Llama SLayer 8B: Shallow Layers Hold the Key to Knowledge Injection

Tianxiang Chen[1,2,3,4]*, **Zhentao Tan**[1,2,3,4], **Tao Gong**[1,2,3]†, **Yue Wu**[4], **Qi Chu**[1,2,3], **Bin Liu**[1,2,3]
Jieping Ye[4], **Nenghai Yu** [1,2,3]
[1]School of Cyber Science and Technology, University of Science and Technology of China
[2]Anhui Province Key Laboratory of Digital Security
[3]CAS Key Laboratory of Electromagnetic Space Information
[4]Alibaba Cloud
Correspondence: tgong@ustc.edu.cn

Abstract

As a manner to augment pre-trained large language models (LLM), knowledge injection is critical to develop vertical domain large models and has been widely studied. Although most current approaches, including parameter-efficient fine-tuning (PEFT) and block expansion methods, uniformly apply knowledge across all LLM layers, it raises the question: are all layers equally crucial for knowledge injection? We begin by evaluating the importance of each layer in finding the optimal layer range for knowledge injection. Intuitively, the more important layers should play a more critical role in knowledge injection and deserve a denser injection. We observe performance dips in question-answering benchmarks after the removal or expansion of the shallow layers, and the degradation shrinks as the layer gets deeper, indicating that the shallow layers hold the key to knowledge injection. This insight leads us to propose the S strategy, a post-pretraining strategy of selectively enhancing shallow layers while pruning the less effective deep ones. Based on this strategy, we introduce Llama Slayer-8B and Llama Slayer-8B-Instruct. We experimented on the corpus of code & math and demonstrated the effectiveness of our strategy. Further experiments across different LLM, Mistral-7B, and a legal corpus confirmed the general applicability of the approach, underscoring its wide-ranging efficacy. Our code is available at: https://github.com/txchen-USTC/Llama-Slayer.

1 Introduction

Large Language Models (LLMs) have significantly revolutionized the natural language processing area, showcasing unparalleled abilities across various tasks (Achiam et al., 2023). Despite their versatility, LLMs exhibit limitations in specialized areas such as mathematics, programming, etc., which hinder the potential of wide-ranging applications. To address these gaps, existing work (Liu et al., 2023; Wang et al., 2023) has sought to enhance the diverse skills of pre-trained LLMs through customized data strategies. However, they require extensive computational efforts and massive data volumes, challenging the widespread accessibility of LLM research. Furthermore, while Parameter-Efficient Fine-Tuning (PEFT) techniques offer a reduction in training requirements, their effectiveness tends to diminish (Biderman et al., 2024; Wu et al., 2024) compared to traditional fine-tuning methods, especially as the size of the model and the dataset grows.

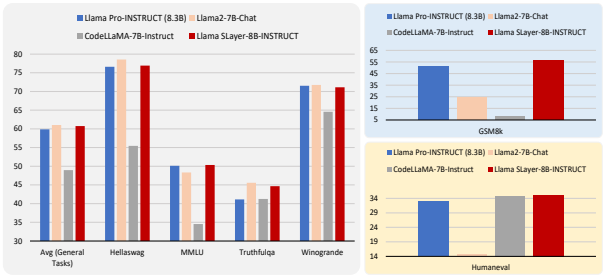

Figure 1: Llama SLayer-8B-INSTRUCT achieves state-of-the-art performance across several tasks, spanning from general language tasks to specific domain tasks (programming and mathematics), outperforming its LLaMA series predecessors.

Subsequently, another line of research emerged, focusing on methods such as model merging (Akiba et al., 2024) and model expansion (Wu et al., 2024; Choi and Gazeley, 2024; Kim et al., 2023). Model merging methods strive to synthesize a multifaceted model that amalgamates insights from various pre-trained domain-specific LLMs, potentially crafting a model adept at addressing a multitude of tasks concurrently. However, the process of training multiple domain-specific LLMs is resource-intensive. On the other hand, model expansion methods, exemplified by Llama Pro, seek

*Work done during an internship at Alibaba Cloud.
†Tao Gong is the corresponding author.

Findings of the Association for Computational Linguistics: EACL 2024, pages 5991–6002

to refine pre-trained models for domain-specific applications in the post-pretraining phase by only fine-tuning the expanded layers. Therefore, it can employ significantly fewer trainable parameters than full model fine-tuning.

However, present model expansion methods generally treat each part of LLMs equally, although different layers may exhibit varying sensitivity to incorporated knowledge. This lack of differentiation can result in less-than-ideal knowledge injection results. An intuitive idea is to inject knowledge into the most important layers so that the LLM can more sufficiently leverage the new knowledge without the overhead of redundant adjustments across all layers. To this end, we select three different metrics to evaluate the importance of each layer to find which part of the LLM is more important to knowledge injection. Our findings suggest that the shallow layers are more important compared to the last few layers, as the drop in precision - whether through the removal or addition of the last few layers - is markedly less significant than the drops induced by manipulating the shallow layers. Drawing on this insight, we propose S strategy, a novel strategy to knowledge injection that concentrates on enriching the shallow layers while deleting the least important deepest layers. Based on the proposed strategy, we further introduce Llama SLayer-8B and Llama SLayer-8B-INSTRUCT, versatile LLMs that excelling in programming, and mathematics and general language tasks. Figure 2 displays the superiority of Llama SLayer-8B-INSTRUCT.

