Parameter-Efficient Sparsity Crafting from Dense to Mixture-of-Experts for Instruction Tuning on General Tasks

Haoyuan Wu[♠], **Haisheng Zheng**[♡], **Zhuolun He**[♠,♣], **Bei Yu**[♠]

[♠]The Chinese University of Hong Kong, Hong Kong SAR
[♡]Shanghai Artificial Intelligent Laboratory, China
[♣]ChatEDA Tech, China
`{hywu24,byu}@cse.cuhk.edu.hk`

Abstract

Large language models (LLMs) have demonstrated considerable proficiency in general natural language processing (NLP) tasks. Instruction tuning, a successful paradigm, enhances the ability of LLMs to follow natural language instructions and exhibit robust generalization across general tasks. However, these models often encounter performance limitations across multiple tasks due to constrained model capacity. Expanding this capacity during the instruction tuning phase poses significant challenges. To address this issue, we introduce parameter-efficient sparsity crafting (PESC), which crafts dense models into sparse models using the mixture-of-experts (MoE) architecture. PESC integrates adapters into the MoE layers of sparse models, differentiating experts without altering the individual weights within these layers. This method significantly reduces computational costs and GPU memory requirements, facilitating model capacity expansion through a minimal parameter increase when guaranteeing the quality of approximation in function space compared to original sparse upcycling. Our empirical evaluation demonstrates the effectiveness of the PESC method. Using PESC during instruction tuning, our best sparse model outperforms other sparse and dense models and exhibits superior general capabilities compared to GPT-3.5. Our code is available at `https://github.com/wuhy68/Parameter-Efficient-MoE`.

1 Introduction

Recent advancements in NLP have been significantly propelled by the advent of LLMs such as GPT (Brown et al., 2020; OpenAI, 2023), Llama (Touvron et al., 2023a,b), Mistral (Mistral AI, 2023; Jiang et al., 2024), etc. The increasing scale of LLMs has established them as the experts for NLP tasks due to their exceptional ability to identify complex linguistic patterns (Wei et al., 2022).

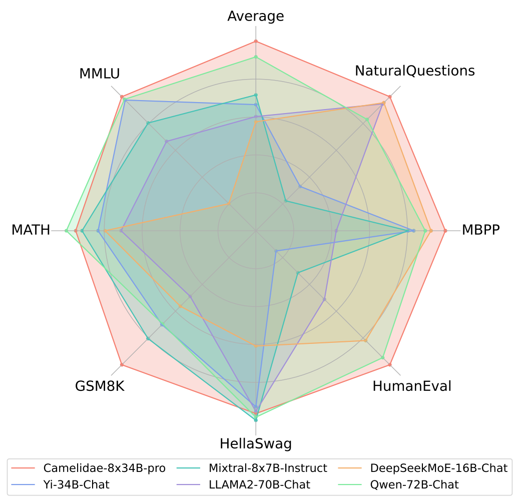

Figure 1: Camelidae-8×34B-pro achieves excellent performance across general tasks.

A prominent method for training LLMs is instruction tuning (Wei et al., 2021). This approach utilizes large-scale, well-formatted instruction data, enabling LLMs to refine their pre-trained representations to comply with human instructions (Taori et al., 2023; Xu et al., 2024; Dettmers et al., 2024; Mukherjee et al., 2023). Such instruction-tuned LLMs exhibit remarkable generalization capabilities in NLP tasks (Longpre et al., 2023). This generalization requires training on a broad range of instruction-following tasks from multiple domains such as math, code, biology, etc (Chung et al., 2022; Sanh et al., 2021). However, the inherent complexity of these tasks can hinder model fine-tuning (Zhang and Yang, 2021). Specifically, models of certain sizes may struggle to optimize losses from conflicting tasks, resulting in subpar performance for general tasks.

The scaling law (Chung et al., 2022) suggests that increasing the model's scale is crucial for better performance. Expanding the model's capacity can also improve instruction tuning effectiveness for general tasks (Kaplan et al., 2020). Nonetheless, most LLMs are pre-trained dense models designed based on transformer architecture, which limits scalability during instruction tuning. Komatsuzaki et al. (2023) presented a method for upcycling dense models into sparse activated MoE models, which boast greater capacity (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Puigcerver et al., 2023). Notably, Shen et al. (2023) suggested that MoE models respond more effectively to instruction tuning compared to dense models. Consequently, converting dense models into MoE models during instruction tuning has the potential to achieve great performance on general tasks. This conversion involves initializing each expert in the MoE models as a copy of the feedforward neural network (FFN) layers (Chen et al., 2015; Rae et al., 2021). Given the parameter scale of current LLMs, training such giant models requires updating the weights of experts in the MoE layer, which is constrained by GPU memory resources and computational costs.

To mitigate these challenges, we introduce parameter-efficient sparsity crafting (PESC), an approach that effectively expands model capacity while synergizing with parameter-efficient fine-tuning (PEFT) techniques (Houlsby et al., 2019; Dettmers et al., 2024). PESC involves inserting adapters (Houlsby et al., 2019) into the MoE layers of sparse models, allowing differentiation between experts without altering each expert's weights in the MoE layers when guaranteeing the quality of the approximation in function space compared to original sparse upcycling (Komatsuzaki et al., 2023). Considering that the more sophisticated construction can improve the approximation (Ding et al., 2022), we also apply the QLoRA (Dettmers et al., 2024) technique to update other weights in the sparse models. As shown in Figure 1, our Camelidae-8×34B-pro, instruction fine-tuned utilizing PESC, achieved the best performance among various open-source sparse models and dense models. Our contributions are described as follows:

- We propose an approach, parameter-efficient sparsity crafting (PESC), for the extension of the model capacity efficiently.
- We implement the PESC method for instruction tuning across general tasks, achieving significant performance improvements on various benchmarks.
- We develop Camelidae models, sparse models trained with the PESC method, achieving the best performance across open-source sparse models and demonstrating superior general capabilities compared to GPT-3.5.

