LLAMA PRO: Progressive LLaMA with Block Expansion

Anonymous ACL submission

Abstract

Humans generally acquire new skills without compromising the old; however, the opposite holds for Large Language Models (LLMs), *e.g.*, from LLaMA to CodeLLaMA. To this end, we propose a new post-pretraining method for LLMs with an expansion of Transformer blocks. We tune the expanded blocks using only new corpus, efficiently and effectively improving the model's knowledge while mitigating forgetting. In this paper, we experiment on the corpus of code and math, yielding **LLAMA PRO-8.3B**, a versatile foundation model initialized from LLaMA2-7B, excelling in general tasks, programming, and mathematics. LLAMA PRO and its instruction-following counterpart (LLAMA PRO - INSTRUCT) achieve advanced performance among various benchmarks, demonstrating superiority over existing open models in the LLaMA family and the immense potential of reasoning and addressing diverse tasks as an intelligent agent. Our findings provide valuable insights into integrating natural and programming languages, laying a solid foundation for developing advanced language agents that operate effectively in various environments.

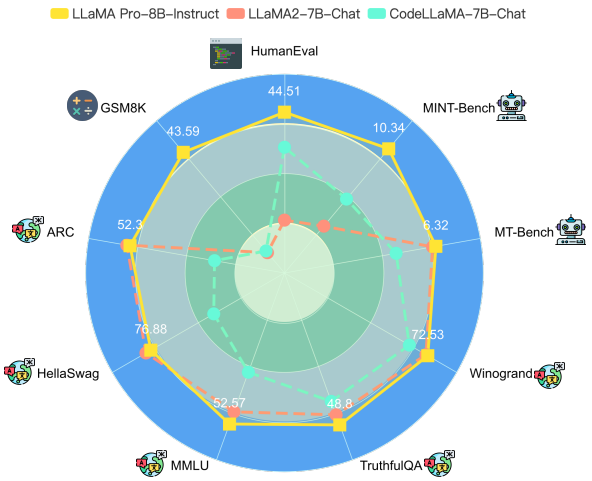


Figure 1: LLAMA PRO - INSTRUCT delivers state-of-the-art performance across a wide variety of tasks, ranging from general language to specific domains, superior to existing models from the LLaMA series.

1 Introduction

The advent of Large Language Models (LLMs) has revolutionized the field of natural language processing, exhibiting remarkable proficiency in a variety of real-world tasks (OpenAI, 2023; Chowdhery et al., 2023). Despite the versatility, LLMs still fall short in certain domains, for example, programming, mathematics, biomedical, or finance. This limitation impedes the progress of developing generic language agents for broader applications.

Existing works (Liu et al., 2023; Li et al., 2023a; Wu et al., 2023b) attempted to improve the multi-faceted capabilities of pre-trained LLMs with tailored data recipes. While feasible, they require substantial computational resources and vast amounts of data, which poses a challenge to the democratization of LLM research. Consequently, another line of research, known as domain-adaptive pre-training, focuses on post-pretraining with domain-specific corpora (Gururangan et al., 2020). These approaches have demonstrated efficacy in adapting various LLMs to specific domains (Roziere et al., 2023; Azerbayev et al., 2023; Wu et al., 2023b; Xu et al., 2023b), resulting in enhanced performance on downstream domain-specific tasks at a reduced computational cost.

Nonetheless, a considerable obstacle emerges in catastrophic forgetting (De Lange et al., 2021). Post-pretraining often leads to a decline in the model's original general abilities, inhibiting the fine-tuned performance of the model on diverse tasks (Cheng et al., 2023; Dong et al., 2023). This necessitates a method that can inject domain-specific knowledge into LLMs while preserving their general abilities, thereby enhancing their comprehensive capabilities.

Towards this end, we introduce a simple yet ef-

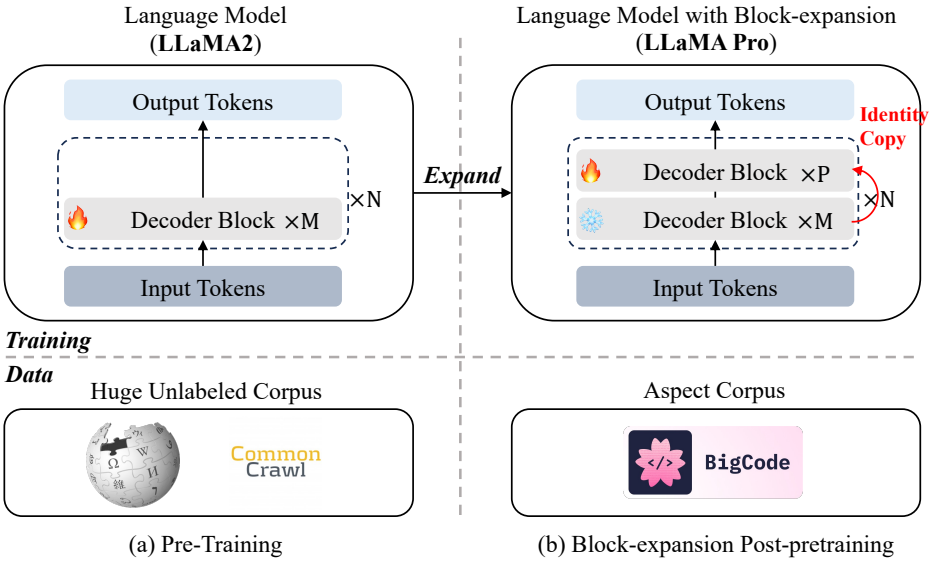


Figure 2: (a) We begin with a large language model (LLM) pre-trained on a massive unlabeled corpus, resulting in a model with strong general capabilities. Here we select the off-the-shelf LLaMA2 for convenience. (b) We employ backbone expansion and fine-tune the expanded identity blocks using the aspect corpus while freezing the blocks inherited from the base model. The model after post-pretraining can be used for instruction tuning as usual.

fective post-pretraining method, termed *block expansion*. We expand the off-the-shelf pre-trained LLM using copied Transformer blocks, as illustrated in Figure 2. The newly added blocks, whose linear layers are zero-initialized to enable identity mapping, are further tuned with only domain-specific corpus while the remaining blocks are frozen. After tuning, the extended pre-trained model excels in both general and domain-specific tasks.

In practice, we extend the pre-trained LLaMA2-7B (Touvron et al., 2023) by eight more blocks, yielding LLAMA PRO, a foundation model with 8.3B parameters, and enhanced performance in programming, coding, and reasoning. We pre-train LLAMA PRO's expanded blocks on 80B tokens using open-source code and math data for 2830 GPU Hours (16 NVIDIA H800 GPUs for about 7 days). We further perform supervised instruction tuning (fully fine-tuning of all the blocks, *aka* SFT) on LLAMA PRO with approximately 80M tokens, yielding LLAMA PRO - INSTRUCT. It is noted that pre-trained models produced by our block expansion method are well-compatible with the subsequent SFT techniques without specific modification.

As shown in Figure 1, LLAMA PRO - INSTRUCT reaches state-of-the-art performance across a broad range of general, code (*i.e.*, HumanEval), and math (*i.e.*, GSM8K) tasks. Furthermore, we assess the capabilities of LLAMA PRO - INSTRUCT as a language agent across various scenarios (*i.e.*, MINT-Bench), with a focus on the tool usage abilities and the capacity to ground in environmental and human feedback. We also employ GPT-4 (OpenAI, 2023) automatic evaluation to assess LLAMA PRO's ability to serve as an effective assistant (*i.e.*, MT-Bench). Comprehensive experimental results indicate the superiority of LLAMA PRO - INSTRUCT over other models from the LLaMA family on both benchmarks and practical applications. Our contributions are three-fold:

- We propose a novel post-pretraining method for LLMs, termed block expansion, enabling the injection of new knowledge while preserving the initial capabilities.

- We introduce LLAMA PRO and LLAMA PRO - INSTRUCT, versatile LLMs that well integrate natural and programming languages, excelling in general tasks, programming, and mathematics.

