if necessary. (2) **Row/Column Reduction**: Subtract the minimum value of each row from its elements, then repeat for columns, creating at least one zero per row and column. (3) **Initial Matching**: Identify a maximum matching using zero-cost entries. If a perfect match is found, the algorithm terminates. (4) **Iterative Adjustment**: If unmatched, iteratively (4-a) cover zeros with minimal lines, (4-b) compute the smallest uncovered element δ, (4-c) adjust the matrix by subtracting δ from uncovered elements and adding δ to doubly-covered elements, and (4-d) update the matching. This loop continues until the number of covering lines equals the matrix dimension, guaranteeing an optimal assignment. The algorithm operates in $O(n^3)$ time [43], where n is the number of nodes in the smaller partition.

A.2 Hyperparameters

Tables 8 and 9 detail the hyperparameters of LoRA and LoRASuite for commonsense and math tasks, respectively.

Table 8: Hyperparameter configurations of LoRA for models on the commonsense reasoning tasks.

Hyperparameters	LoRA	LoRASuite
Rank	32	
a	32	
Dropout	0	
Optimizer	AdamW	
LR	3e-4	1e-3
LR Scheduler	Linear	
Batch Size	16	
Warmup Ratio	0.1	0
Epochs	3	
Target Module	Q,K,V,O	

Table 9: Hyperparameter configurations of LoRA for models on the mathematical tasks.

Hyperparameters	LoRA	LoRASuite
Rank	32	
a	32	
Dropout	0	
Optimizer	AdamW	
LR	3e-4	1e-3
LR Scheduler	Linear	
Batch Size	16	
Warmup Steps	100	0
Epochs	3	
Target Module	Q,K,V,O	

A.3 Different Target Modules

Figures 5 and 6 present the performance of LoRASuite using q_proj, k_proj, v_proj, o_proj, up_proj, and down_proj as target modules. As shown, the performance improvement on the math dataset becomes more pronounced as the number of target modules increases. A likely explanation is that the commonsense corpus significantly overlaps with the LLM pre-training dataset; consequently, LoRASuite's additional small-scale fine-tuning is more impactful for mathematical tasks.

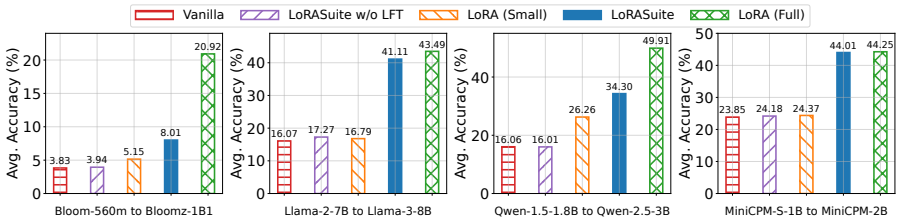

Figure 5: Performance comparison on math tasks for different LLMs with up_proj and down_proj as target modules.

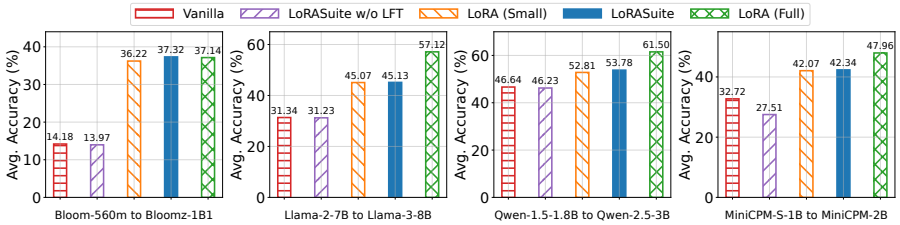

Figure 6: Performance comparison on commonsense tasks for different LLMs with up_proj and down_proj as target modules.

A.4 Heatmap of Layer Similarity

Figures 7 to 10 illustrate the CKA layer similarities of various LLM backbones before and after upgrading. The similarity patterns vary significantly across models, with MiniCPM and Pythia showing the highest variance. Notably, all models exhibit a clear block structure, similar to the three distinct representation spaces—"beginning," "middle," and "end" described in [59].