The main contributions of this paper can be summarized to three aspects:

- We propose a novel post-pretraining strategy for LLMs, namely S strategy, that focuses knowledge injection to the important layers while pruning the ineffective layers.
- Based on our S strategy, we introduce Llama SLayer-8B and Llama SLayer-8B-INSTRUCT, versatile LLMs that excelling in programming, and mathematics and general language tasks.
- We benchmark the family of Llama SLayer on extensive datasets, demonstrating its exceptional performance and significant promise for diverse and complex applications.

2 Related Works

Here we introduce four prevalent types of methods for injecting domain-specific knowledge, including full fine-tuning, parameter-efficient fine-tuning, model merging, and model expansion.

2.1 Full Fine-tuning Methods

Full fine-tuning of Pretrained Language Models (PLMs) involves retraining all parameters for a particular task with domain-specific knowledge (Touvron et al., 2023a; Liu et al., 2019). Initially trained on vast unsupervised datasets to learn broad language representations, these PLMs may underperform on specialized tasks due to lack of domain-specific expertise (Xu and Wang, 2023; Dabre et al., 2019). Full fine-tuning addresses this by adapting models such as HuaTuo (Wang et al., 2023), a Chinese biomedical LLM based on LLaMA-7B, and programming-focused LLMs such as CodeLLama (Roziere et al., 2023) and Code-Qwen (Li et al., 2023), for targeted applications. Despite its effectiveness, this approach requires extensive computational resources and substantial labeled data, which pose challenges like overfitting on small task-specific datasets, particularly as PLMs increase in size and complexity (Pfeiffer et al., 2020).

2.2 Parameter-Efficient Fine-Tuning Methods

To reduce computational demands, Parameter-Efficient Fine-Tuning (PEFT) techniques modify only trivial parts of PLMs, maintaining performance comparable to full fine-tuning. LoRA (Hu et al., 2021) uses trainable low-rank matrices to update weights with the original weights of the PLM remaining unchanged, while AdaLoRA (Zhang et al., 2023) adjusts the rank of these matrices for optimized performance. Adapter-based Fine-tuning (Houlsby et al., 2019; Lei et al., 2024) introduces adapters into the transformer architecture, allowing fine-tuning with minimal alteration to pre-trained parameters. However, as the size of models and datasets increases significantly, PEFT methods tend to fall behind in performance compared to full fine-tuning approaches (Biderman et al., 2024; Wu et al., 2024).

2.3 Model Merging Method

Model merging methods aim to create a comprehensive model by integrating knowledge from several pre-trained domain-specific LLMs. Task Arithmetic (Yadav et al., 2024) construct task vectors

by differentiating pre-trained and fine-tuned model weights, allowing for model behavior adjustments through arithmetic operations. DARE (Yu et al., 2023a) further refines this by focusing on and enhancing the critical disparities between models. Evolutionary algorithms proposed by Takuya et al. (Akiba et al., 2024) streamline the merging process without necessitating fine-tuning, although this method's reliance on multiple pre-trained models and the subjective nature of the merging strategy may complicate its broad use. However, generating multiple domain-specific LLMs still requires substantial computational resources.

2.4 Model Expansion Method

To reach a better trade-off between computational resources and domain-specific performance, the model expansion method has been introduced. These techniques typically incorporate a more trainable parameters than PEFT methods, albeit significantly fewer than what is employed in full fine-tuning, and have been shown to yield impressive results. SOLAR 10.7B (Kim et al., 2023) features an innovative approach that involves merging the initial 24 layers with the final 24 layers of the same model in depth as its continuous pre-training strategy. However, its superior performance comes at the cost of training all 48 layers after expansion. Llama Pro (Wu et al., 2024) adopts a method of evenly distributing expansion blocks across all layers of the model. These expansion blocks are initialized by duplicating the weights of the preceding block and zeroing out specific weights to guarantee the same initial output as the original base model. During the subsequent phase of continual pre-training, only these expanded blocks get trained.

However, present model expansion methods lack exploration on which part of layers is more suitable for merging, since different layers may not be equally sensitive to the injected knowledge. To explore which parts of LLMs are pivotal for knowledge injection, we evaluated different LLMs based on layer importance and discovered that shallow layers wield greater importance than deep layers. Based on this, we propose a novel knowledge injection strategy, namely the S strategy, that targets the knowledge injection to the important layers via block expansion and dispenses with the ineffective last few layers. We implement this strategy and propose Llama Slayer 8B to validate its effectiveness-enabling LLMs to better specialize in specific tasks while preserving general abilities.

3 Method

3.1 Evaluation Metrics of Layer Importance

3.1.1 Angular Distance

We first try to evaluate layer importance from the feature transition aspect. To this end, we adopt the angular distance (AD) metric to evaluate the significance of each block. The angular distance $\mathcal{A}_{i,i+1}$ between the input features of block i and block $i+1$ is calculated as follows:

$$\mathcal{A}_{i,i+1} = \frac{1}{\pi} arccos(\frac{x_i^T x_{i+1}}{||x_i||||x_{i+1}||}) \quad (1)$$

where $||.||$ denotes the L_2-norm. This metric helps identify blocks where significant data processing shifts occur when exposed to new data, indicating their pivotal role in adapting to new knowledge. We calculate AD using the MMLU test benchmark to get a low-fluctuation estimate. Higher AD denotes higher difference between between inputs and outputs of each block, therefore the areas with higher angular distance are earmarked for modifications, such as layer expansion, to augment the model's adaptability and improve its performance on the specific domain dataset.