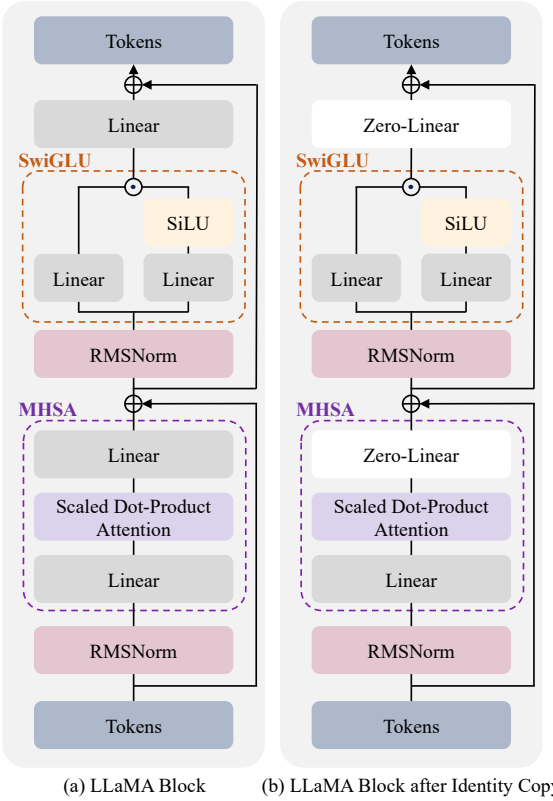
- We benchmark the family of LLAMA PRO on extensive datasets, including both traditional and agent-oriented tasks, demonstrating its superiority and great potential in broader complex applications.

2 Related Work

Advancements in Large Language Models. Recent advancements in large language models have led to significant progress, with model and data scale growth driving state-of-the-art performance across various tasks (Hoffmann et al., 2022; Kaplan et al., 2020; Chowdhery et al., 2023). The development of generalist models has enabled addressing diverse problems and rapid adaptation to new tasks (Radford et al., 2019; Brown et al., 2020). The open-source community has further contributed by releasing powerful models like LLaMA (Touvron et al., 2023) and CodeLLaMA (Roziere et al., 2023). Our work builds upon these developments, providing a method for specializing LLMs in the code domain, fostering future research and applications.

Post-pretraining. Language model applications typically involve a two-step process: general-domain pretraining followed by domain-specific training (Roziere et al., 2023; Azerbayev et al., 2023). Fine-tuning often aims to enhance instruction-following abilities (Sanh et al., 2021; Wei et al., 2021; Wang et al., 2023d) or align model outputs with human preferences (Ziegler et al., 2019; Ouyang et al., 2022; Bai et al., 2022). Some research explores parameter-efficient fine-tuning methods for adapting pretrained models to new domains (Houlsby et al., 2019; Hu et al., 2021; Wu et al., 2023a), while others focus on continual learning post-pretraining (Wang et al., 2023b; Gupta et al., 2023; Scialom et al., 2022). Our work proposes an adaptation strategy that combines continued training with general capability maintenance, allowing LLMs to specialize without sacrificing overall performance.

Progressive Learning. Progressive training has gained attention for accelerating large-scale model training in computer vision (Zhang et al., 2023) and NLP research (Yao et al., 2023; Li et al., 2023b). Gong et al. (2019) proposed a stacking method doubling model depth successively. CompoundGrow(Gu et al., 2020) extends stacking with Feed-Forward Network expansion in schedule design. Shen et al. (2022) introduced a staged method supporting hidden size expansion. Bert2BERT(Chen et al., 2021a) and LiGO (Wang et al., 2023a) accommodate all growth dimensions. Our method utilizes depth growth to maintain general performance while adapting to specific domains.



(a) LLaMA Block (b) LLaMA Block after Identity Copy

Figure 3: (a) An overview of the LLaMA Block, comprising an MHSA mechanism followed by the FFN with SwiGLU activation. (b) The Identity LLaMA block after an identity copy, achieved by initializing the output linear matrix to zero in order to preserve the output from the base LLaMA model.

3 Method

3.1 Preliminaries: The LLaMA Block

The LLaMA block consists of a multi-head self-attention (MHSA) mechanism followed by a position-wise feed-forward network (FFN) with residual connections and a Swish-Gated Linear Unit (SwiGLU) operation as Figure 3 shows. Given an input x, the LLaMA block produces an output y as described by the following equations:

$$\begin{aligned} x' &= x + \text{MHSA}(\text{RMSNorm}(x)) \\ y &= x' + \text{FFN}(\text{RMSNorm}(x')) \end{aligned} \tag{1}$$

The input x has a dimension of $n \times d$, where n is the sequence length and d is the hidden size. The output y has the same dimension as the input x. The MHSA operation is a crucial component of the transformer, defined as:

$$\text{MHSA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2}$$

where Q, K, and V are the query, key, and value matrices, respectively, and W^O is the output weight

matrix without bias . Each head is computed as:

$$\text{head}_i = \text{Attention}(xW_i^Q, xW_i^K, xW_i^V)$$
$$\text{Attention}(Q_i, K_i, V_i) = \text{Softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \tag{3}$$

with W_i^Q, W_i^K, and W_i^V being the corresponding weight matrices for the i-th head.

The FFN block in the LLaMA model utilizes the SwiGLU activation function, which is defined as:

$$\begin{aligned} \text{SwiGLU}(x, W, V) &= \text{SiLU}(xW) \otimes (xV) \\ \text{FFN}(x) &= \text{SwiGLU}(x, W_1, W_2)W_3 \end{aligned} \tag{4}$$

where $\otimes$ denotes element-wise multiplication, W_1, W_2, and W_3 are the weight matrices without bias, $\text{SiLU}(x) = x \otimes \sigma(x)$.

3.2 Block Expansion

Given a model with blocks $(\phi_0, \phi_1, ..., \phi_L)$, the block expansion incorporates an identity block ϕ_{id} after each block in the original model, ensuring that the expanded model maintains the same output after expansion. The identity block is defined as $\phi_{id}(x) = x$ so the input and output are identical.

Suppose we have an initial model with L blocks that needs to be expanded to L' blocks. First, we partition the original L blocks into N groups, with each group containing $\frac{L}{N}$ blocks. For each group, we create identity copies of the top P blocks and stack them on top of each group, as depicted in Figure 3. We arrange these blocks in an interleaved manner to maintain the structural characteristic of the transformer model, whose prior is that deeper blocks encode more complex information (Van Aken et al., 2019; Tenney et al., 2019). This process leads to an increased depth in the model while maintaining its output behavior.

Shen et al. (Shen et al., 2022) proposed the initialization of scale parameters in the Norm modules within the identity blocks to zero for the construction of the identity block. However, this approach may not be effective when applied to the LLaMA block. The reason lies in the fact that the gradient of the loss function L with respect to the RMSNorm weight w during backpropagation would be zero. This would prevent the training of RMSNorm, implying that when $\text{RMSNorm}(x') = 0$, the following condition will hold:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \frac{\partial \text{FFN}(\text{RMSNorm}(x'))}{\partial \text{RMSNorm}(x')} \frac{\partial \text{RMSNorm}(x')}{\partial w} = 0. \tag{5}$$

This equation signifies that the gradient of the loss function with respect to the weight of RMSNorm is zero, which would hinder the training of the RMSNorm module. This is further explained in Appendix A. Referring to the LLaMA block formulation in Equation 1, the identity can be achieved as long as $\text{MHSA}(\text{RMSNorm}(x)) = \mathbf{0}$ and $\text{FFN}(\text{RMSNorm}(x')) = \mathbf{0}$. We initialize the W^O and W_3 weight matrices in the identity blocks to zero. Due to the presence of residual connections and the absence of bias terms in the LLaMA block, only the residual flows through the identity block. As a result, the entire block is reduced to an identity block at initialization, preserving the output from the initial model.