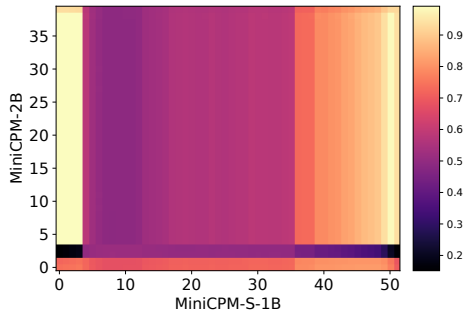

Figure 7: Heatmap of CKA layer similarity between MiniCPM-S-1B and MiniCPM-2B.

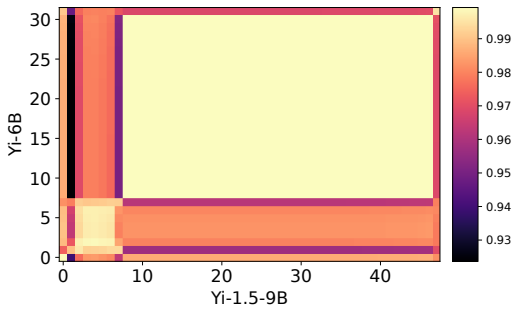

Figure 8: Heatmap of CKA layer similarity between Yi-6B and Yi-1.5-9B.

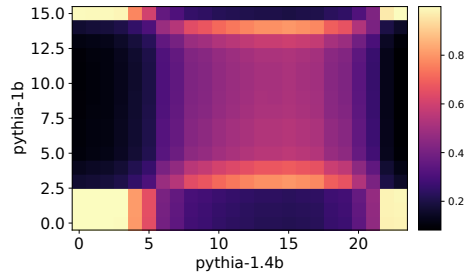

Figure 9: Heatmap of CKA layer similarity between pythia-1b and pythia-1.4b.

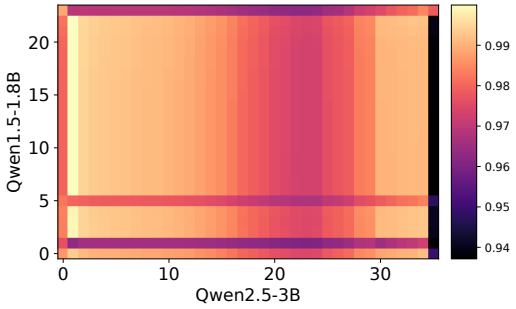

Figure 10: Heatmap of CKA layer similarity between Qwen1.5-1.8B and Qwen2.5-3B.

A.5 Detailed Results for Different LLM Backbones

Tables 10–19 provide the raw data underlying the aggregated results in Figures 2 and 3.

Table 10: Performance on math tasks when LLM upgrades from Yi-6B to Yi-1.5-9B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
Yi-6B	-	48.86	63	49.02	17.21	13.78	39.08	37.9	38.41
	LoRA (10k)	68.61	94	72.05	28.51	22.44	61.76	44.9	56.04
Yi-1.5-9B	-	78.48	66.67	82.87	47.99	41.34	78.57	65.1	65.86
	LoRASuite w/o LFT	90.89	76.5	88.98	52.01	38.58	86.55	78	73.07
	LoRA (100)	82.78	70.5	86.02	54.66	40.16	81.93	68.9	69.28
	LoRASuite w LFT (100)	86.58	91.33	89.76	66.26	30.31	84.45	75.1	74.83
	LoRA (10k)	86.84	95.33	91.14	68.54	32.28	85.29	74.8	76.32

Table 11: Performance on commonsense tasks when LLM upgrades from Yi-6B to Yi-1.5-9B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
Yi-6B	-	64.25	36.51	14.94	17.63	4.66	16.13	16.5	17.6	23.53
	LoRA (10k)	69.33	37.92	64.43	52.64	70.72	41.55	51.01	57.40	55.63
Yi-1.5-9B	-	61.13	76.39	63.2	66.13	17.36	58.11	66.54	45.4	56.78
	LoRASuite w/o LFT	56.94	73.78	77.84	78.41	61.09	77.56	89.48	79.20	74.29
	LoRA (100)	66.61	75.90	65.05	60.13	64.56	73.98	80.05	59	68.16
	LoRASuite w LFT (100)	69.30	72.09	72.82	48.98	73.09	73.55	86.78	69	70.70
	LoRA (10k)	71.71	81.77	78.20	90.69	82.48	84.04	92.34	83.8	83.13