3.1.2 Performance Drop after Layer Removal

We can also locate the important layer areas by comparing the overall model performance drop on general question-answering benchmarks when removing different layers. We consider the layers that exhibit the most significant drop in performance once they are deleted as the most critical. Through experiments on Llama2 7B and Mistral 7B on two general QA benchmarks (MMLU (Hendrycks et al., 2020) and ARC (Clark et al., 2018)), we find the shallow layers as the important ones, so we locate the first half of all layers as the core area for knowledge injection. The choice of these two benchmarks stems from observing that the LLM after layer deletion or insertion has lost logic inference ability (evidenced by zero accuracy on GSM8k (Cobbe et al., 2021) and Humaneval (Chen et al., 2021)), so we select the two multi-choice QA general benchmarks to measure performance drops.

Our intuition for layer removal comes from thinking about the representations as slowly changing the function of layer index. In particular, the layer-to-layer evolution of representations for a

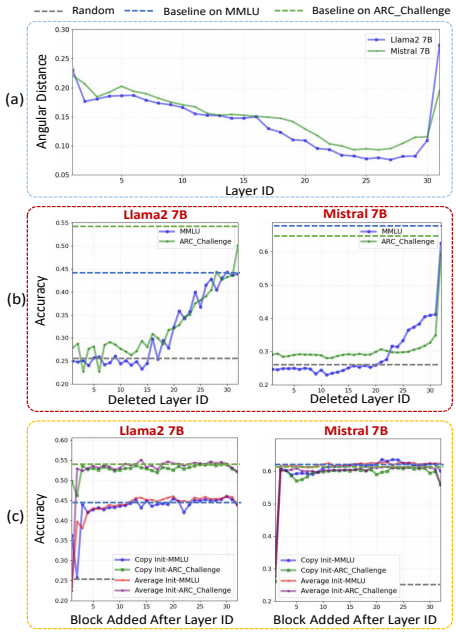

Figure 2: (a) Angular distance between inputs and outputs of each block vs. the layer number; (b) Accuracy of Llama2-7B and Mistral 7B dropping one layer on two general QA benchmarks; (c) Accuracy of Llama2-7B and Mistral 7B with one more inserted block on the same two QA benchmarks. The initialization of the expanded blocks includes identity copy and averaging the adjacent block weights.

transformer is given by a residual iteration equation

$$x_{i+1} = x_i + F(x_i, \theta_i) \tag{2}$$

where x_i and θ_i are the input and parameter vectors for layer i, respectively, and $F(x_i, \theta_i)$ describes the transformation of one multi-head self-attention and MLP layer block. If we remove layer i, then we must now connect the old input to that layer, x_{i-1}, into the block function of layer $i+1$ as

$$x_{i+1} = x_{i-1} + F(x_{i-1}, \theta_{i-1}) \tag{3}$$

Comparing Equation 2 with Equation 3 we can find a mismatch between the original input and new input, which should be very damaging for the network and cause a performance drop.

3.1.3 Performance Drop after Layer Insertion

Layer insertion is the reverse process of layer removal. In particular, the layers that suffer the most pronounced performance drop when testing on MMLU (Hendrycks et al., 2020) and ARC (Clark et al., 2018) when inserting a new block after it are deemed the most crucial.

Two different initialization methods can be employed for the insertion. The first is the identity copy of the weights from the preceding block. If we expand a block between layer i and $i+1$, the layer-to-layer evolution of representations from layer i to $i+1$, initiated by identity copy, can be depicted as

$$x_{i+1} = x_i + F(x_i, \theta_i) + F(x_i + F(x_i, \theta_i), \theta_i) \tag{4}$$

In addition, we also employs weight averaging as the second method for the inserted block by averaging the weights of the adjacent two blocks. The reason for this initialization is that we think identity copy may be not smooth enough and we hope to start the later continual pre-training from a more smooth initialization state. In this way, the representation evolution from layer i to $i+1$ is

$$x_{i+1} = x_i + F(x_i, \theta_i) + F(x_i + F(x_i, \theta_i), \frac{\theta_i + \theta_{i+1}}{2}) \tag{5}$$

3.2 Evaluation Analysis

We illustrate the layer importance evaluation results in Figure 2. Observing the angular distances across the layers (Figure 2 (a)), we note that the first half and the final layer exhibit higher angular distances. The performance impact of removing the layers (Figure 2 (b)) shows a more noticeable drop in the initial half than in the last half of the layers. Similarly, observing the performance drop after layer insertion (Figure 2 (c)), we find that layer insertion mirrors this trend of performance drop. Additionally, initializing with the weight-averaging method has demonstrated less impact on performance compared to the identity copy method.

According to the above observations, we conclude that the shallow layers play a more crucial role in knowledge injection compared to the deeper layers. This is evidenced by (1) the angular distances are generally higher in shallow layers, indicating a more significant shift in features in these areas, and (2) the elimination or inserting of shallow layers poses more severe impacts on the overall performance of LLM in two QA benchmarks, underscoring the pivotal role of shallow layers and the relative ineffectiveness of deep layers. Given the ineffectiveness of the deepest layers, we opt to prune them prior to post-pretraining to assess the impact on final performance. We also observe a more modest decline in performance when layers inserted via weight averaging initialization are compared to those initialized with an identity copy, suggesting that weight averaging introduces additional