The entire training pipeline is depicted in Figure 2. Our method concentrates on the post-pretraining stage, targeting specific domain corpora. We begin by initializing our model with large language models trained on extensive unlabeled general corpora, where all blocks will be fine-tuned. To enhance the model's capacity for accommodating additional domain knowledge while retaining its general knowledge, we employ block expansion to increase the number of blocks in the LLM. During this process, we only fine-tune the newly added blocks while freezing the original blocks, thereby preserving the general abilities of the model.

4 Experiments

This section presents our key experimental findings. We begin with experimental settings (described in Sec. 4.1), and then verify the effectiveness of block expanded tuning after pretraining (described in Sec. 4.2). Next, we give the supervised finetuning (SFT) results (described in Sec. 4.3). Finally, ablation studies of the key design choices are presented (described in Sec. 4.4).

4.1 Experimental Settings

Pretrain details. We construct a dataset that concentrates on code and math. For the code component, we rely on the Stack-dedup dataset, which is a compilation of permissively licensed source codes from GitHub. Among all the programming languages available in Stack-dedup, we specifically utilize the Python split. As for the math component, we opt for the Proof-pile-2 dataset (Azerbayev et al., 2023), a 55-billion-token amalgamation of scientific papers, web data containing mathematical content, and mathematical code. The details

can be found in Appendix B.

We initialize our base model with LLaMA2-7B and expand the number of blocks from 32 to 40 using an interleaved approach. In the block expansion process, we configure the parameters as $P = 1$, $M = 4$, and $N = 8$, resulting in 8 groups where each group expands from 4 blocks to 5 blocks. For the code and math corpus pretraining, we employ a batch size of 1024, a sequence length of 4096, a warmup ratio of 6%, a learning rate of 2e-4, and a Cosine learning rate scheduler. We also use bf16 mixed precision, a weight decay of 0.1, and gradient clipping at 1.0. To speed up the training process, we apply the flash-attention mechanism.

Our experiment is conducted on 16 NVIDIA H800 GPUs. LLAMA PRO is trained for a total of 15,900 steps. This training process corresponds to approximately 2830 H800 GPU hours.

SFT details. During the instruction fine-tuning phase, we combine five data sources to create LLAMA PRO - INSTRUCT as shown in Table 6. The final sft dataset consists of approximately 1M samples. To fine-tune the basic models, we employ specific configurations, including a batch size of 128, a sequence length of 4096, 0.03 warmup ratio, a learning rate of 2e-5, a Cosine learning rate scheduler, and bf16 mixed precision.

Evaluation details. We conduct a comparative analysis of LLAMA PRO with the latest state-of-the-art (SOTA) Large Language Models (LLMs). The evaluation is performed on six key general benchmarks using the Eleuther AI Language Model Evaluation Harness[1], a unified framework designed to test generative language models across a vast array of evaluation tasks. For code-related tasks, we employ the BigCode Evaluation Harness[2] to evaluate HumanEval and MBPP, and we report the pass@1 rate of code tasks with greedy decoding. The evaluation details can be found in Appendix C.

4.2 Pretrain Results

We evaluate LLAMA PRO's performance with benchmark datasets from the Open LLM Leaderboard. Furthermore, we incorporate coding benchmark datasets, including HumanEval pass@1 and MBPP pass@1, as well as the math benchmark GSM8K, to provide a comprehensive evaluation.

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

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

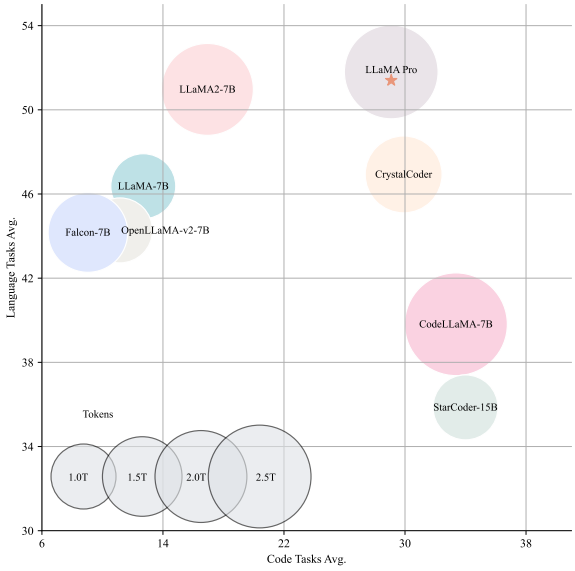


Figure 4: We compare LLAMA PRO's general performance and code performance to a set of models trained around the same time, spanning from general LLMs to code-oriented LLMs. The size of the blobs is proportional to the number of tokens trained. Mistral-7B is not included here, as the number of tokens is not reported in its paper.

We compare the performance of LLAMA PRO with a selection of state-of-the-art pretrained models that were trained around the same period with similar size. This includes general-purpose pretrained models like LLaMA2 and code-oriented pretrained models like CodeLLaMA. The results are presented in Table 1.

The results highlight that LLAMA PRO effectively balances natural language processing and coding capabilities. It not only preserves the general performance of its base model, LLaMA2-7B, but also surpasses it in the average performance of general language tasks. Conversely, CodeLLaMA-7B sacrifices general performance. We attribute this improvement to our expansion design, which freezes the initial LLaMA blocks to maintain their capabilities and increases the blocks to accommodate domain-specific knowledge.

As depicted in Figure 4, LLAMA PRO shows robust general performance alongside code performance that is on par with code-oriented LLMs. Situated on the Pareto frontier, LLAMA PRO has undergone fine-tuning with an additional 80B tokens in conjunction with LLaMA2, which more than doubles the code tasks average performance. In contrast, CodeLLaMA is fine-tuned with 500B tokens. LLAMA PRO excels in general performance

Model	Language Tasks					Math Tasks		Code Tasks		Avg.
	ARC	HellaSwag	MMLU	TruthfulQA	Winogrande	GSM8K	GSM8K-PoT	HumanEval	MBPP	
Pretrained comparison										
LLAMA PRO (8B)	54.10	77.94	47.88	39.04	73.95	17.89	25.42	28.66	33.20	**44.23**
CrystalCoder (7B)	47.01	71.97	48.78	35.91	67.17	10.77	24.96	28.38	36.38	41.26
LLaMA2-7B	53.07	78.59	46.87	38.76	74.03	14.48	17.68	13.05	20.09	39.62
CodeLLaMA-7B	39.93	60.80	31.12	37.82	64.01	5.16	25.20	33.50	41.40	37.66
StarCoder-15B	30.38	47.93	29.96	41.28	56.12	9.48	25.09	33.63	43.28	35.24
LLaMA-7B	50.94	77.81	35.69	34.33	71.43	8.04	10.46	10.61	17.04	35.15
OpenLLaMA-v2-7B	43.69	72.20	41.29	35.54	69.38	3.49	5.46	15.32	12.69	33.23
Falcon-7B	47.87	78.13	27.79	34.26	72.38	4.62	4.32	9.42	13.39	32.46
SFT comparison										
LLAMA PRO - INSTRUCT	52.30	76.88	52.57	48.80	72.53	43.59	55.61	44.51	37.88	**53.85**
LLaMA2-7B-Chat	52.90	78.55	48.32	45.57	71.74	7.35	19.73	14.63	21.60	40.04
CodeLLaMA-7B-Instruct	36.52	55.44	34.54	41.25	64.56	7.96	34.67	34.80	44.4	39.35
WizardCoder-Python-7B	41.81	65.06	32.29	36.32	61.72	4.70	17.60	42.07	47.20	38.75
WizardMath-7B	54.10	79.55	45.97	43.65	72.69	2.73	25.57	12.20	18.00	39.38

Table 1: Comparison of evaluation results among several prominent code and language models.

Model	MT Bench
Alpaca-13B	4.53
CodeLLaMA-7B-Instruct	5.71
Vicuna-7B	6.17
LLaMA2-7B-Chat	6.27
LLAMA PRO - INSTRUCT	**6.32**

Table 2: GPT-4 automatic evaluation of Chatbot models. LLAMA PRO - INSTRUCT outperforms widely used LLaMA community chatbots.

while maintaining code performance that is competitive with code-oriented LLMs, whether they are trained from scratch, such as StarCoder-15B and CrystalCoder, or fine-tuned like CodeLLaMA-7B.