Table 12: Performance on math tasks when LLM upgrades from Pythia-1B to Pythia-1.4B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
Pythia-1B	-	1.01	2.50	0.98	0.83	21.26	0.84	1.1	4.07
	LoRA (10k)	12.66	19.50	12.40	1.97	14.57	7.56	8.30	10.99
Pythia-1.4B	-	1.52	2.67	0.39	1.14	20.47	0	0.9	3.87
	LoRASuite w/o LFT	0.76	2.33	1.18	0.53	21.65	0	0.8	3.89
	LoRA (1k)	2.03	4.83	3.15	1.52	12.6	0.84	2.1	3.87
	LoRASuite w LFT (1k)	10.63	15	9.65	1.82	15.75	7.56	6.2	9.52
	LoRA (10k)	27.85	43.67	25.39	3.56	20.87	21.43	14.4	22.45

Table 13: Performance on commonsense tasks when LLM upgrades from Pythia-1B to Pythia-1.4B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
Pythia-1B	-	54.19	15.56	3.53	8.22	26.05	4.61	4.08	7	15.41
	LoRA (10k)	61.65	33.08	34.03	25.06	49.49	23.29	24.62	28.4	34.95
Pythia-1.4B	-	53.82	11.32	14.48	16.22	22.65	7.51	9.93	10.2	18.27
	LoRASuite w/o LFT	51.16	11.48	13.82	14.91	14.29	8.79	10.1	9.4	16.74
	LoRA (100)	57.16	45.76	29.48	22.76	48.07	21.50	22.22	26	34.12
	LoRASuite w LFT (100)	56.06	48.64	32.70	25.05	50.43	22.70	24.49	23.40	35.43
	LoRA (10k)	48.38	45.1	32.45	24.99	51.54	23.12	25.21	29.4	35.02

Table 14: Performance on math tasks when LLM upgrades from Bloom-560m to Bloomz-1B1. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
Bloom-560m	-	0.76	2.17	0.59	0.76	5.91	0.42	0.80	1.63
	LoRA (10k)	8.35	8.17	7.68	2.27	20.47	5.04	2.10	7.73
Bloomz-1B1	-	1.77	4.67	0.79	2.65	11.81	2.1	3	3.83
	LoRASuite w/o LFT	1.77	4.67	0.79	2.5	12.2	2.1	2.9	3.85
	LoRA (1k)	2.03	2	2.95	1.97	11.81	4.2	1.7	3.81
	LoRASuite w LFT (1k)	6.84	4	4.72	1.67	17.72	5.88	3	6.26
	LoRA (10k)	17.22	16.67	19.09	3.11	19.29	15.13	11.1	14.52

Table 15: Performance on commonsense tasks when LLM upgrades from Bloom-560m to Bloomz-1B1. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
Bloom-560m	-	6.88	8.54	7.32	14.43	2.45	5.46	3.91	4.4	6.67
	LoRA (10k)	54.86	50.76	33.67	25.23	51.38	24.06	25.38	26.8	36.52
Bloomz-1B1	-	38.65	35.36	0.15	23.96	13.89	0.51	0.34	0.60	14.18
	LoRASuite w/o LFT	38.65	35.64	0.15	23.90	13.97	0.60	0.34	0.80	14.26
	LoRA (100)	56.06	0.71	31.12	22.21	38.75	19.11	21.76	26.4	27.02
	LoRASuite w LFT (100)	62.17	49.67	32.91	25.09	50.83	22.70	25.08	27.60	37.01
	LoRA (10k)	60.92	51.51	35.21	24.33	51.07	25.43	24.12	22.4	36.87