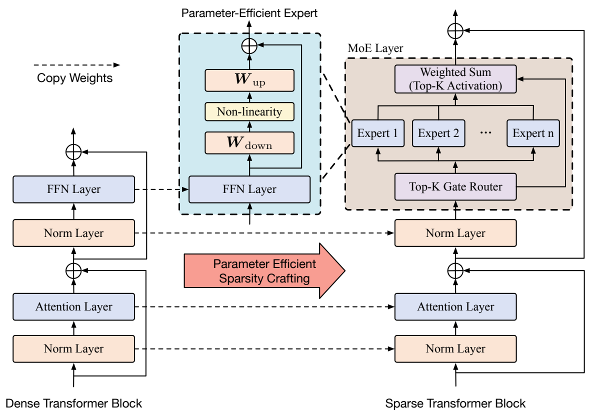

Figure 2: Overview of the parameter-efficient sparsity crafting with parameter-efficient experts.

2 Methodology

2.1 Preliminaries

Adapters. Houlsby et al. (2019) proposed the integration of adapters into pre-trained transformer-based models to enhance parameter efficiency. This approach involves tuning only the parameters added by the adapters. An adapter consists of two matrices, $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d_1 \times d_2}$ and $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{d_2 \times d_1}$, coupled with a non-linear function $\sigma(\cdot)$. Here, d_1 and d_2 denote the feature dimensions in the pre-trained models and the adapter's hidden dimension, respectively, with $d_2 < d_1$ typically. Given a feature $\boldsymbol{U} \in \mathbb{R}^{N \times d_1}$ in the pre-trained model, the output of the Adapter module is expressed as:

$$\boldsymbol{U}' = \sigma(\boldsymbol{U}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}} + \boldsymbol{U}. \tag{1}$$

Mixture-of-Experts. As depicted in Figure 2, an MoE layer comprises n experts, $\{E_i\}_{i=1}^{n}$, and a router R. The output $\boldsymbol{y}$ for an input $\boldsymbol{x}$ in the MoE layer is computed as:

$$\boldsymbol{y} = \sum_{i=1}^{n} R(\boldsymbol{x})_i E_i(\boldsymbol{x}), \tag{2}$$

where $R(\boldsymbol{x})_i$ represents the output of the gating network for the i-th expert, and $E_i(\boldsymbol{x})$ is the output of the i-th expert.

Sparsity Crafting. Building on the concept of sparsity upcycling (Komatsuzaki et al., 2023), sparsity crafting leverages the weights of dense models. As depicted in Figure 2, sparsity crafting involves a transformative process: substituting the

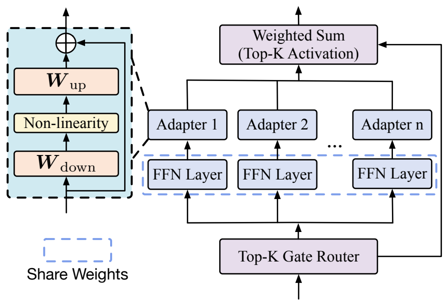

Figure 3: Detailed design of the MoE layer for PESC utilizing parameter-efficient experts. All the FFN layers share the same weights.

FFN layer F within each block of the dense transformer model with an MoE layer. This replacement gives rise to an innovatively sparse transformer block. During the initialization phase of sparsity crafting, each expert E_i within the MoE layer is initialized with the FFN layer F. To ensure structural coherence, other components, such as the normalization and attention layers, are replicated directly from the dense transformer block.

For clarity, let us define $\mathcal{F}_i(\theta_i)$ as the objective function for the i-th expert in the MoE layer, where θ_i represents the parameters for E_i. θ_i is initialized from θ_o, which are the parameters of the FFN layer F from the original dense model. The essence of the sparsity crafting training regimen lies in the optimization of $\mathcal{F}_i(\theta_i)$. The goal is to derive θ_i^+, the optimized parameters for each expert. This is formally expressed as:

$$\theta_i^+ = \arg\min_{\theta_i} \mathcal{F}_i(\theta_i). \quad (3)$$

After the instruction tuning process utilizing the sparsity crafting technique, the optimized parameter sets $\{\theta_i^+\}_{i=1}^n$ are obtained for experts $\{E_i\}_{i=1}^n$ in the MoE layer.

2.2 Parameter-Efficient Sparsity Crafting

As shown in Equation (3), traditional sparsity crafting necessitates optimizing the parameters $\{\theta_i\}_{i=1}^n$ for each expert E_i in the MoE layer, leading to significant resource consumption, including training time and memory costs due to the extensive parameters of FFN layers in LLMs. Consequently, as illustrated in Figure 2, we introduce PESC, an approach that addresses the high training time and memory costs associated with sparsity crafting in LLMs. Specifically, PESC, leveraging the parameter-efficient fine-tuning (PEFT) paradigm, focuses on tuning a smaller subset of parameters to achieve efficiency.

The core of PESC lies in its objective function, $\tilde{\mathcal{F}}_i(\theta_i, \omega_i)$, where ω_i represents the select parameters for tuning. Notably, the parameters of ω_i is significantly less than θ_i, as indicated by $|\omega_i| \ll |\theta_i|$, where $|\cdot|$ indicates the number of parameters involved. Each expert E_i begins the process with the initial state (θ_o, ω_o), where ω_o is initialized to zero to facilitate identity mapping, resulting in $\tilde{\mathcal{F}}_i(\theta_o, \omega_o) = \mathcal{F}_i(\theta_o)$. The training procedure for PESC is thus the optimization of $\tilde{\mathcal{F}}_i(\theta_o, \omega_i)$, leading to a solution ω_i^+ defined as:

$$\omega_i^+ = \arg\min_{\omega_i} \tilde{\mathcal{F}}_i(\theta_o, \omega_i). \quad (4)$$

Considering that $|\omega_i| \ll |\theta_i|$, we have

$$\sum_{i=1}^{n} |\omega_i^+| + |\theta_o| = n \times |\omega_o| + |\theta_o| \ll n \times |\theta_o| = \sum_{i=1}^{n} |\theta_i^+|. \quad (5)$$

Consequently, this solution set $\{\omega_i^+\}_{i=1}^n$ is more efficient than the original sparsity crafting parameters $\{\theta_i^+\}_{i=1}^n$ for the set $\{E_i\}_{i=1}^n$.