4.3 SFT Results

Modern LLMs typically undergo supervised fine-tuning or instruction tuning after pretraining on vast amounts of unlabeled data. In this section, we aim to demonstrate that our expansion strategy can adapt to this widely used training pipeline, just as traditional LLMs do.

Table 1 presents a comparison of evaluation results among several prominent supervised fine-tuning (SFT) LLMs from the LLaMA community, across general tasks, math tasks, and code tasks benchmarks. As a singular SFT model, LLAMA PRO - INSTRUCT attains state-of-the-art performance, even when compared to specifically tuned models such as WizardCoder and WizardMath. This demonstrates its more comprehensive capabilities.

As seen in Figure 1, LLAMA PRO - INSTRUCT boosts both code and math tasks to SOTA performances while maintaining reliable general performance. We enhance the average performance of LLaMA2-7B-chat and CodeLLaMA-7B-instruct by 13.81% and 14.50% respectively, which highlights the benefits of balancing textual and coding abilities.

To assess the comprehensive conversational performance of the LLAMA PRO - INSTRUCT assistant, we evaluate it using the MT-Bench with GPT-4 automatic scoring, as proposed by Vicuna (Zheng et al., 2023). As depicted in Table 2, LLAMA PRO - INSTRUCT surpasses widely used chatbots from the LLaMA community. This indicates its potential as a chatbot capable of providing helpful responses, in addition to its impressive performance in traditional benchmarks. The details of MT-Bench can be found in the Appendix E.

We use MINT-Bench (Wang et al., 2023c) to evaluate our model's ability to solve multi-turn interactions by using tools. MINT-Bench tests LLMs' ability to use tools by generating and executing Python code, focusing on tool-augmented task-solving and leveraging natural language feedback. MINT includes eight datasets covering reasoning, code generation, and decision-making. The details of MINT can be found in the Appendix D. The results are shown in Table 3. LLAMA PRO - INSTRUCT achieves SOTA performance compared to similar size models in multi-turn interactions with the use of tools.

Model	Interaction Turns					Avg.
	1	2	3	4	5	
AgentLM-7B	0.0	4.44	5.29	6.48	7.34	4.71
CodeLLaMA-7B-Instruct	0.34	7.85	10.24	9.73	8.70	7.37
LLaMA2-7B-Chat	1.02	4.27	6.66	6.48	7.34	5.77
Mistral-Instruct-v0.1	1.54	12.12	13.31	14.16	13.99	**11.02**
LLAMA PRO - INSTRUCT	0.68	12.63	11.95	11.95	**14.68**	10.38

Table 3: In the tool-augmented reasoning assessments, we evaluate the model's proficiency in integrating tools into its reasoning workflow. The model's effectiveness is measured by its success rate across various stages of interaction.

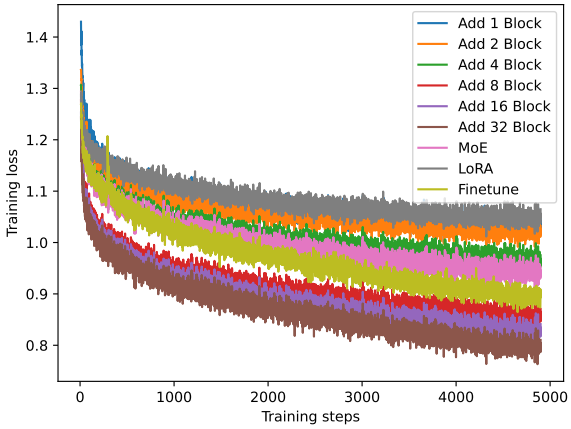


Figure 5: The training loss is analyzed with respect to the addition of varying blocks and mixture-of-expert (MoE) expansion, in conjunction with traditional training strategies such as finetuning and LoRA.

4.4 Ablation Study

Apart from the aspect of code corpus, we explore our method on another domain: law, with the freelaw subset of Pile dataset as our pretrain corpus (Gao et al., 2020). We evaluate on UNFAIR-ToS (Lippi et al., 2019) of the LexGLUE benchmark (Chalkidis et al., 2021).

In our experiment, we assess the scalability of our block expansion method in terms of training loss and downstream task performance as we increase the number of added blocks. We also compare our method with the Mixture-of-Expert (MoE) expansion method (Fedus et al., 2022) and traditional training strategies, such as fine-tuning and LoRA (Hu et al., 2021). The details can be found in Appendix G.

We analyze the training loss with varying added blocks (Figure 5). The loss consistently decreases during training, regardless of the number of added blocks, and decreases more rapidly with larger models. These findings indicate that our method demonstrates strong scalability with larger models and more data.

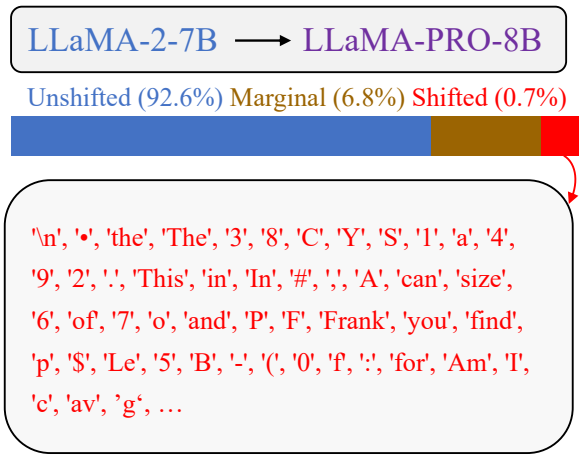


Figure 6: Token distribution shift after block expansion compared to the initial LLaMA-2-7B. The proportions of unshifted, marginally shifted, and significantly shifted tokens are color-coded and presented as percentages. Frequently shifted tokens are displayed below.

However, a lower overall training loss does not necessarily guarantee superior performance on domain-specific tasks. Therefore, we evaluate models of different sizes on both general language tasks and Unfair-ToS, as shown in Table 4. All the expanded models effectively preserve the general capabilities of the initial model. For the domain-specific task, larger models achieve better performance. We find that adding eight blocks provides optimal performance with minimal cost compared to larger models, hence we adopt this as our default strategy. The performance of MoE is comparable to our method with four added blocks. Figure 9 illustrates the differences between traditional training strategies such as fine-tuning and LoRA, and our proposed method. We observe that while LoRA effectively preserves the general ability, it struggles to model the distribution of a new domain, as also evidenced by the training loss depicted in Figure 5. In contrast, full fine-tuning results in a more significant drop in general performance.

In line with the approach of Lin et al. (2023), we analyze the token distribution between the original LLaMA and LLAMA PRO to assess the similarity in their behavior when answering general questions from the Alpaca dataset (Taori et al., 2023). As depicted in Figure 6, the token distribution shift between LLaMA and LLAMA PRO is subtle. Detailed information can be found in Appendix F.

We also analyze the impact of the position where the identity blocks are added, either at the bottom or the top of the model, compared to adding them interleaved, as shown in Table 4. We observe that

Method	Language Tasks						Law Task	Avg.
	ARC	HellaSwag	MMLU	TruthfulQA	Winogrand	Avg.	Unfair-ToS	
Add 1 Block	52.30	77.92	38.62	37.80	73.16	55.96	67.45	61.71
Add 2 Block	53.16	77.91	39.62	38.92	73.01	56.52	69.57	63.05
Add 4 Block	52.39	76.92	37.30	40.53	72.22	55.87	71.31	63.59
Add 8 Block	52.90	76.63	41.74	39.83	72.38	56.70	75.11	**65.91**
Add 16 Block	51.88	76.59	41.35	40.13	71.82	56.35	**75.17**	65.76
Add 32 Block	50.77	76.72	40.68	41.66	72.77	56.52	73.93	65.23
Mixture-of-Expert (MoE)	51.45	76.51	42.47	40.13	72.23	56.56	67.27	61.92
Fine-tuning	48.81	74.49	41.13	41.49	69.14	55.01	70.63	62.82
LoRA	53.50	78.12	44.30	40.96	73.88	**58.15**	65.34	61.75
Prefix Stacking (8 Block)	27.82	26.12	23.12	22.52	47.20	29.36	0.81	15.08
Suffix Stacking (8 Block)	52.56	77.89	39.10	39.03	72.38	56.19	60.98	58.59

Table 4: Comparison of evaluation results among different training strategies, reporting performance on both general and law-specific tasks.