Table 16: Performance on math tasks when LLM upgrades from Llama-2-7B to Llama-3-8B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
Llama-2-7B	-	1.77	1.50	1.77	0.99	21.65	0.84	1.50	4.29
	LoRA (10k)	29.11	42.50	28.15	8.11	11.42	28.57	18.60	23.78
Llama-3-8B	-	18.73	16.67	19.09	4.09	21.26	19.33	13.3	16.07
	LoRASuite w/o LFT	18.99	16.33	18.90	3.18	22.05	16.39	12.70	15.51
	LoRA (100)	20.25	20.50	21.46	7.05	21.65	15.97	14.90	17.40
	LoRASuite w LFT (100)	45.82	70	46.85	17.74	20.47	44.96	35.4	40.18
	LoRA (10k)	55.70	85	56.69	23.35	23.62	53.78	48.50	49.52

Table 17: Performance on commonsense tasks when LLM upgrades from Llama-2-7B to Llama-3-8B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
Llama-2-7B	-	56.21	15.34	17.30	14.68	15.79	5.55	7.11	10.60	17.82
	LoRA (10k)	63.18	25.46	42.63	31.60	58.72	20.82	27.95	32.40	37.85
Llama-3-8B	-	51.44	45.81	34.34	23.09	12.47	27.22	29.71	26.60	31.34
	LoRASuite w/o LFT	52.26	45.97	33.67	23.07	12.94	27.05	30.13	26	31.39
	LoRA (100)	62.29	48.59	42.78	29.25	51.93	35.67	43.86	43.4	44.72
	LoRASuite w LFT (100)	62.39	57.67	45.80	26.76	54.54	35.32	45.12	42.40	46.25
	LoRA (10k)	72.97	50.22	51.54	37.3	72.14	31.48	34.60	41	48.91

Table 18: Performance on math tasks when LLM upgrades from Qwen-1.5-1.8B to Qwen-2.5-3B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	AddSub	MultiArith	SingleEq	GSM8K	AQuA	MAWPS	SVAMP	Avg.
Qwen-1.5-1.8B	-	32.91	51.33	52.56	9.70	11.02	47.06	32.70	33.90
	LoRA (10k)	52.91	79.67	59.25	12.74	14.17	55.04	33.50	43.90
Qwen-2.5-3B	-	22.53	18.33	13.39	5.91	28.35	7.98	15.90	16.06
	LoRASuite w/o LFT	23.29	21.50	13.98	5.84	25.59	6.30	15.10	15.94
	LoRA (100)	30.13	25.83	15.55	8.19	26.77	13.45	15.40	19.33
	LoRASuite w LFT (100)	53.16	82	66.73	44.50	37.40	57.98	56.40	56.88
	LoRA (10k)	42.28	88.67	62.40	32.90	24.80	44.96	56.20	50.32

Table 19: Performance on commonsense tasks when LLM upgrades from Qwen-1.5-1.8B to Qwen-2.5-3B. The number in parentheses represents the scale of fine-tuning datasets.

Base Model	PEFT	BoolQ	PIQA	SIQA	HellaSwag	WinoG	ARC-c	ARC-e	OBQA	Avg.
Qwen-1.5-1.8B	-	51.16	43.42	18.17	24.13	18.55	20.14	22.01	23.40	27.62
	LoRA (10k)	52.75	54.52	58.55	32.48	57.14	37.46	49.66	51.80	48.30
Qwen-2.5-3B	-	65.17	57.89	43.81	36.69	52.72	36.18	40.07	40.60	46.64
	LoRASuite w/o LFT	65.72	57.40	43.40	36.19	51.30	36.35	40.36	40.60	46.42
	LoRA (100)	28.10	56.91	47.29	38.87	50.04	37.45	43.81	39.20	42.71
	LoRASuite w LFT (100)	63.09	58.27	60.70	33.73	54.46	49.66	61.36	54	54.41
	LoRA (10k)	60.31	59.96	59.21	46.93	63.93	49.66	56.36	54	56.30

A.6 Ablation Studies

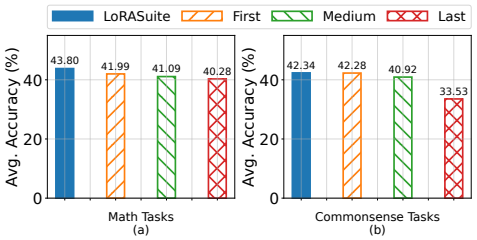

Figure 11: Performance comparison with different layer mapping algorithms when upgrading the LLM from MiniCPM-S-1B to MiniCPM-2B.