To ensure the effectiveness of PESC compared to traditional sparsity crafting, it is vital to maintain a small approximation error, as defined by:

$$|\tilde{\mathcal{F}}_i(\theta_i^+, \omega_o) - \tilde{\mathcal{F}}_i(\theta_o, \omega_i^+)| < \xi, \quad (6)$$

where ξ is the approximation error. This can be achieved by designing an approximate function $\tilde{\mathcal{F}}_i(\theta_o, \omega_i^+)$ that closely matches $\tilde{\mathcal{F}}_i(\theta_i^+, \omega_o)$ (Houlsby et al., 2019; Ding et al., 2022). Considering that the trajectory of θ_i optimization approximately follows a manifold, which can be projected into a lower-dimensional space such as adapter in Equation (1). The approximation error is contingent on the representational capacity of the inserted adapters. Given the universal approximation property of MLP layers with general activation functions, the Adapter module is a universal approximator (Funahashi, 1989; Leshno et al., 1993; Kidger and Lyons, 2020). As a result, utilizing the adapters as ω_i can effectively ensure the quality of the approximation of $\tilde{\mathcal{F}}_i(\theta_i^+, \omega_o)$.

2.3 Model Design

Parameter-Efficient Experts. According to the analysis in Section 2.2, adapters can guarantee a good lower bound ξ in Equation (6). Consequently, we can introduce parameter-efficient MoE layers by integrating adapters, thereby achieving sparsity in a more parameter-efficient manner.

In the training of sparse transformer blocks, gradients are back-propagated to each expert, necessitating parameter updates. For a collection of n experts, original sparsity crafting demands a computational cost n times that of a single FFN layer. As depicted in Figure 3, our PESC utilizes adapters to circumvent redundant updates of the expert weights θ_i. Specifically, we update the ω_i of n inserted adapters to differentiate between experts without altering each expert's original weights θ_o replicated from the original FFN layer. Thus, for a given input $\boldsymbol{x}$, Equation (2) can be reformulated as:

$$\boldsymbol{y} = \sum_{i=0}^{n} R(\boldsymbol{x})_i A_i(E(\boldsymbol{x})), \tag{7}$$

where $A_i(\boldsymbol{x})$ construct the parameter-efficient expert as follows:

$$A_i(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}_{i\text{down}})\boldsymbol{W}_{i\text{up}} + \boldsymbol{x}. \tag{8}$$

Considering that the more sophisticated construction can improve the approximation, we can also update the shared weights θ_o of $\{E_i\}_{i=1}^{n}$. As illustrated in Equation (7), this approach allows for efficient scaling of the model capacity by introducing a minimal number of parameters across n inserted adapters.

Top-K Gate Router. Within the sparse transformer block, the MoE layer encompasses a specified number of experts. A router, employing a softmax activation function, models a probability distribution over these experts, reflecting each expert's capability to process incoming tokens. The router's weights, denoted as $\boldsymbol{W}_r$, which are integrated into the sparse transformer block, are initially randomly initialized. As depicted in Figure 3, we utilize the top-k gate router within the sparse transformer block (Lepikhin et al., 2020; Du et al., 2022). This router activates the most suitable two experts out of n experts $\{E_i\}_{i=1}^{n}$ for each token $\boldsymbol{x}$ in an input sequence. After receiving the input token $\boldsymbol{x}$, the router produces router logits $R(\boldsymbol{x}) = \boldsymbol{W}_r \cdot \boldsymbol{x}$. Before being normalized via a softmax distribution over the available n experts, we perform the KeepTopK function. The KeepTopK function is applied to retain only the top-k values of the router logits, assigning $-\infty$ to the rest, effectively zeroing them post-softmax normalization. Thus, given a token $\boldsymbol{x}$, the router's output logit is represented as:

$$R(\boldsymbol{x}) = \text{Softmax}(\text{KeepTopK}(\boldsymbol{W}_r \cdot \boldsymbol{x})). \tag{9}$$

The gate value of each expert E_i for the input token $\boldsymbol{x}$ is $R(\boldsymbol{x})_i$. Despite an increase in parameters, the experts of the MoE layer are activated sparsely, implying that only a limited subset of experts is used per input token. This approach enhances the capacity of the model while maintaining computational efficiency. The top-k gate router selects the best two experts for each token during inference. In an MoE layer with n experts, this enables up to $\binom{n}{k}$ different combinations of experts, as opposed to a single combination in the traditional transformer architecture, providing enhanced computational adaptability.

Experts Loading Balance. The top-k gate router, through its gating mechanism, tends to disproportionately favor a few experts, leading to an imbalance where these experts are more frequently trained and consequently chosen by the router. To counter this imbalance and promote uniform expert utilization, an auxiliary loss as suggested by Fedus et al. (2022) is integrated during training for each sparse transformer block. With n experts and a batch B containing T tokens, this auxiliary loss $\mathcal{L}$ for experts loading balance is calculated as the scaled dot-product of vectors $\boldsymbol{f}$ and $\boldsymbol{p}$,

$$\mathcal{L} = \alpha \cdot n \cdot \sum_{i=1}^{n} \boldsymbol{f}_i \cdot \boldsymbol{p}_i, \tag{10}$$

where f_i denotes the fraction of tokens dispatched to expert i and p_i represents the fraction of router probability allocated to expert i. α is a multiplicative coefficient for the auxiliary losses. We utilize an $\alpha = 10^{-2}$ which was sufficiently large to ensure load balancing while small enough to not overwhelm the primary cross-entropy objective. As the ideal scenario entails uniform routing across the n experts, both vectors should ideally have values of $\frac{1}{n}$. The auxiliary loss of Equation (10) fosters this uniform distribution, achieving its minimum under such conditions.

3 Experiments

3.1 Settings

Training Data. To demonstrate the learning ability of the sparse model with MoE layers, we simultaneously trained the model on a diverse set of skills, encompassing coding, mathematical, and other general abilities from various subjects. This training involved integrating three distinct datasets from varied domains during the instruction tuning phase: SlimOrca (Lian et al., 2023; Mukherjee et al., 2023;

Longpre et al., 2023), Magicoder (Wei et al., 2023), and MetaMathQA (Yu et al., 2023) datasets. After filtration and sampling, we can get two instruction datasets including IDAE-500K and IDAE-720K finally. We provide more details of IDAE datasets in Appendix A.