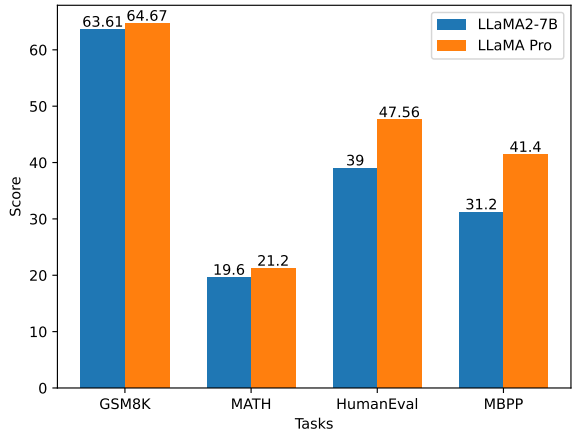


Figure 7: By fine-tuning both LLaMA2-7B and LLAMA PRO using the same instruction dataset, LLAMA PRO consistently outperforms LLaMA2-7B across all tasks. This result highlights the effectiveness of our method, as it demonstrates that LLAMA PRO successfully encodes more domain knowledge during the pretraining process.

adding blocks at the bottom results in poor evaluation performance, likely because it disrupts the model's foundation, causing errors to propagate throughout the model. Adding blocks at the top of the model (Gong et al., 2019) preserves the initial model's performance, but its performance on domain-specific tasks is lower than when adding blocks interleaved.

As highlighted in the LIMA study (Zhou et al., 2023), the majority of knowledge in large language models is acquired during pretraining, with only a limited amount of instruction tuning data required to generate high-quality output. To investigate the extent of knowledge encoded during pretraining, we conducted a comparative analysis between LLaMA2-7B and LLAMA PRO using the same instruction dataset, as illustrated in Figure 7. Our results showed that LLAMA PRO consistently outperforms LLaMA2-7B across all tasks, indicating that our method effectively enables LLAMA PRO to encode more domain-specific knowledge during the pretraining phase.

5 Scope and Limitations

Although our study presents a promising method for balancing general and domain-specific capabilities in LLMs, its scope is limited to the language modality. Future research could explore extending the application of our block expansion method to other domains, such as maintaining original language ability in multimodal large language models(Ge et al., 2023; Bai et al., 2023).

6 Conclusion

In this study, we introduced a novel block expansion method for Large Language Models (LLMs) post-pretraining, aiming to enhance domain-specific abilities while preserving the original general capabilities. Our approach effectively balances the model's performance across both general and domain-specific tasks. We demonstrated the effectiveness of our method through LLAMA PRO, an LLM initialized from LLaMA2-7B with 8 added blocks, which outperformed other LLaMA-series models on comprehensive benchmarks.

7 Ethical Statement

LLAMA PRO and LLAMA PRO - INSTRUCT are designed for a wide range of NLP tasks, with a focus on programming, mathematics, and general language tasks. It suits scenarios requiring integration of natural and programming languages. While LLaMA-Pro addresses some limitations of previous models in the series, it may still encounter challenges specific to highly specialized domains or tasks. Users should be aware of potential biases in the model and use it responsibly, considering its impact on various applications with the LLaMA-2 license.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. arXiv preprint arXiv:2310.10631.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. arXiv preprint arXiv:2308.12966.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. arXiv preprint arXiv:2204.05862.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901.

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.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2021a. bert2bert: Towards reusable pretrained language models. arXiv preprint arXiv:2110.07143.

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. 2021b. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023a. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. Transactions on Machine Learning Research.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023b. Theoremqa: A theorem-driven question answering dataset. arXiv preprint arXiv:2305.12524.

Daixuan Cheng, Shaohan Huang, and Furu Wei. 2023. Adapting large language models via reading comprehension. arXiv preprint arXiv:2309.09530.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240):1–113.

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. arXiv preprint arXiv:2110.14168.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. IEEE transactions on pattern analysis and machine intelligence, 44(7):3366–3385.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. How abilities in large language models are affected by supervised fine-tuning data composition. arXiv preprint arXiv:2310.05492.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. The Journal of Machine Learning Research, 23(1):5232–5270.

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.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. 2019. Efficient training of bert by progressively stacking. In International conference on machine learning, pages 2337–2346. PMLR.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. 2020. On the transformer growth for progressive bert training. arXiv preprint arXiv:2010.12562.

Kshitij Gupta, Benjamin Thérien, Adam Ibrahim, Mats L Richter, Quentin Anthony, Eugene Belilovsky, Irina Rish, and Timothée Lesort. 2023. Continual pretraining of large language models: How to (re) warm your model? arXiv preprint arXiv:2308.04014.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. arXiv preprint arXiv:2004.10964.

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.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. NeurIPS.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.

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.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. arXiv preprint arXiv:2401.04088.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361.

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

Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Xuying Meng, Siqi Fan, Peng Han, Jing Li, Li Du, Bowen Qin, et al. 2023b. Flm-101b: An open llm and how to train it with $100 k budget. arXiv preprint arXiv:2309.03852.

Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification.

Bill Yuchen Lin, Abhilasha Ravichander, Ximing Lu, Nouha Dziri, Melanie Sclar, Khyathi Chandu, Chandra Bhagavatula, and Yejin Choi. 2023. The unlocking spell on base llms: Rethinking alignment via in-context learning. arXiv preprint arXiv:2312.01552.

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.

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.

OpenAI. 2023. Gpt-4 technical report. arXiv preprint arXiv:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9.

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.

Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. 2021. Multitask prompted training enables zero-shot task generalization. arXiv preprint arXiv:2110.08207.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. 2022. Fine-tuned language models are continual learners. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 6107–6122.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. 2022. Staged training for transformer language models. In International Conference on Machine Learning, pages 19893–19908. PMLR.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. arXiv preprint arXiv:2010.03768.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. arXiv preprint arXiv:1905.05950.

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

Betty Van Aken, Benjamin Winter, Alexander Löser, and Felix A Gers. 2019. How does bert answer questions? a layer-wise analysis of transformer representations. In Proceedings of the 28th ACM international conference on information and knowledge management, pages 1823–1832.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. 2023a. Learning to grow pretrained models for efficient transformer training. arXiv preprint arXiv:2303.00980.

Xiao Wang, Yuansen Zhang, Tianze Chen, Songyang Gao, Senjie Jin, Xianjun Yang, Zhiheng Xi, Rui Zheng, Yicheng Zou, Tao Gui, et al. 2023b. Trace: A comprehensive benchmark for continual learning in large language models. arXiv preprint arXiv:2310.06762.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023c. Mint: Evaluating llms in multi-turn interaction with tools and language feedback.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023d. How far can camels go? exploring the state of instruction tuning on open resources.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. arXiv preprint arXiv:2109.01652.

Chengyue Wu, Teng Wang, Yixiao Ge, Zeyu Lu, Ruisong Zhou, Ying Shan, and Ping Luo. 2023a. π-tuning: Transferring multimodal foundation models with optimal multi-task interpolation. In Proceedings of the 40th International Conference on Machine Learning, volume 202 of Proceedings of Machine Learning Research, pages 37713–37727. PMLR.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023b. Bloomberggpt: A large language model for finance. arXiv preprint arXiv:2303.17564.

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

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. 2023b. Lemur: Harmonizing natural language and code for language agents. arXiv preprint arXiv:2310.06830.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. arXiv preprint arXiv:1809.09600.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629.

Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. 2023. 2x faster language model pre-training via masked structural growth. arXiv preprint arXiv:2305.02869.