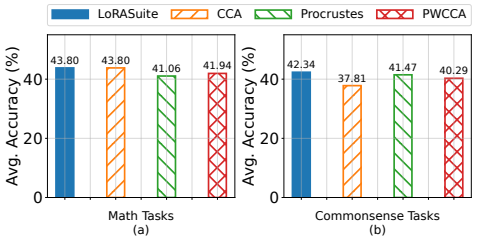

Figure 12: Performance comparison with different representational similarity methods when upgrading the LLM from MiniCPM-S-1B to MiniCPM-2B.

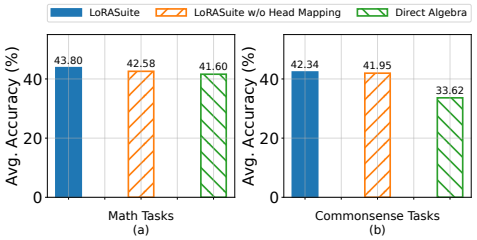

Figure 13: Performance comparison with different attention head mapping algorithms when upgrading the LLM from MiniCPM-S-1B to MiniCPM-2B.

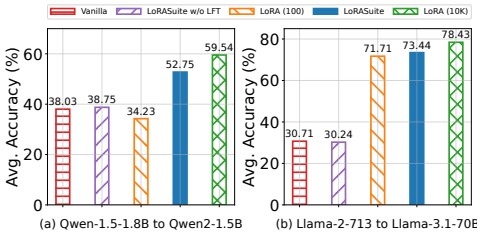

Figure 14: Performance on math tasks when LLM upgrades from Qwen-1.5-1.8B to Qwen2-1.5B and from Llama-2-13b-hf to Llama-3.1-70B.

CKA-based Layer Mapping. Figure 11 compares the performance of different layer mapping algorithms: "LoRASuite" refers to our CKA-based method described in Section 3.1; "First" maps layers sequentially from beginning to end; "Medium" spreads mappings outward from the middle; and "Last" maps from end to beginning. Our CKA-based approach outperforms the others on both math and commonsense tasks, demonstrating the effectiveness of using representational similarity to guide layer mapping.

Figure 12 further evaluates different representational similarity metrics. Canonical Correlation Analysis (CCA) [37] identifies linear projections that maximize correlation between two sets of activations. Projection-Weighted CCA (PWCCA) [60] extends CCA by weighting individual canonical correlations based on their importance. Procrustes analysis seeks the optimal orthogonal transformation that minimizes the Frobenius norm between two matrices. Our minibatch CKA-based method achieves superior performance across both task types, demonstrating the advantage of using minibatch CKA for guiding layer mapping.

Attention-head Mapping. Figure 13 compares the performance of different attention head mapping algorithms. "LoRASuite" refers to our Hungarian-based method described in Section 3.1, while "LoRASuite w/o Head Mapping" uses the default ordering without enforcing one-to-one correspondence based on cosine similarity, and the "Direct Algebra" method calculates the new LoRA weights by $\Delta W_n = W_o - W_n + \Delta W_o + c$ where c is just a learnable set of weights need to fine-tune. Our approach consistently outperforms the baseline on both math and commonsense tasks, highlighting the effectiveness of using input-independent interaction matrices and similarity-based head mapping. For the direct algebra method, the performance gap, especially on commonsense tasks, suggests that directly aligning to the trained $W_o + \Delta W_o$ may discard valuable information encoded in the new weight W_n through its pretraining and post-training. As a result, even with further lightweight fine-tuning, this method struggles to fully leverage the potential of the stronger target model W_n.