Evaluation Benchmarks. Our evaluation compares the performance of dense and sparse models on academic benchmarks. The dense models include Llama2 (Touvron et al., 2023b), Vicuna (Zheng et al., 2023), Yi (01 AI, 2023), SUSChat (SUSTech IDEA, 2023), Qwen (Bai et al., 2023), GPT3.5 (Brown et al., 2020), and our Camel models, while the sparse models encompass Mixtral (Jiang et al., 2024), DeepSeekMoE (Dai et al., 2024), and our Camelidae models. Evaluations are conducted using OpenCompass (OpenCompass, 2023), LM-Eval-Harness (Gao et al., 2023), and our internal evaluation libraries, summarizing performances across well-known benchmarks. These benchmarks are illustrated as follows:

- **Code:** Evaluation includes pass@1 scores for HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021).
- **Math:** Accuracy scores for GSM8K (Cobbe et al., 2021) (5-shot) and MATH (Hendrycks et al., 2021) (4-shot) benchmarks.
- **Commonsense Reasoning (CR):** Accuracy scores for PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy, and ARC-challenge (Clark et al., 2018).
- **Word Knowledge (WK):** Assessment of 0-shot performance on NaturalQuestions (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017) utilizing the exact match (EM) metric.
- **Aggregated Benchmarks:** Overall results for MMLU (Hendrycks et al., 2020) (5-shot) utilizing accuracy scores metrics.

Notably, for more detailed experiment results, please refer to Appendix C.

Camel and Camelidae Models. We fine-tuned Camel and Camelidae models using identical datasets, IDAE-500K, to ensure fair comparisons between dense and sparse models. Specifically, Camel models are dense models while Camelidae models are sparse models with MoE architecture. Notably, to further enhance the capabilities of the sparse models, we also utilize IDAE-720K for the instruction-tuning of the Camelidae-pro model. All Camelidae models utilize the top-2 gate router.

Implementation Details. We employed QLoRA (Dettmers et al., 2024) techniques for effective fine-tuning of both the Camel and Camelidae models derived from Llama2-7B (Touvron et al., 2023b), Llama2-13B (Touvron et al., 2023b), and Yi-34B (01 AI, 2023). As for the QLoRA configuration, we used a 4-bit quantization scheme for our experiments, which significantly reduces memory usage while preserving model performance. This process entailed using a constant learning rate schedule with a warm-up ratio of 0.03, and the paged AdamW (Dettmers et al., 2024; Loshchilov and Hutter, 2017) optimizer with a learning rate of 2×10^{-4}, no weight decay, a batch size of 128, and a sequence length of 2048 tokens. The models underwent instruction tuning for one epoch on 16 A100 GPUs, each equipped with 80G memory. Please refer to Appendix B for more details.

3.2 Comparison with Chat LLMs

We present the performance of various chat LLMs on a set of standardized benchmarks. The chat models evaluated are Camelidae-8×34B-pro, Mixtral-8×7B-Instruct (Jiang et al., 2024), DeepSeekMoE-16B-Chat (Dai et al., 2024), Yi-34B-Chat (01 AI, 2023), Llama2-70B-Chat (Touvron et al., 2023b), Qwen-72B-Chat (Bai et al., 2023), and GPT-3.5 (Brown et al., 2020). The benchmarks cover a range of domains, including multiple-choice questions across 57 subjects (MMLU), grade-school math (GSM8K), math problems across various difficulty levels (MATH), Python coding tasks (HumanEval), Python code generation (MBPP), commonsense reasoning (HellaSwag), and world knowledge question answering (NaturalQuestions).

As shown in Section 3.1, Camelidae-8×34B-pro demonstrates its strengths in its wide range of knowledge, mathematical, coding, and commonsense reasoning capabilities across various sparse and dense models.

Knowledge and Reasoning Abilities. Camelidae-8×34B-pro demonstrates impressive performance on MMLU with a high success rate of 75.7%, indicating its wide-ranging professional and academic knowledge. Meanwhile, Camelidae-8×34B-pro scores 31.2% on NaturalQuestions, demonstrating a comprehensive world knowledge base. Although Camelidae-8×34B-pro is weaker than some models in the HellaSwag benchmark, its 85.2% accuracy is still decent for commonsense reasoning.

Mathematical Proficiency. Camelidae-8×34B-pro excels on the GSM8K benchmark with 79.4%

	Sparse Chat Models			Dense Chat Models			
	Camelidae 8×34B-pro	Mixtral 8×7B Inst.	DeepSeekMoE 16B Chat	Yi 34B Chat	Llama2 70B Chat	Qwen 72B Chat	GPT-3.5
MMLU (Acc.) (Hendrycks et al., 2020)	**75.7%** (5-shot)	68.7% (5-shot)	47.2% (5-shot)	74.8% (5-shot)	63.8% (5-shot)	75.0% (5-shot)	70.0% (5-shot)
GSM8K (Acc.) (Cobbe et al., 2021)	**79.4%** (5-shot)	71.7% (5-shot)	62.2% (5-shot)	67.6% (5-shot)	59.3% (5-shot)	67.4% (5-shot)	57.1% (5-shot)
MATH (Acc.) (Hendrycks et al., 2021)	24.0% (4-shot)	22.1% (4-shot)	15.2% (4-shot)	17.3% (4-shot)	10.4% (4-shot)	26.8% (4-shot)	**34.1%** (4-shot)
HumanEval (Pass@1) (Chen et al., 2021)	**48.8%** (0-shot)	25.6% (0-shot)	42.7% (0-shot)	20.1% (0-shot)	32.3% (0-shot)	47.0% (0-shot)	48.1% (0-shot)
MBPP (Pass@1) ((Austin et al., 2021)	**43.2%** (4-shot)	40.6% (4-shot)	42.2% (4-shot)	41.0% (4-shot)	35.6% (4-shot)	41.8% (4-shot)	-
HellaSwag (Acc.) (Zellers et al., 2019)	85.2% (10-shot)	**86.5%** (10-shot)	72.2% (10-shot)	83.9% (10-shot)	84.8% (10-shot)	85.9% (10-shot)	85.5% (10-shot)
NaturalQuestions (EM) (Kwiatkowski et al., 2019)	**31.2%** (0-shot)	22.5% (0-shot)	30.7% (0-shot)	23.7% (0-shot)	30.6% (0-shot)	29.3% (0-shot)	-

Table 1: Performance of Camelidae-8×34B-pro on academic benchmarks. We present a detailed comparison of the Camelidae-8×34B-pro model with the various open-source sparse chat models and dense chat models. We bold the highest scores among all models.