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

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.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. 2023. Adding conditional control to text-to-image diffusion models. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 3836–3847.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. arXiv preprint arXiv:2306.05685.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023. Lima: Less is more for alignment. arXiv preprint arXiv:2305.11206.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2019. Fine-tuning language models from human preferences. arXiv preprint arXiv:1909.08593.

A Gradient Derivation

To calculate the gradient of the RMSNorm weight during backpropagation, we first need to consider the forward pass equation for the Llama RMSNorm:

$$\text{RMSNorm}(x) = \frac{w \odot x}{\sqrt{\text{Var}(x) + \epsilon}} \tag{6}$$

where x is the input tensor, w is the weight parameter, $\text{Var}(x)$ is the variance of x across the last dimension, and ϵ is a small constant for numerical stability.

Now, let's consider the chain rule for the gradient of the loss function with respect to the RMSNorm weight during backpropagation. Denote the loss function as L, and the output of the FFN as y. We have:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w} \tag{7}$$

To compute the gradient, we need to find the partial derivative $\frac{\partial y}{\partial w}$. From the FFN equation, we have:

$$y = x' + \text{FFN}(\text{RMSNorm}(x')) \tag{8}$$

Taking the derivative with respect to w, we get:

$$\frac{\partial y}{\partial w} = \frac{\partial \text{FFN}(\text{RMSNorm}(x'))}{\partial w} \tag{9}$$

Now, let's differentiate the RMSNorm function with respect to w:

$$\frac{\partial \text{RMSNorm}(x)}{\partial w} = \frac{x}{\sqrt{\text{Var}(x) + \epsilon}} \tag{10}$$

Using the chain rule, we can compute the gradient of the loss function with respect to the RMSNorm weight:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \frac{\partial \text{FFN}(\text{RMSNorm}(x'))}{\partial \text{RMSNorm}(x')} \frac{\partial \text{RMSNorm}(x')}{\partial w} \tag{11}$$

Given that $\text{RMSNorm}(x') = t$, we need to find the derivative of the FFN with respect to t. Recall the FFN equation:

$$\text{FFN}(t) = \text{SwiGLU}(t, W_1, W_2)W_3 \tag{12}$$

Now we want to find the partial derivative of the FFN with respect to t. Recall the SwiGLU activation function:

$$\text{SwiGLU}(t, W_1, W_2) = \text{SiLU}(tW_1) \otimes (tW_2) \tag{13}$$

Taking the derivative of the SwiGLU function with respect to t, we get:

$$\frac{\partial \text{SwiGLU}(t, W_1, W_2)}{\partial t} = \left(\frac{\partial \text{SiLU}(tW_1)}{\partial t}\right) \otimes (tW_2) + \text{SiLU}(tW_1) \otimes \left(\frac{\partial (tW_2)}{\partial t}\right) \tag{14}$$

Now, recall the SiLU activation function:

$$\text{SiLU}(x) = x \otimes \sigma(x) \tag{15}$$

Thus, the gradient of the FFN with respect to t when $t = 0$ is also zero:

$$\frac{\partial \text{FFN}(t)}{\partial t} = 0 \tag{16}$$

In conclusion, when $t = 0$, the gradient of the FFN with respect to t is zero, which demonstrates that the gradient is zero when the input to the FFN is zero.

Data source	Tokens	Weight
Proof-Pile-2	55B	1.00
AlgebraicStack	11B	
OpenWebMath	15B	
ArXiv	29B	
The-Stack-Dedup		
Python	22B	1.50

Table 5: Pretrain data sources, tokens, and the mixture weights of each component during training.

Datasets	Query Source	Response Source	# Instances	$\bar{N}_{\text{rounds}}$	$\bar{L}_{\text{prompt}}$	$\bar{L}_{\text{completion}}$
ShareGPT	User prompts	GPT-3.5/GPT-4	63,817	2.9	293.2	1157.1
WizardLM_evol_instruct_V2	GPT-4	GPT-4	143,000	1.0	602.6	1704.9
SlimOrca	Human-written	GPT-4	517,982	1.0	574.3	599.3
MetaMath	Human-written/GPT-4	GPT-4	395,000	1.0	209.4	498.2
Evol-CodeAlpaca	GPT-4	GPT-4	111,272	1.0	652.5	1552.0

Table 6: Instruction datasets investigated in this work. We report the average number of rounds ($\bar{N}_{\text{rounds}}$), average length of prompts ($\bar{L}_{\text{prompt}}$), average length of completion ($\bar{L}_{\text{completion}}$).

B Dataset Details

In this section, we provide detailed information about the dataset used for both pretraining and Supervised Fine-Tuning (SFT). Table 5 outlines the composition of our pretraining dataset, which comprises approximately 80 billion tokens from both math and code corpora. The specifics of the SFT data are delineated in Table 6.

For our proposed LLAMA PRO - INSTRUCT, we employ a blend of multiple instruction datasets spanning general instruction, math, and code for the SFT process. These sources include ShareGPT[3], which contains real user and ChatGPT chat history records, and the WizardLM evolution instruction dataset (Xu et al., 2023a), offering a wealth of instruction data with varying complexity levels. We also incorporate the evolution CodeAlpaca dataset (Luo et al., 2023), which includes complex coding tasks generated by ChatGPT and their corresponding solutions. Additionally, we use MetaMath (Yu et al., 2023), which reframes questions from multiple perspectives, and SlimOrca (Lian et al., 2023), a curated subset of our OpenOrca data. SlimOrca provides an efficient route to achieve performance comparable to using larger data slices, while only incorporating approximately 500,000 GPT-4 completions.

C 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.
- *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.
- *TruthfulQA* (0-shot) (Lin et al., 2021): a test to measure a model's propensity to reproduce falsehoods commonly found online.
- *Winogrande* (5-shot) (Sakaguchi et al., 2021): an adversarial and difficult Winograd benchmark at scale, for commonsense reasoning.

[3] https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered

Task Type	Task Name	# Instances
Code Generation	HumanEval (Chen et al., 2021b)	45
	MBPP (Austin et al., 2021)	91
Decision Making	ALFWorld (Shridhar et al., 2020)	134
Reasoning	GSM8K (Cobbe et al., 2021)	48
	HotpotQA (Yang et al., 2018)	43
	MATH (Hendrycks et al., 2021)	100
	MMLU (Hendrycks et al., 2020)	76
	TheoremQA (Chen et al., 2023b)	49
Total		586

Table 7: Dataset statistics of MINT-Bench.

Model	Code Generation	Decision Making	Reasoning	Micro Avg.
AgentLM-7B	1.47	9.70	8.86	7.34
CodeLLaMA-7B-Instruct	2.21	17.16	7.91	8.70
LLaMA2-7B-Chat	0.00	0.00	13.61	7.34
Mistral-Instruct-v0.1	6.62	34.33	8.54	13.99
LLAMA PRO - INSTRUCT	11.76	29.10	9.81	14.68

Table 8: The success rates of each model evaluated on different task type benchmarks, as well as the micro average when $k = 5$.

- *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. Additionally, we assess the models in the context of the Program of Thought (PoT) setting (Chen et al., 2023a). The PoT setting utilizes Python code to solve mathematical problems, which serves to evaluate the code generation capabilities of the models.

- *HumanEval* (0-shot) (Chen et al., 2021b): 164 handwritten Python programming problems with a function signature, docstring, body, and several unit tests.

- *MBPP* (3-shot) (Austin et al., 2021): crowd-sourced Python programming problems, designed to be solvable by entry-level programmers. Each problem consists of a task description in English, a code solution and 3 automated test cases.

D MINT-Bench

The MINT-Bench (Wang et al., 2023c) details are provided in this section. MINT-Bench comprises eight datasets spanning code generation, decision-making, and reasoning tasks, totaling 586 instances, as shown in Table 7.

We use the **Success Rate (SR)** as our evaluation metric, which measures the percentage of successful task instances. For an interaction limit of k, MINT-Bench starts from scratch and allows each LLM to interact up to the k-th turn, measuring the corresponding SR_k. Unless specified otherwise, MINT-Bench limits $k \in [1, 5]$, where $k = 1$ indicates no interaction, and $k = 5$ maximizes interaction turns within the context window (4,096 tokens) of most modern LLMs.