A.7 Loss Convergence

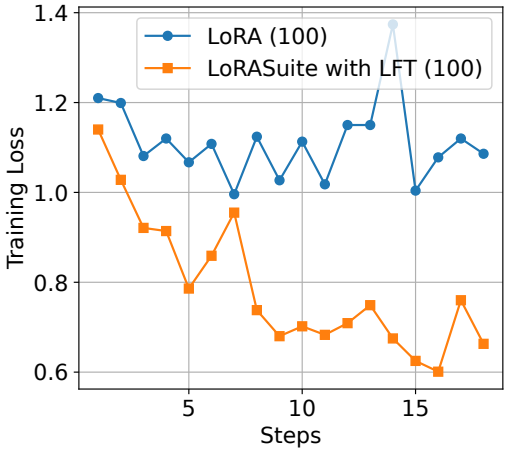

Figure 15: Step-wise training loss comparison between "LoRA (100)" and "LoRASuite with LFT (100)".

In Figure 15, we visualize the step-wise training loss for "LoRA (100)" and "LoRASuite with LFT (100)" under the settings of batch size = 16 and micro batch size = 4 for MiniCPM-2B. As mentioned in the Section 3.2, the default linear learning-rate scheduler of vanilla LoRA (100) with a warm-up phase to stabilize training from random initialization. However, with limited data, a considerable steps is spent in the warm-up stage, which slows convergence and limits performance. In contrast, LoRASuite starts with transformed parameters already close to a good solution, allowing us to omit warm-up and use a higher learning rate to accelerate convergence. This leads to better performance, especially in low-data regimes.

Table 20: Performance comparison for merged LoRA weights.

Base Model	PEFT Module	Avg. Math Accuracy	Avg. Common Accuracy
MiniCPM-S-1B	TIES-merged LoRA	17.45	11.53
MiniCPM-2B	TIES-merged LoRASuite-transformed LoRA	23.90 (1.37×)	22.57 (1.96×)

A.8 Merged LoRA Adaptation

Prior studies, such as DARE [61] and TIES [62], have demonstrated the feasibility and effectiveness of merging task-specific LoRA modules. In this section, we evaluate whether LoRASuite maintains its effectiveness when LoRA weights are merged across different tasks.

As shown in Table 20, we merged MiniCPM-S-1B's math and commonsense LoRA adapters using TIES with equal weights (0.5 each) and applied the same merging procedure to the LoRASuite-transformed adapters of MiniCPM-2B. Compared with the simple average of individual task performances, the merged LoRASuite-transformed adapters improved math and commonsense scores by 1.37× and 1.96×, respectively, demonstrating that our method remains effective even when merging weights from distinct tasks.

A.9 More LLM Backbones

Figure 14 (a) presents the performance on math tasks when upgrading LLM backbones from Qwen-1.5-1.8B to Qwen2-1.5B, covering five major architectural changes: hidden size, intermediate size, layer depth, head number, and attention type. Similarly, Figure 14 (b) reports results for the upgrade from Llama-2-13B to Llama-3.1-70B, encompassing all six key architectural changes. In both cases, LoRASuite remains consistently effective.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our contribution is outlined as a separate paragraph in § 1.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations of our work are outlined as a separate paragraph in § 5.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We have provided the time complexity analysis for the algorithms involved in § 3.3 and § A.1, respectively.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Consistent with previous studies, our evaluation employs the widely recognized open-source benchmark [56].

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use the same open-source dataset as [56] and our code is available in `https://github.com/YananLi18/LoRASuite`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We provide detailed instructions on how to reproduce the main experimental in § 4

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We primarily report average performance metrics rather than error bars in our experiments. This choice aligns with prior studies that also emphasize mean performance. Additionally, due to significant variability across different datasets, reporting deviations comparable in magnitude to the averages would not yield meaningful insights. Nevertheless, we provide comprehensive data, enabling interested readers to compute error bars independently.

Guidelines:

- The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide detailed hardware information in § 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and believe that our research conforms to it.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: In § 1, we discussed real-world examples illustrating how our approach can promisingly mitigate the negative impacts associated with the current status quo, including excessive time consumption, high costs, and environmental harm.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper is intended for LoRa adaptation and does not involve high-risk data or models.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We have properly cited the original code, data, and models in § 4.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: We will provide detailed documentation for the new assets upon acceptance.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing or research with human subjects; therefore, Institutional Review Board (IRB) approval is not applicable.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.