	Camel-7B	Camelidae 8×7B	Camel-13B	Camelidae 8×13B	Camel-34B	Camelidae 8×34B	Camelidae 8×34B-pro
# Total Params	7B	8B	13B	15B	34B	38B	38B
# Activated Params	7B	7B	13B	14B	34B	35B	35B
# Training Instructions	500K	500K	500K	500K	500K	500K	720K
MMLU (Acc.)	47.7	**48.3**	**54.4**	**54.4**	75.3	75.6	**75.7**
HumanEval (Pass@1)	17.7	**18.3**	28.7	**30.6**	42.1	43.9	**48.8**
MBPP (Pass@1)	21.0	**23.4**	30.3	**30.4**	40.6	41.4	**43.2**
GSM8K (Acc.)	40.7	**44.0**	50.2	**52.6**	76.1	78.3	**79.4**
MATH (Acc.)	4.8	**5.8**	8.4	**9.8**	18.2	22.6	**24.0**
PIQA (Acc.)	79.7	**79.9**	**80.9**	**80.9**	82.3	82.7	**83.6**
HellaSwag (Acc.)	**76.8**	**76.8**	79.8	**80.1**	82.6	**83.2**	82.5
Winogrande (Acc.)	71.3	**72.1**	74.6	**74.7**	80.0	**80.9**	80.1
ARC-easy (Acc.)	**75.0**	**75.0**	77.7	**78.8**	86.1	86.2	**86.6**
ARC-challenge (Acc.)	47.9	**49.6**	**54.3**	54.2	63.6	**65.2**	63.3
NaturalQuestions (EM)	17.6	**17.8**	24.7	**26.8**	31.6	**32.2**	31.2
TriviaQA (EM)	**51.0**	**51.0**	57.5	**59.4**	63.3	**63.4**	62.5

Table 2: Overall performance on all the evaluation benchmarks of dense models (Camel) and sparse (Camelidae) models across different model sizes. We bold the highest scores separately for different model sizes.

accuracy, the highest among models. However, its 24.0% score on the MATH benchmark lags behind GPT-3.5, indicating a relative weakness in solving more complex mathematical problems.

Coding Skills. Camelidae-8×34B-pro demonstrates strong coding abilities with 48.8% accuracy on the HumanEval benchmark, comparable to GPT-3.5, and a 43.2% pass rate on the MBPP Python code generation benchmark, showcasing its prowess in understanding and generating code.

3.3 Ablation Studies

Dense models vs. Sparse Models. We evaluate the efficacy of our novel training methodology through a comparative analysis of Camelidae models, encompassing both dense and sparse configurations across various parameter sizes, as delineated in Table 2 and Table 3. Camelidae models demonstrate a significant advantage over counterparts across different model sizes. This superiority is particularly evident in tasks requiring a deeper understanding, including code and mathematical benchmarks, highlighting the efficacy of our training approach in augmenting model capabilities. To ensure equitable comparisons, Camel and Camelidae models were fine-tuned using the same dataset, IDAE-500K. As indicated in Table 2, the Camelidae models, as sparse models, consistently display superior performance over the dense Camel models of comparable sizes. Moreover, Camelidae-8x34B-pro, which is trained utilizing the IDAE-720K dataset, outperforms Camelidae-8x34B which indicates that the

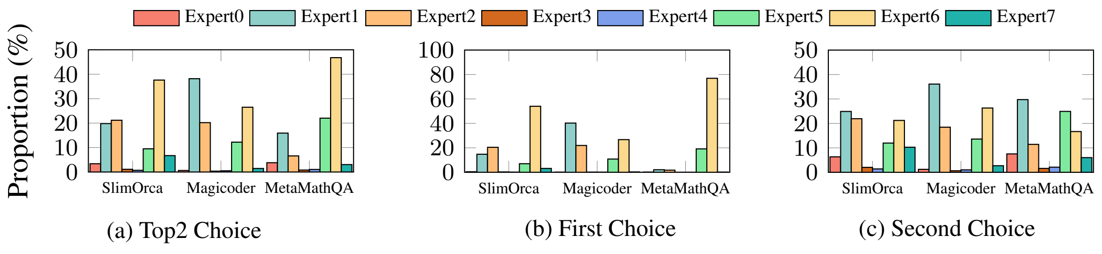

Figure 4: Proportion of tokens assigned to each expert on different dataset subsets.

Model	# Params	Avg.	Code	Math	CR	WK	MMLU
Llama2-7B-Chat	7B	35.4	14.9	15.1	66.7	33.0	47.3
Vicuna-7B	7B	34.0	9.6	13.5	67.6	29.2	**50.1**
Camelidae-8×7B	8B	**39.9**	**20.9**	**24.9**	**70.7**	34.4	48.3
Llama2-13B-Chat	13B	41.8	23.1	21.2	70.9	40.0	53.8
Vicuna-13B	13B	39.9	10.7	21.0	70.8	41.1	**55.8**
Camelidae-8×13B	15B	**46.5**	**30.5**	**30.7**	**73.8**	**43.1**	54.4
Yi-34B-Chat	34B	51.8	30.4	42.5	73.3	38.0	74.8
SUSChat-34B	34B	53.3	25.9	47.2	78.8	38.3	**76.4**
Camelidae-8×34B	38B	59.3	42.7	50.5	**79.7**	**47.8**	75.6
Camelidae-8×34B-pro	38B	**59.9**	**46.0**	**51.7**	79.2	46.9	75.7

Table 3: Overall performance on grouped benchmarks of various dense models (Llama2-Chat (Touvron et al., 2023b), Vicuna (Zheng et al., 2023), Yi-Chat (01 AI, 2023), SUSChat (SUSTech IDEA, 2023)) across different model sizes. We bold the highest scores separately for different model sizes.

effectiveness of our method is sustained even with the increment of the training data volume.