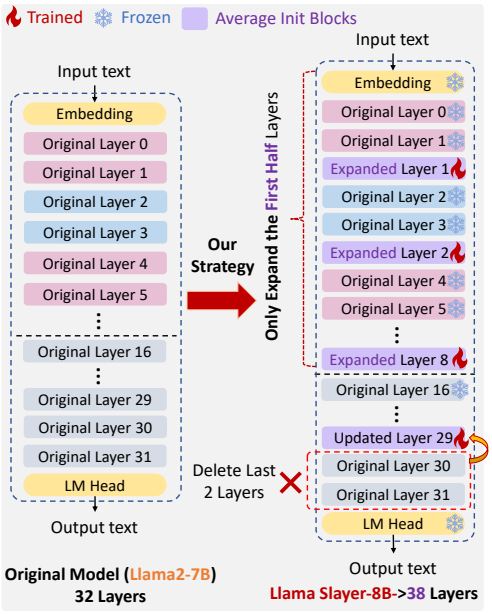

Figure 3: Our strategy focuses on infusing domain-specific knowledge to the first half of the model layers (shallow layers) and the final layer. This is achieved by augmenting the model via block expansion after the deletion of layers deemed non-essential. The expanded blocks are initialized using a linear interpolation technique to ensure a coherent knowledge structure, and only the expanded blocks are fine-tuned.

coherence to the expanded LLM. This prompts further exploration into the potential benefits of using weight averaging as an initialization strategy for post-pretraining.

3.3 S Strategy

Based on the above analyses, our S strategy involves expanding the blocks in the areas of the first half (important) of the layer and removing the last few unimportant layers before post-pre-training, as shown in Figure 3. Only the expanded blocks and final pruned layers get fine-tuned, designated in purple. The block expansion is adopted from Llama Pro (Wu et al., 2024), but the main differences of our strategy with Llama Pro (Wu et al., 2024) lie in three aspects: (1) we expand blocks at intervals within the shallow (first half) layers based on importance, rather than equally expanding across all layers; (2) We employ a weight averaging method to initialize these expanded blocks, which adds layer coherence and is better than the identity copy used by Llama Pro (Wu et al., 2024). (3) We remove the less critical final to make the LLM lighter.

Specifically, given an LLM with L blocks, the block expansion incorporates an identity block after each block in the original model, ensuring that the expanded model maintains the same output after expansion. Suppose that we have an initial model with L blocks. Since we regard the shallow layers as more important, we divide the first half $(L/2)$ of the original L blocks into N segments, with each segment consisting of L_N blocks. To expand these segments, we replicate the foremost block of each and place it atop the respective segment. The expanded N blocks are then initialized by averaging the weights of two neighboring blocks. Considering the limited impact of the deep few LLM blocks, we proceed to delete the last few D layers and the final expanded LLM has $L + N - D$ blocks. To maintain continuity, the final block is reinitialized for training using the combined average weights of the discarded D blocks.

4 Experiments

In this section, we detail our key experimental findings. Initially, we discuss our experimental settings (described in Sec. 4.1), and then provide our post-pretraining results (CPT+SFT) (described in Sec. 4.2). Finally, ablation studies of the different block expansion mode, different LLM and data corpus are presented (described in Sec. 4.3).

4.1 Experimental Settings.

4.1.1 CPT details.

We construct a dataset that concentrates on code and math. For the math component, we opt for the Proof-pile-2 dataset[1], a 55-billion-token amalgamation of scientific papers, web data containing mathematical content, and mathematical code. As for the code component, the code fragment of our dataset, we draw upon the Stack-dedup dataset[2], a vast repository of openly licensed source codes aggregated from GitHub. Among all the programming languages in Stack-dedup, we only select the 22-billion-token Python division. Notably, due to constraints on our computational resources, we only sample a smaller subset totaling 30 billion tokens from both datasets, maintaining a Math to Code ratio of 5:2, for continued pretraining. In the ablation study section, we further downsize our dataset selection to a 5-billion-token extract, equally distributed following a Math:Code ratio of 5:2, again, to improve experiment efficiency.

[1]https://huggingface.co/datasets/EleutherAI/proof-pile-2
[2]https://huggingface.co/datasets/bigcode/the-stack-dedup

Properties	CPT datasets (30B token)		SFT datasets (1.2M samples)			
	Stack-dedup-Python	Proof-pile-2	WizardLM evol instruct	SlimOrca	MetaMath	Evol-CodeAlpaca
Total # Samples	22B token	55B token	0.143M	0.518M	0.395M	0.113M
Used # Samples	8.6B token	21.4B token	0.143M	0.518M	0.395M	0.113M
Open Source	✓	✓	✓	✓	✓	✓

Table 1: Training datasets used for the continued pretraining (CPT) and supervised fine-tuning (SFT) stages, respectively. The 'Total # Samples' indicates the total number of samples in the entire dataset. The 'Used # Samples' indicates the actual number of samples that were used in training, which could be no more than the total number of samples in a given dataset. 'Open Source' indicates whether the dataset is open-sourced.

Our main experiments are conducted on Llama2-7B. Specifically, we expand the block number from 32 to 38 using an interleaved approach. In the block expansion process, we only expand the first half 16 layers, setting the parameters to $L_N = 2$ and $N = 8$. This configuration leads to the formation of 8 groups, with each group expanding from 2 blocks to 3 blocks. For the code and math corpus pretraining, we adopt a batch size of 1024 and a sequence length of 4096, combined with a 2% warmup ratio. The learning rate is set at 2e-4, utilizing a Cosine learning rate decay strategy. To enhance efficiency, we use bf16 mixed precision training, apply a weight decay factor of 0.1, and institute a gradient clipping threshold of 1.0, and apply the flash-attention mechanism during training. Our 30B token experiments were conducted on 256 NVIDIA A100 GPUs to save the training time and our 5B token ablation experiments were conducted on 32 NVIDIA A100 GPUs, all trained for 2 epochs.