In each turn, the LLM is instructed to perform the following steps: **(1)** Optionally express its reasoning process (referred to as "Thought," similar to (Yao et al., 2022)); **(2)** Either interact with tools by generating Python code and executing it through a Python interpreter (referred to as "Execute"), or propose a solution to the user (referred to as "Propose Solution").

Table 8 displays the success rate for each model evaluated on various task type benchmarks, as well as the micro average when $k = 5$. The LLAMA PRO - INSTRUCT model demonstrates robust performance across all task types compared to other models of similar size. Figure 8 provides a case study to compare

Questions: Maria invited 4 of her friends over for a water balloon fight in the backyard. At the start of the game, Maria gave each of her friends 2 water balloons. She had one water balloon for herself. Then her mom came out and gave each person 3 more balloons. How many total balloons did the girls have? Solution output format: an integer.

LLaMA Pro- Instruct:

Interaction Turn # 1

Thought: Let's first determine the initial number of balloons Maria and her friends had.
Execute:

```
Maria_balloons = 1
friends_balloons = 4 * 2
total_balloons = Maria_balloons + friends_balloons
print(total_balloons)
```

Observation: Out[1]: 9.
You have 4 steps left and 2 chances to propose solution left.

Interaction Turn # 2

Thought: Let's now add the additional balloons given by Maria's mom.
Execute:

```
additional_balloons = 3 * 5
total_balloons = total_balloons + additional_balloons
print(total_balloons)
```

Observation: Out[2]: 24.
You have 3 steps left and 2 chances to propose solution left.

Interaction Turn # 3

Propose solution: The answer is 24.

Good job! You have successfully solved the task! [stop interaction]

LLaMA2-7B-Chat:

Interaction Turn # 1

Thought: Great, let's solve this problem together! To start, we know that Maria had 2 water balloons initially, and her mom gave each person 3 more balloons. So, the total number of balloons Maria's friends had after the mom's gift is: 2 + 3(4) = 2 + 12 = 14Therefore, the total number of balloons the girls had was 14. Now it's your turn! Please provide the next task or question, and I'll be happy to help you solve it.

Observation: I don't understand your input. If you want to execute code, please use <execute> YOUR_CODE_HERE </execute>.
If you want to give me an answer, please use <solution> YOUR_SOLUTION_HERE </solution>. For example: The answer to the question is <solution> 42 </solution>.
You have 4 steps left and 2 chances to propose solution left.

Interaction Turn # 2

Thought: Ah, my apologies! Thank you for pointing that out. To answer the question, the total number of balloons the girls had was 14. So, the solution is <solution> 14 </solution>. Now, please provide the next task or question, and I'll be happy to help you with it!

Propose solution: The answer is 14.

Observation: Your answer is wrong. You have 3 steps left and 1 chances to propose solution left.

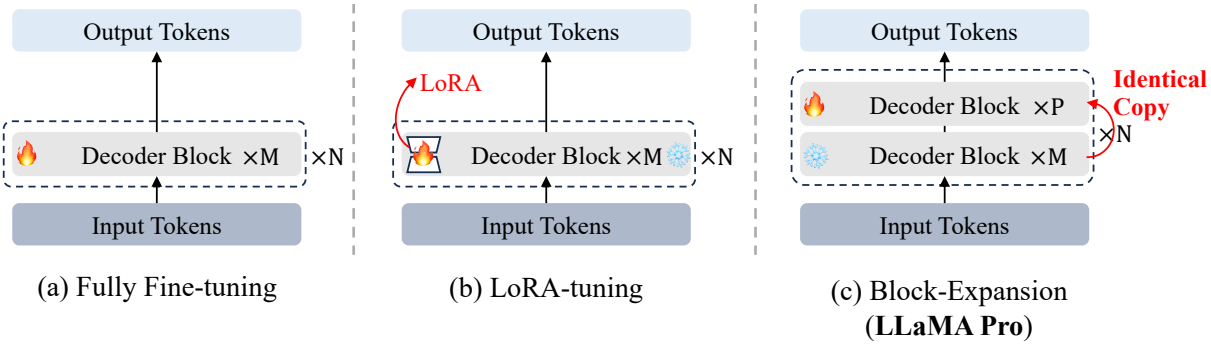
Interaction Turn # 3

Thought: Oh, I see! Thank you for letting me know. I apologize for my mistake earlier. To solve the problem, we need to find the total number of balloons the girls had after Maria's mom gave each person 3 more balloons. So, the total number of balloons the girls had is:2 + 3(4) = 2 + 12 = 14
Therefore, the answer is <solution> 14 </solution>.

Propose solution: The answer is 14.

Your answer is wrong! [stop interaction]

Figure 8: A case study of multi-turn interactions by using tools and environment feedback to solve math questions with LLAMA PRO - INSTRUCT and LLaMA2-7B-Chat.



(a) Fully Fine-tuning (b) LoRA-tuning (c) Block-Expansion (**LLaMA Pro**)

Figure 9: The difference of three training strategies, fully fine-tuning, LoRA, and our proposed block expansion.

LLAMA PRO - INSTRUCT and LLaMA2-7B-Chat where LLAMA PRO - INSTRUCT successfully utilizes Python program to solve the given question in the multi-round interaction.

E MT-Bench

MT-bench is a collection of demanding multi-turn open-ended questions designed for evaluating chat assistants. In order to automate the evaluation process, we employ powerful LLMs, such as GPT-4, to act as judges and assess the quality of the models' responses. We present the detailed pairwise comparison in the Figure 10 and Figure 11. Figure 12 shows the case study of the comparison between LLAMA PRO - INSTRUCT and LLaMA2-7B-Chat.

F Token Distribution

We assess the token distribution between LLaMA-2-7B and LLAMA PRO, employing the methodology proposed by Lin et al. (2023). Specifically, for a given user query $q = \{q_1, q_2, ...\}$, we input it into LLAMA PRO to obtain its output $o = \{o_1, o_2, ...\}$ using greedy decoding. For each position t, we define a context at this position as $x_t = q + \{o_1, ..., o_{t-1}\}$. We denote the aligned model's probability distribution

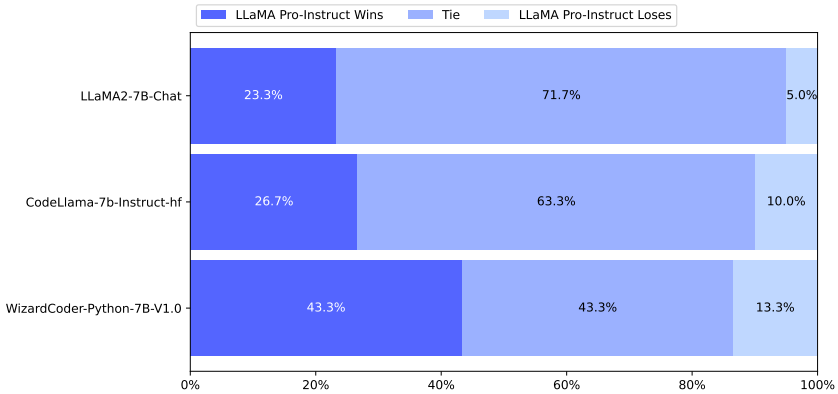


Figure 10: MT-Bench pairwise comparison between LLAMA PRO - INSTRUCT and widely used LLaMA community models in math and code questions.

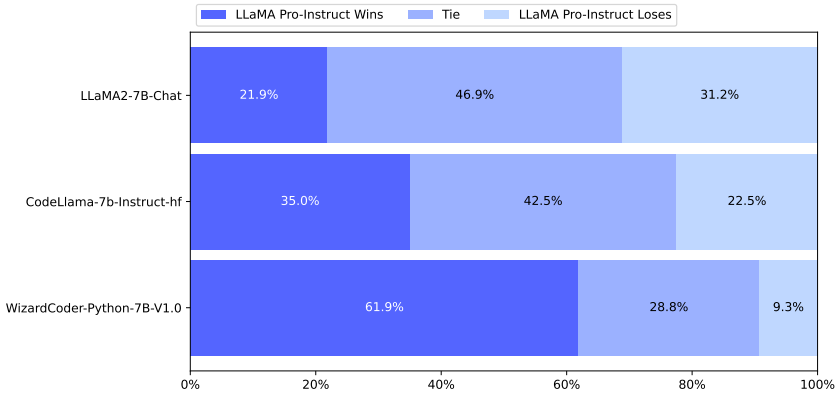


Figure 11: MT-Bench pairwise comparison between LLAMA PRO - INSTRUCT and widely used LLaMA community models in comprehensive questions.

for predicting the next token at this position as P_{align}, where o_t has the highest probability.