Numbers of Experts. The results from the study, as shown in Table 4, clearly demonstrate that increasing the number of experts in the MoE layers significantly enhances the model's performance. This trend is evident in the progressive improvement in scores across various academic benchmarks as the number of experts increases from 4 to 16 in the Camelidae models. Notably, the Camelidae-16×7B model exhibits exceptional performance on all the benchmarks. This positive correlation between the number of experts and the model's performance indicates the untapped potential of our approach. Specifically, a further increase in the number of experts might yield even more substantial advancements in model performance.

3.4 Routing Analysis

Our study rigorously examined the expert selection process by the router, with a keen focus on ascertaining whether specific experts demonstrate specialization in distinct domains such as coding and mathematics.

This inquiry involved a thorough analysis of the distribution patterns of selected experts across various dataset subsets. These included SlimOrca (Lian et al., 2023; Mukherjee et al., 2023; Longpre et al., 2023), Magicoder (Wei et al., 2023), and MetaMathQA (Yu et al., 2023). The outcomes of this analysis are depicted in Figure 4, with particular emphasis on the 15th layers of the Camelidae-8×7B model.

Model	# Experts	Avg.	Code	Math	CR	WK	MMLU
Camelidae-4×7B	4	39.6	20.7	24.3	70.2	33.3	49.3
Camelidae-8×7B	8	39.9	20.9	24.9	**70.7**	34.4	48.3
Camelidae-16×7B	16	**40.5**	**21.6**	**25.8**	**70.7**	**35.0**	**49.4**

Table 4: Evaluation on different numbers of experts in the MoE layers. We bold the highest scores for each grouped benchmark.

Our findings highlight discernible variations in the distribution of experts among the three datasets. For instance, Expert 1 exhibits a notably higher activation within the Magicoder dataset, while Expert 6 demonstrates a significant activation rate in the MetaMathQA dataset relative to other experts. These observations suggest that the router operates with a structured syntactic approach. Importantly, despite the variation in expert selection across different datasets, certain experts (specifically Experts 1, 2, 5, and 6) consistently exhibit elevated activation rates.

4 Related Work

4.1 Dense and Sparse Models

Traditional dense models activate all parameters during training and inference, leading to high computational and memory requirements as model sizes increase. In contrast, sparse models, employing the MoE architecture (Shazeer et al., 2017), activate only a subset of the total available parameters for each input token. In sparse models, the FFN layer is replaced by an MoE layer, directing each input token to a select group of expert networks for processing. The final token representation is an amalgamation of outputs from these chosen experts. Despite an increase in parameters, the sparse activation of experts ensures computational efficiency while enhancing model capabilities. The sparse models with MoE architecture have been extensively explored in the field of NLP (Lepikhin et al., 2020; Du et al., 2022; Fedus et al., 2022), particularly with its integration into the transformer block. Our approach adopts the routing strategy from (Lepikhin et al., 2020; Du et al., 2022), with selective parameter activation to achieve computational efficiency.

4.2 Reuse of Trained Weights

Recent studies have focused on improving training efficiency by leveraging pre-existing model weights for a warm start, thus minimizing training expenses (Chen et al., 2015; Rae et al., 2021; Yang et al., 2021; Lin et al., 2021; Lan et al., 2019). Sparse Upcycling (Komatsuzaki et al., 2023) introduces a methodology to initialize sparse MoE models using weights from a pre-trained dense model. This approach significantly reduces the computational resources needed compared to the training of the original dense model. Sparse Upcycling involves the direct transfer of layer normalization, attention, and embedding parameters from the dense model to the new sparse model. Moreover, it replaces some Multilayer Perceptron (MLP) layers with MoE layers, initializing the experts in these layers with weights from the dense model's MLP. This process effectively transfers valuable learned representations from the dense model's pre-training phase into the sparse model. In our research, we adopt this method, reusing weights from a pre-trained dense model for our PESC method.

4.3 Parameter-Efficient Fine-Tuning

Traditionally, full fine-tuning has been the norm for adapting pre-trained models, including LLMs. However, due to the immense size of LLMs, this approach demands substantial computational resources. To mitigate this, numerous PEFT methods have emerged (Houlsby et al., 2019; Hu et al., 2021; Li and Liang, 2021; Liu et al., 2022; Wu et al., 2024a). PEFT focuses on training a limited subset of parameters, either from the existing model or newly added ones. Adapter-based methods (Houlsby et al., 2019; Hu et al., 2021; Liu et al., 2022; Wu et al., 2024a) integrate small, learnable modules called adapters into pre-trained models, fine-tuning only these newly inserted parameters. Among these, QLoRA (Dettmers et al., 2024) has gained popularity for its efficiency in fine-tuning LLMs, yielding results comparable to full fine-tuning. Another emerging trend in PEFT is prefix-/prompt-tuning (Lester et al., 2021; Li and Liang, 2021), involving the addition of learnable token vectors to either the keys and values in attention modules or directly to the input sequence. In this study, we insert adapters after the copied FFN layers to construct MoE layers and employ QLoRA to update the other weight metrics of LLMs.

4.4 Mixture of LoRA Experts

Other works also explore the combination of MoE with PEFT techniques (Diao et al., 2023; Gou et al., 2023; Wu et al., 2024b; Liu et al., 2023; Luo et al., 2024; Dou et al., 2024). For instance, LoRAMoE (Dou et al., 2024) focuses on the retention of world knowledge, and MoELoRA (Luo et al., 2024) focuses on the Math and CommonSense Reasoning ability utilizing PEFT frameworks which unify MOE and LoRA. However, the mixture of LoRA framework incurs additional computational costs including higher memory usage and slower speed without parallelism during the training and inference process. Our PESC method, in contrast, does not face these challenges. PESC builds on the adapter-based model framework, fine-tuning multiple adapters inserted after the copied FFN layers instead of all the copied FFN layers in corresponding experts. In our MoE design of PESC, each expert utilizes a single adapter module, significantly reducing the overall memory footprint compared to LoRA module, which would require multiple modules per expert due to its placement in FFN and attention layers. This distinction is particularly crucial when dealing with a large number of experts, as memory constraints become increasingly challenging. Moreover, our adapter-based experts enable parallel computation across experts due to their independence from each other's outputs, unlike LoRA, where dependencies between layers could limit parallelism. This design accelerates training time, especially in scenarios where the number of experts grows large, ensuring scalability and efficiency. It's also worth noting that LoRA might require merging weights into the main model for inference, leading to increased memory usage and potential latency issues, especially since multiple tokens activate different experts. On the contrary, the adapter-based parameter-efficient MoE does not impose such overhead during inference, maintaining a low computational burden similar to the original dense model.