4.1.2 SFT details

During the instruction fine-tuning phase, we follow (Wu et al., 2024) and amalgamate four distinct data sources (WizardLM evolution instruction dataset (Xu et al., 2023), evolution CodeAlpaca dataset (Luo et al., 2023), MetaMath (Yu et al., 2023b) and SlimOrca (**?**)) to forge the final SFT dataset of our Llama Slayer-8B-Instruct. The final SFT dataset comprises upwards of 1.2M samples. To fine-tune the basic models, we employ specific configurations, including a batch size of 128, a sequence length of 4096, 0.02 warmup ratio, a learning rate of 1e-5, a Cosine learning rate scheduler, and bf16 mixed precision.

4.1.3 Evaluation Metrics

We conduct a comparative analysis of Llama2-7B expanded with our proposed S strategy with the latest state-of-the-art (SOTA) LLMs. We adopt seven datasets as benchmarks for evaluation: *ARC* (25-shot)(Clark et al., 2018), *HellaSWAG* (10-shot)(Zellers et al., 2019), *MMLU* (5-shot)(Hendrycks et al., 2020), *TruthfulQA* (0-shot) (Lin et al., 2021), *Winogrande* (5-shot) (Sakaguchi et al., 2021), *GSM8K* (5-shot) (Cobbe et al., 2021) and *HumanEval* (0-shot)(Chen et al., 2021). Also, the average scores for the seven tasks are given. Among these benchmarks, the first five are employed to test the basic knowledge abilities, and GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021) are used to test math and coding abilities, respectively. We employ the BigCode Evaluation Harness[3] to evaluate HumanEval and the Eleuther AI Language Model Evaluation Harness[4] to evaluate the other six benchmarks.

4.2 CPT & SFT Results

We evaluated the performance of our SLayer-8B model against a series of state-of-the-art pretrained models of similar size. This comparison encompassed both general purpose models, such as LLaMA Pro 8.3B (Wu et al., 2024), LLaMA2-7B (Touvron et al., 2023b), and Falcon-7B (Almazrouei et al., 2023), as well as coding-specialized models, such as CodeLLaMA (Roziere et al., 2023) and math-specialized models, such as Mammoth-7B (Yue et al.). The results are detailed in Table 2.

The results highlight that SLayer-8B effectively balances natural language processing and math and coding capabilities. Not only retains the general capabilities of its base model, LLaMA2-7B (Touvron et al., 2023b), more effectively than Llama Pro, it excels in mathematical and coding tasks. In contrast, CodeLLaMA-7B (Roziere et al., 2023) opts to compromise its overall performance to improve its coding proficiency. This enhancement is credited to our expansion strategy, which was de-

[3]https://github.com/bigcode-project/bigcode-evaluation-harness

[4]https://github.com/EleutherAI/lm-evaluation-harness

Model	Language Tasks					Math & Code Tasks		Avg.
	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K	HumanEval	
Pretrained Comparison								
LLaMA2-7B (Touvron et al., 2023b) ♢	54.18	78.59	45.87	38.76	74.03	13.12	12.83	45.34
LLaMA-7B (Touvron et al., 2023a)	50.94	77.81	35.69	34.33	71.43	8.04	10.61	41.26
CrystalCoder (7B) (Liu et al., 2023)	47.01	71.97	48.78	35.91	67.17	10.77	28.38	44.28
CodeLLaMA-7B (Roziere et al., 2023)	39.93	60.80	31.12	37.82	64.01	5.16	33.50	38.91
StarCoder-15B (Li et al., 2023)	30.38	47.93	29.96	41.28	56.12	9.48	15.32	32.92
OpenLLaMA-v2-7B (Geng and Liu, 2023)	43.69	72.20	41.29	35.54	69.38	3.49	15.32	40.13
Falcon-7B (Almazrouei et al., 2023)	47.87	78.13	27.79	34.26	72.38	4.62	9.62	39.24
Llama Pro (8.3B) ♣ (Wu et al., 2024)	53.67	77.83	47.17	37.37	72.30	17.02 (+3.9%)	20.95 (+8.1%)	46.62(+1.3%)
Llama SLayer-8B	54.35	77.46	48.25	36.13	73.88	**19.03** (+5.9%)	**22.62** (+9.8%)	**47.39**(+2.1%)
SFT Comparison								
LLaMA2-7B-Chat (Touvron et al., 2023b)	52.90	78.55	48.32	45.57	71.74	23.95	14.63	47.95
CodeLLaMA-7B-Instruct (Roziere et al., 2023)	36.52	55.44	34.54	41.25	64.56	7.96	34.80	39.30
Mammoth-7B (Yue et al.)	49.15	75.72	42.29	38.98	70.88	53.6	10.98	48.94
Falcon-7B-Instruct (Almazrouei et al., 2023)	45.82	70.78	25.66	44.07	68.03	4.70	–	–
LLAMA PRO-INSTRUCT (8.3B) ♣ (Wu et al., 2024)	51.21	76.62	50.12	41.13	71.53	50.18(+26.2%)	32.92(+18.3%)	53.39(+5.4%)
Llama Slayer-8B-INSTRUCT	48.98	76.91	50.34	44.65	71.12	**56.25**(+32.3%)	**35.15**(+20.5%)	**54.77**(+6.8%)