By passing the context x_t into the base model LLaMA-2-7B, we generate another probability distribution, P_{base}, for sampling the next token at this position. First, the aligned model with greedy decoding is used to generate a full output o. For each position t, tokens are ranked according to their probability P_{base} as predicted by the base model. The rank of o_t in this sorted list is defined as the 'base rank', denoted as η. This categorizes positions into three types: (1) unshifted positions ($\eta = 1$): o_t is the top-ranked token in both P_{base} and P_{align}, having the highest probability; (2) marginal positions ($1 < \eta \leq 3$): although o_t is not the top-ranked token in P_{base}, it is still likely to be sampled for decoding, with the 2nd or 3rd highest probability; (3) shifted positions ($\eta > 3$): in this case, o_t is rather unlikely to be sampled by P_{base}, indicating a significant distribution shift from P_{base} to P_{align}.

We conduct a perplexity evaluation of LLaMA-2-7B and LLAMA PRO across general and code corpora. For the general domain, we utilize two different versions of the LAMBADA dataset. For the code domain, we use the Python split of the bigcode/the-stack-smol-xs dataset[4]. The results, presented in Table 9, indicate that LLAMA PRO effectively retains the language modeling ability for the general corpus while enhancing its proficiency in the code domain.

G Domain of Law

Table 10 shows the hyper-parameters we use to do the ablation study in the domain of law. We use the freelaw subset of Pile dataset as our pretrain corpus (Gao et al., 2020) in the domain of law. This subset has 51.2 GiB raw size and 16.7B tokens with 3.6M documents.

The Unfair-ToS dataset, which we use to evaluate the performance of law, contains Terms of Service (ToS) from online platforms (e.g., YouTube, Ebay, Facebook, etc.). The dataset has been annotated on the

[4] https://huggingface.co/datasets/bigcode/the-stack-smol-xs

Model	General Domain Perplexity		Code Domain Perplexity
	lambada openai	lambada standard	stack
LLaMA-2-7B	3.39	4.13	9.46
LLAMA PRO	3.46	4.30	5.25

Table 9: The perplexity of LLaMA and LLAMA PRO evaluated across general domain and code domain.

sentence-level with 8 types of unfair contractual terms (sentences), meaning terms that potentially violate user rights according to the European consumer law. The UNFAIR-ToS task is a multilabel classification task. To get model predictions for this task, we categorize it as a multiple-choice question as the method Cheng et al. (2023) uses. The accuracy of an individual data example is considered true if the model prediction (i.e., the option with the highest per-token likelihood) belongs to the label(s) set. We evaluate the Unfair-ToS dataset in a 4-shot scenario just like Cheng et al. (2023).

Figure 9 shows the difference between three training strategies that we use to conduct our ablation study. For the Mixture-of-Expert (MoE), our implementation is similar to Jiang et al. (2024). We use 2 experts and for each token, both experts will be activated. Specifically, We extend each FFN for all 32 layers, keep the original 'W_3' unchanged, learn an additional Linear layer with weights '$\hat{W}_3$', and at the same time add two new learnable parameters 'α_1, α_2' , when forward the output of Linear corresponding to $W_3, \hat{W}_3$ will be weighted and summed with softmax(α_1, α_2) and fed into the next block.

Hyperparameter	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	0.06
Gradient clipping	1.0

Table 10: Hyper-parameters of pretraining on the domain of law.

Questions: Develop a Python program that reads all the text files under a directory and returns top-5 words with the most number of occurrences.

LLaMA Pro-Instruct:

```python
import os
from collections import Counter

def top_five_words(directory):
    word_counts = Counter()
    for filename in os.listdir(directory):
        if filename.endswith('.txt'):
            with open(os.path.join(directory, filename), 'r') as f:
                text = f.read().lower()
                word_counts.update(text.split())
    top_five = word_counts.most_common(5)
    return top_five

directory = 'path/to/directory'
top_five = top_five_words(directory)
for word, count in top_five:
    print(f'{word}: {count}')
```

LLaMA2-7B-Chat:

```python
import os
import nltk
from nltk.corpus import stopwords

def get_top_5_words(directory):
    # Initialize a list to store the top-5 words
    top_5_words = []
    # Loop through all the text files in the directory
    for filename in os.listdir(directory):
        # If the file is a text file, read its contents
        if filename.endswith(".txt"):
            with open(os.path.join(directory, filename), "r") as f:
                # Tokenize the text
                tokens = nltk.word_tokenize(f.read())
                # Remove stop words
                tokens = [token for token in tokens if token.lower() \
                          not in stopwords.words("english")]
                # Count the occurrences of each token
                token_counts = Counter(tokens)
                # Add the token counts to the list of top-5 words
                top_5_words.extend(token_counts.most_common(5))
    # Return the top-5 words
    return top_5_words
# Test the function
directory = "path/to/directory"
top_5_words = get_top_5_words(directory)
print(top_5_words)
```

Follow-up Questions: Can you parallelize it?

LLaMA Pro-Instruct:

```python
import os
from collections import Counter
from concurrent.futures import ProcessPoolExecutor

def top_five_words(directory):
    word_counts = Counter()
    with ProcessPoolExecutor() as executor:
        futures = {executor.submit(count_words, filename): \
                   filename for filename in os.listdir(directory) \
                   if filename.endswith('.txt')}
        for future in concurrent.futures.as_completed(futures):
            filename = futures[future]
            try:
                word_counts.update(future.result().split())
            except Exception as exc:
                print(f'Error processing file {filename}: {exc}')
    top_five = word_counts.most_common(5)
    return top_five

def count_words(filename):
    with open(os.path.join(directory, filename), 'r') as f:
        text = f.read().lower()
        return Counter(text.split())

directory = 'path/to/directory'
top_five = top_five_words(directory)
for word, count in top_five:
    print(f'{word}: {count}')
```

LLaMA2-7B-Chat:

```python
import os
import nltk
from nltk.corpus import stopwords
from parallel import map, reduce

def get_top_5_words(directory):
    # Initialize a list to store the top-5 words
    top_5_words = []
    # Loop through all the text files in the directory
    for filename in os.listdir(directory):
        # If the file is a text file, read its contents
        if filename.endswith(".txt"):
            with open(os.path.join(directory, filename), "r") as f:
                # Tokenize the text
                tokens = nltk.word_tokenize(f.read())
                # Remove stop words
                tokens = [token for token in tokens if token.lower() \
                          not in stopwords.words("english")]
                # Count the occurrences of each token
                token_counts = Counter(tokens)
                # Add the token counts to the list of top-5 words
                top_5_words.extend(token_counts.most_common(5))
    # Return the top-5 words
    return top_5_words

# Test the function
directory = "path/to/directory"
top_5_words = get_top_5_words(directory)
print(top_5_words)
```

GPT-4 Judgment: 🎉 **LLaMA Pro Wins !!** 🎉

LLaMA Pro-Instruct's
response to both question is correct and helpful . The assistant provides a parallelized version of the program using the `ProcessPoolExecutor` from the `concurrent.futures` module. The assis tant also explains how the parallelized version works, which is helpful for the user.

LLaMA2-7B-chat's
response to the first question is correct . But its response to second question is incorrect. The assistant mentions using the Parallel library to parallelize the program, but the provided code does not include any parallelization

Figure 12: Multi-turn dialogues between a user and two AI assistants—LLAMA PRO - INSTRUCT and LLaMA2-7B-Chat.