5 Conclusion

In this paper, we introduce Parameter-Efficient Sparsity Crafting (PESC) which upcycles dense models into sparse models utilizing the MoE architecture. PESC incorporates adapters (Houlsby et al., 2019) within the MoE layers of sparse models, enabling the differentiation of experts without modifying the individual weights of each expert, and guarantees the quality of the approximation compared to traditional sparsity upcycling (Komatsuzaki et al., 2023) in function space (Section 2.2). This technique significantly reduces computational costs and GPU memory requirements compared to sparse upcycling. It facilitates the expansion of model capacity with a minimal parameter increase due to the integration of adapters. We apply the PESC method to instruction tuning across various general tasks, resulting in notable performance enhancements on various benchmarks (Section 3). Additionally, we develop sparse models, Camelidae, using the PESC approach and achieve superior performance across various open-source sparse models and demonstrate superior general capabilities compared to GPT-3.5.

Limitation

The PESC method introduces slightly more parameters compared to some PEFT techniques (LoRA, etc.). The instruction tuning process of the sparse models utilizing the PESC method would require more GPU memory and computation time compared to dense models. Although PESC enhances the performance of instruction tuning for general tasks, it may still not match the performance of sparse upcycling with full fine-tuning, as PESC is a mathematical approximation of sparse upcycling as illustrated in Equation (6).

Acknowledgement

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. CUHK14210723 and No. CUHK14211824), and the MIND project (MINDXZ202404).

References

01 AI. 2023. Yi. https://github.com/01-ai/Yi.

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*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PiQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*.

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. In *Advances in neural information processing systems*.

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*.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2015. Net2Net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

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*.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. 2024. DeepSeekMoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*.

Shizhe Diao, Tianyang Xu, Ruijia Xu, Jiawei Wang, and Tong Zhang. 2023. Mixture-of-Domain-Adapters: Decoupling and Injecting Domain Knowledge to Pretrained Language Models' Memories. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 5113–5129.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2022. Delta Tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, et al. 2024. LoRAMoE: Alleviating World Knowledge Forgetting in Large Language Models via MoE-Style Plugin. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1932–1945.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. GLaM: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*.

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*.

Ken-Ichi Funahashi. 1989. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Yunhao Gou, Zhili Liu, Kai Chen, Lanqing Hong, Hang Xu, Aoxue Li, Dit-Yan Yeung, James T Kwok, and Yu Zhang. 2023. Mixture of Cluster-conditional LoRA Experts for Vision-language Instruction Tuning. *arXiv preprint arXiv:2312.12379*.

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. *arXiv preprint arXiv:2103.03874*.

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*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

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*.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*.

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*.

Patrick Kidger and Terry Lyons. 2020. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2023. Sparse Upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations*.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural Questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. AlBert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Conference on Empirical Methods in Natural Language Processing*.

Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *The Association for Computational Linguistics*.

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.

Junyang Lin, An Yang, Jinze Bai, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Yong Li, Wei Lin, et al. 2021. M6-10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. *arXiv preprint arXiv:2110.03888*.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*.

Qidong Liu, Xian Wu, Xiangyu Zhao, Yuanshao Zhu, Derong Xu, Feng Tian, and Yefeng Zheng. 2023. MoELoRA: An MoE-based parameter efficient finetuning method for multi-task medical applications. *arXiv preprint arXiv:2310.18339*.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. 2023. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. 2024. MoELoRA: Contrastive learning guided mixture of experts on parameter-efficient fine-tuning for large language models. *arXiv preprint arXiv:2402.12851*.

Mistral AI. 2023. Mistral. `https://mistral.ai/news/announcing-mistral-7b//`.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of GPT-4. *arXiv preprint arXiv:2306.02707*.

OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

OpenCompass. 2023. OpenCompass: A Universal Evaluation Platform for Foundation Models. `https://github.com/open-compass/opencompass`.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. 2023. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*.

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*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Sheng Shen, Le Hou, Yanqi Zhou, Nan Du, Shayne Longpre, Jason Wei, Hyung Won Chung, Barret Zoph, William Fedus, Xinyun Chen, et al. 2023. Mixture-of-experts meets instruction tuning: A winning combination for large language models. *arXiv preprint arXiv:2305.14705*.

SUSTech IDEA. 2023. SUSChat. `https://github.com/SUSTech-IDEA/SUS-Chat`.

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`.

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*.

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*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy

Liang, Jeff Dean, and William Fedus. 2022. Emergent Abilities of Large Language Models. *Journal of Machine Learning Research*.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Haoyuan Wu, Xinyun Zhang, Peng Xu, Peiyu Liao, Xufeng Yao, and Bei Yu. 2024a. p-Laplacian Adaptation for Generative Pre-trained Vision-Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 6003–6011.

Xu Wu, Shaohan Huang, and Furu Wei. 2024b. MoLE: Mixture of loRA experts. In *International Conference on Learning Representations*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2024. WizardLM: Empowering large language models to follow complex instructions. In *International Conference on Learning Representations*.

Shuo Yang, Le Hou, Xiaodan Song, Qiang Liu, and Denny Zhou. 2021. Speeding up deep model training by sharing weights and then unsharing. *arXiv preprint arXiv:2110.03848*.

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*.

Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena.

A Details of IDAE Datasets

We show the proportion of SlimORCA (Lian et al., 2023; Mukherjee et al., 2023; Longpre et al., 2023), Magicoder (Wei et al., 2023), and MetaMathQA (Yu et al., 2023) datasets in IDAE-500K and IDAE-720K datasets in Table 5.