Table 2: Comparison of evaluation results among several prominent code and language models. The figures in bold mark the highest ones in each column. The Llama Pro (8.3B) ♣ means that this is our self-continually pre-trained version on our 30B token training corpus. The red percentage increases in parentheses are relative to Llama2-7B. The evaluation results of other pre-trained and chat models are adopted from the Open LLM Leaderboard.

veloped based on empirical research. By directing specific knowledge to the crucial layers, freezing the initial blocks of LLaMA, and training the expansion blocks initialized via interpolation, we achieve more effective knowledge injection while preserving the model's general strengths. SFT often leads to more significant improvements in the evaluation metrics compared to CPT. This is because the evaluation of an LLM assesses its understanding of questions, response standardization, and general knowledge. Although the CPT process helps the model acquire a broad range of knowledge, it may not enhance response standardization as effectively. In contrast, SFT specifically trains the model to follow instructions more accurately and generate more standardized and precise responses, leading to greater improvements in evaluation metrics.

4.3 Efficiency Comparison

We have compared in Table 4 the cost of our Llama Slayer 8.3B and Llama Pro 8B in terms of parameters and training time in our 30B token CPT dataset. The training time is fairly compared to the same settings on 8 NVIDIA A100 GPUs. Although our model's trainable parameters are slightly higher, our model still saves 5.5% training time cost compared to Llama Pro. In addition, since our model requires less parameters, it is more efficient in storage and inference.

4.4 Ablation Studies

The ablation study of different block expansion mode is shown in Table 3. In particular, here the data set for ablation is the version of the 5B token extract, which is different from the 30B CPT data set used in Table 2 to improve the efficiency of the experiment. To clarify, the notation $(2+1) \times 8|16$ means that we expand the model by adding one block for every two blocks of the head eight times, and only the eight expanded blocks marked red will be fine-tuned.

For fair comparison, we first set the trainable parameter number to the same and explored varying densities of block expansion throughout the layers. This strategy aims to identify the most advantageous segments for expansion within the LLM during continued pre-training. Our findings highlight a preference for expanding the shallow layers over a uniform distribution across the entire network.

We also explored the impact of expanding range by comparing with expanding within the first 1/3 of the layer range, rather than the first 1/2, to reduce computational costs. The results of this approach are presented in Table 2. Specifically, since the range is limited to the first 1/3 of the layers (approximately 12 layers), we expanded 1 layer for every 2 consecutive layers, repeating this process 6 times starting from the first layer. We then removed the last 2 layers and performed a weighted average on the final layer. This method is denoted as $(2+1) \times 6|17|1\diamondsuit$, where the number '1' marked in red indicates that only the red colored layers (the expanded 6 layers and the final 1 layer) get tuned. The result is that the overall performance of expanding within the first 1/3 of the layers is lower compared to expanding within the first 1/2 of the layers. This can be attributed to the observations in Figure 2(a), which shows a significant drop in

Block Expansion Mode (5B token data)	Language Tasks					Math & Code Tasks		Avg.
	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K	HumanEval	
LLaMA2-7B (Touvron et al., 2023b)	54.18	78.59	45.87	38.76	74.03	13.12	12.83	45.34
$(4+1)\times 8$ (Llama Pro)	51.37	78.12	44.36	37.42	72.69	14.73	18.08	45.25
$(2+1)\times 3 \vert (4+1)\times 3 \vert (7+1)\times 2$	52.05	78.06	44.88	37.09	73.32	15.19	18.23	45.55
$(2+1)\times 3 \vert (7+1)\times 2 \vert (4+1)\times 3$	52.13	77.85	45.37	37.07	73.48	14.86	18.02	45.54
$(4+1)\times 3 \vert (2+1)\times 3 \vert (7+1)\times 2$	52.47	78.16	45.46	38.32	73.70	15.54	18.69	46.05
$(4+1)\times 3 \vert (7+1)\times 2 \vert (2+1)\times 3$	52.22	78.02	44.55	38.42	73.01	15.39	18.20	45.77
$(7+1)\times 2 \vert (4+1)\times 3 \vert (2+1)\times 3$	52.13	79.23	45.44	35.61	74.74	14.71	17.68	45.65
$(7+1)\times 2 \vert (2+1)\times 3 \vert (4+1)\times 3$	51.02	77.88	45.66	36.37	74.27	15.01	17.59	45.40
$16 \vert (2+1)\times 8$	52.88	78.01	44.54	37.28	73.48	14.72	16.68	45.37
$(2+1)\times 8 \vert 16$	53.05	77.10	44.52	37.98	73.11	15.96	19.50	45.89
$(2+1)\times 8 \vert 16\diamondsuit$	53.16	77.25	44.95	38.28	73.48	16.09	19.52	46.10
$(2+1)\times 6 \vert 17 \vert 1\diamondsuit$	51.16	77.20	45.89	37.18	72.96	15.52	18.51	45.49
$(2+1)\times 8 \vert 11 \vert 1\diamondsuit$ (delete 4 blocks, Ours)	49.83	75.52	45.06	37.33	73.24	13.72	16.08	44.40
$(2+1)\times 8 \vert 12 \vert 1\diamondsuit$ (delete 3 blocks, Ours)	50.15	76.02	45.22	37.64	73.30	15.13	17.96	45.06
$(2+1)\times 8 \vert 13 \vert 1\diamondsuit$ (delete 2 blocks, Ours)	51.45	77.73	46.26	37.91	73.56	**16.37**	**19.73**	**46.14**

Table 3: Comparison of evaluation results among several prominent code and language models. The numbers in red denote that the expanded blocks and only these blocks get fine-tuned at the continued pre-training stage. Methods marked with $\diamondsuit$ indicate that the expanded blocks use weight averaging for initialization. In contrast, other methods use identity copy-for initialization.

Table 4: Comparison of parameters and training time.

Method	Trainable Parameters (B)	Total Parameters (B)	CPT Time Cost (h)
Llama Pro	1.75	8.3	740
Llama Slayer-8B	1.95	7.9	700

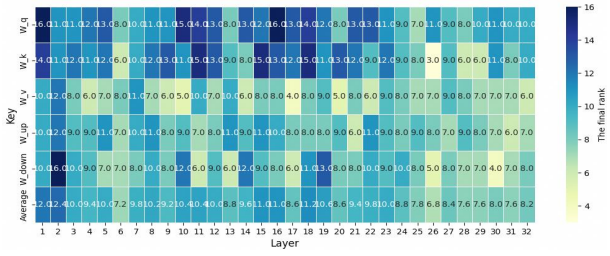

Figure 4: The resulting rank of each incremental matrix when fine-tuning Llama2-7B on our 5B token math+code dataset with AdaLoRA.

angular distance from the middle (16th) layer. This suggests that the layers beyond the first 1/3 play a crucial role in maintaining or improving model performance.

To further demonstrate the importance of shallow layers, we fine-tune Llama2-7B on our 5B token math code dataset with AdaLoRA (Zhang et al., 2023), which can dynamically allocate parameter budgets to weight matrices based on importance ratings. We display the resulting rank distribution and the average ranks of each incremental matrix during AdaLoRA fintuing in Fig. 4. As shown in Fig. 4, AdaLoRA predominantly enriches the weight matrices in the shallow layers, corroborating our insight that these layers are more crucial for infusing new knowledge. Furthermore, we also calculate the angular distances for Llama Slayer (13.60) and Llama Pro (13.17) after training on our compiled 30B token math+code dataset and observe that the former surpasses the latter, again proving the effectiveness of our approach.

Furthermore, we evaluate the impact of initialization methods and the acceptable limits for layer reduction in Table 3. We find that weight averaging is a superior method for initializing expanded blocks, since this adds coherence to the expanded LLM. Additionally, we discern that eliminating up to two of the deepest layers is feasible without severely degrading model performance. However, a more aggressive approach to layer deletion has been found to adversely affect the effectiveness of the LLM.

Apart from the aspect of code and math corpus, we also explore our training strategy on a different LLM type, Mistral-7B, and another knowledge domain: law, with the free-law subset of the Pile dataset as our pre-training corpus (Gao et al., 2020). To assess our model's proficiency in legal language, we leveraged the Unfair ToS dataset, which is composed of Terms of Service (ToS) agreements from various online platforms—a critical resource in evaluating legal document comprehension. Our evaluation was carried out using the UNFAIR-ToS benchmark (Lippi et al., 2019) within LexGLUE (Chalkidis et al., 2021), employing a 5-shot learning scenario. The evaluation was also implemented through the Eleuther AI Language Model Evaluation Harness.

In Fig. 5, we detail a comparative analysis of

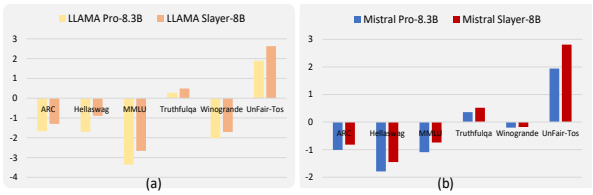

Figure 5: Comparative analysis of performance variations of different training strategies relative to the base model (a) Llama2-7B and (b) Mistral-7B, on both general and law-specific tasks.

performance variations between our Llama/Mistral Slayer-8B models and the self-post pretrained Llama/Mistral Pro, benchmarked against the corresponding foundational models. It is evident from our findings that our tailored strategy significantly enhances domain-specific knowledge injection while more effectively mitigating the issue of catastrophic forgetting compared to the Llama Pro's even block expansion. This improvement can be largely attributed to our idea of expanding LLM blocks based on their layer importance. Such an approach ensures that our knowledge augmentation efforts are concentrated on the most crucial blocks, thereby enhancing the efficiency of post-pretraining. In addition, our utilization of the weight-averaging technique for initializing the expanded blocks provides a smoother approach compared to the direct identity copy employed in Llama Pro. This ensures better layer coherence in the model, as opposed to the abrupt identity copy method. Therefore, our proposed strategy not only fosters smoother integration and adaptation of new knowledge, but also contributes to the refined performance and learning capabilities of our models within domain-specific contexts.

5 Conclusion

We propose S strategy to inject domain-specific knowledge into LLMs via block expansion based on layer importance in the post-pretraining phase. This strategy prioritizes knowledge injection to the important shallow layers while pruning the ineffective deep layers, and can not only bolster the model's specific-domain capabilities but also maintain its general proficiency. Based on the proposed strategy, we introduce LLAMA SLayer-8B and LLAMA SLayer-8B-INSTRUCT, LLMs that derive from the base model LLaMA2-7B. The two models surpass various predecessors in the LLaMA series across a wide array of benchmarks, evidencing the superior performance of our strategy.

6 Limitations

While we introduces an effective strategy based on layer importance to post-pretrain Large Language Models (LLMs) to achieve a balance between general and domain-specific abilities, its applicability is primarily within the realm of language. Future investigations could aim to broaden the utility of our knowledge injection strategy across various fields. This expansion might include adapting the technique to enhance the native language capabilities of multimodal LLMs (Ge et al., 2023), as well as its application in multilingual settings. Furthermore, the adaptability of our proposed strategy to larger LLMs and the feasibility of removing an increased number of deeper layers as the model size escalates are promising avenues for future research.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Takuya Akiba, Makoto Shing, Yujin Tang, Qi Sun, and David Ha. 2024. Evolutionary optimization of model merging recipes. *arXiv preprint arXiv:2403.13187*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Dan Biderman, Jose Gonzalez Ortiz, Jacob Portes, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, et al. 2024. Lora learns less and forgets less. *arXiv preprint arXiv:2405.09673*.

Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Martin Katz, and Nikolaos Aletras. 2021. Lexglue: A benchmark dataset for legal language understanding in english. *arXiv preprint arXiv:2110.00976*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Stephen Choi and William Gazeley. 2024. When life gives you llms, make llm-ade: Large language models with adaptive data engineering. *arXiv preprint arXiv:2404.13028*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*.

Raj Dabre, Atsushi Fujita, and Chenhui Chu. 2019. Exploiting multilingualism through multistage fine-tuning for low-resource neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1410–1416.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Yuying Ge, Sijie Zhao, Ziyun Zeng, Yixiao Ge, Chen Li, Xintao Wang, and Ying Shan. 2023. Making llama see and draw with seed tokenizer. *arXiv preprint arXiv:2310.01218*.

Xinyang Geng and Hao Liu. 2023. Openllama: An open reproduction of llama. *URL: https://github.com/openlm-research/open_llama*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, et al. 2023. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166*.

Tao Lei, Junwen Bai, Siddhartha Brahma, Joshua Ainslie, Kenton Lee, Yanqi Zhou, Nan Du, Vincent Zhao, Yuexin Wu, Bo Li, et al. 2024. Conditional adapters: Parameter-efficient transfer learning with fast inference. *Advances in Neural Information Processing Systems*, 36.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*.

Marco Lippi, Przemysław Pałka, Giuseppe Contissa, Francesca Lagioia, Hans-Wolfgang Micklitz, Giovanni Sartor, and Paolo Torroni. 2019. Claudette: an automated detector of potentially unfair clauses in online terms of service. *Artificial Intelligence and Law*, 27:117–139.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, et al. 2023. Llm360: Towards fully transparent open-source llms. *arXiv preprint arXiv:2312.06550*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterfusion: Non-destructive task composition for transfer learning. *arXiv preprint arXiv:2005.00247*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Haochun Wang, Chi Liu, Nuwa Xi, Zewen Qiang, Sendong Zhao, Bing Qin, and Ting Liu. 2023. Huatuo: Tuning llama model with chinese medical knowledge. *arXiv preprint arXiv:2304.06975*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ping Luo, and Ying Shan. 2024. Llama pro: Progressive llama with block expansion. *arXiv preprint arXiv:2401.02415*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

Lingling Xu and Weiming Wang. 2023. Improving aspect-based sentiment analysis with contrastive learning. *Natural Language Processing Journal*, 3:100009.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. 2024. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. 2023a. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview.

A Evaluation Benchmark

The benchmarks used for evaluation include:

- AI2 Reasoning Challenge (Clark et al., 2018)(25-shot): a set of grade-school science questions.
- HellaSwag (10-shot) (Zellers et al., 2019): a test of commonsense inference, which is easy for humans (approximately 95%) but challenging for SOTA models.
- TruthfulQA (0-shot) (Lin et al., 2021): a test to measure a model's propensity to reproduce falsehoods commonly found online.
- MMLU (5-shot) (Hendrycks et al., 2020): a test to measure a text model's multitask accuracy. The test covers 57 tasks including elementary mathematics, US history, computer science, law, and more.
- Winogrande (5-shot) (Sakaguchi et al., 2021): an adversarial and difficult Winograd benchmark at scale, for commonsense reasoning.
- GSM8k (5-shot) (Cobbe et al., 2021): diverse grade school math word problems to measure a model's ability to solve multi-step mathematical reasoning problems.
- HumanEval (0-shot) (Chen et al., 2021): 164 handwritten Python programming problems with a function signature, docstring, body, and several unit tests.

Table 5: Hyper-parameters of pretraining on the domain of law.

HYPER-PARAMETERS	ASSIGNMENT
BATCH SIZE	1024
MAXIMUM SEQUENCE LENGTH	2,048
MAXIMUM LEARNING RATE	2E-4
OPTIMIZER	ADAM
ADAM BETA WEIGHTS	0.9, 0.95
LEARNING RATE SCHEDULER	COSINE
WARMUP RATIO	0.02
GRADIENT CLIPPING	1.0
EPOCH	2

B Hyper-parameters of pretraining on the domain of law.

We list the detailed settings of our pretraining on the domain of law in Table 5.

C More detailed explanation of Figure 4.

Figure 4 shows the resulting rank of each incremenal matrix when fine-tuning Llama2-7B on our 5B toke nmath+dode data with AdaLoRA. The y axis has six weight items, $W_q, W_k, W_v, W_{up}, W_{down}$

and the average weight value. From the last average line we can see that AdaLoRA allocates more parameter budget to the weight matrices in shallow layers, since the color gradually becomes lighter as the layer gets deeper. Such behavior aligns with our conclusion that the shallow layers are more important to knowledge injection and should be paid more attention during post-pretraining.