	SlimOrca	Magicoder	MetaMathQA
IDAE-500K	300K	100K	100K
IDAE-720K	360K	180K	180K

Table 5: The proportion of SlimORCA, Magicoder, and MetaMathQA datasets in IDAE datasets.

B Implementation Details

We show the hyperparameters that we use for instruction tuning in Table 6.

lr	epoch	LoRA r	LoRA α	Quant Type	Adapter Dim
2×10^{-4}	1	64	16	nf4	512

Table 6: Hyperparameters of instruction tuning.

C Detailed Evaluation Results on Grouped Benchmarks.

We show the detailed evaluation results of each grouped academic benchmark as follows:

- In Table 7, we report the evaluation details of the MMLU benchmark.
- In Table 8, we report the results on GSM8K and MATH benchmarks.
- In Table 9, we compare the results on HumanEval and MBPP benchmarks.
- In Table 10, we show the results on several commonsense reasoning benchmarks.
- In Table 11, We evaluate the performance on NaturalQuestions and TriviaQA benchmarks.

	Humanities	STEM	Social Sciences	Other	Average
LLaMA2-7B	43.2	36.9	51.7	52.6	45.7
LLaMA2-7B-Chat	43.4	38.7	54.7	54.6	47.3
Vicuna-7B	46.0	40.4	58.2	58.1	50.1
Camel-7B	43.9	38.5	55.9	54.6	47.7
Camelidae-8×7B	44.7	38.1	56.9	55.9	48.3
LLaMA2-13B	52.3	44.1	63.7	62.0	55.1
LLaMA2-13B-Chat	50.3	43.9	62.6	60.3	53.8
Vicuna-13B	52.1	44.6	65.3	63.5	55.8
Camel-13B	52.0	42.2	63.0	61.7	54.4
Camelidae-8×13B	52.1	43.3	62.6	61.1	54.4
Yi-34B	71.3	67.3	85.4	80.2	75.5
Yi-34B-Chat	70.5	66.3	84.7	79.9	74.8
SUSChat-34B	72.2	69.6	85.5	80.5	76.4
Camel-34B	72.5	67.3	84.0	79.3	75.3
Camelidae-8×34B	72.8	66.7	83.8	80.4	75.6
Camelidae-8×34B-pro	73.8	66.0	83.8	80.3	75.7

Table 7: Comparison on the performance of MMLU.

	GSM8K	MATH	Average
LLaMA2-7B	16.7	3.3	10.0
LLaMA2-7B-Chat	16.7	3.3	10.0
Vicuna-7B	16.7	3.3	10.0
Camel-7B	40.7	4.8	22.8
Camelidae-8×7B	44.0	5.8	24.9
LLaMA2-13B	29.6	5.0	17.3
LLaMA2-13B-Chat	16.7	3.3	10.0
Vicuna-13B	16.7	3.3	10.0
Camel-13B	50.2	8.4	29.3
Camelidae-8×13B	52.6	9.8	30.7
Yi-34B	67.9	15.9	41.9
Yi-34B-Chat	16.7	3.3	10.0
SUSChat-34B	16.7	3.3	10.0
Camel-34B	76.1	18.2	47.2
Camelidae-8×34B	78.3	22.6	50.5

Table 8: Comparison on mathematical reasoning tasks.

	HumanEval	MBPP	Average
LLaMA2-7B	12.8	14.8	13.8
LLaMA2-7B-Chat	16.7	3.3	10.0
Vicuna-7B	16.7	3.3	10.0
Camel-7B	17.7	21.0	19.4
Camelidae-8×7B	18.3	23.4	20.9
LLaMA2-13B	18.9	26.8	22.9
LLaMA2-13B-Chat	16.7	3.3	10.0
Vicuna-13B	16.7	3.3	10.0
Camel-13B	28.7	30.3	29.5
Camelidae-8×13B	30.6	30.4	30.5
Yi-34B	26.2	38.2	32.2
Yi-34B-Chat	16.7	3.3	10.0
SUSChat-34B	16.7	3.3	10.0
Camel-34B	42.1	40.6	41.4
Camelidae-8×34B	43.9	41.4	42.7

Table 9: Comparison on code generation tasks.

	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	Average
LLaMA2-7B	78.9	75.9	69.5	74.7	46.2	69.0
LLaMA2-7B-Chat	77.0	75.5	66.4	69.7	44.7	66.7
Vicuna-7B	78.0	73.7	69.3	71.3	45.8	67.6
Camel-7B	79.7	76.8	71.3	75.0	47.9	70.1
Camelidae-8×7B	79.9	76.8	72.1	75.0	49.6	70.7
LLaMA2-13B	80.7	80.8	71.9	77.4	48.9	71.6
LLaMA2-13B-Chat	79.1	79.7	71.3	73.8	50.3	70.9
Vicuna-13B	78.9	77.4	71.9	74.8	50.9	70.8
Camel-13B	80.9	79.8	74.6	77.7	54.3	73.5
Camelidae-8×13B	80.9	80.1	74.7	78.8	54.2	73.8
Yi-34B	82.9	83.7	78.9	84.1	61.6	78.2
Yi-34B-Chat	79.9	80.7	77.1	74.3	54.6	73.3
SUSChat-34B	82.0	83.0	81.0	84.8	63.0	78.8
Camel-34B	82.3	82.6	80.0	86.1	63.6	78.9
Camelidae-8×34B	82.7	83.2	80.9	86.2	65.2	79.7
Camelidae-8×34B-pro	83.6	82.5	80.1	86.6	63.3	79.2

Table 10: Comparison on the performance of various commonsense reasoning tasks.

	NaturalQuestions	TriviaQA	Average
LLaMA2-7B	19.1	52.8	36.0
LLaMA2-7B-Chat	19.6	46.4	33.0
Vicuna-7B	15.6	42.8	29.2
Camel-7B	17.6	51.0	34.3
Camelidae-8×7B	17.8	51.0	34.4
LLaMA2-13B	24.8	59.4	42.1
LLaMA2-13B-Chat	25.0	55.0	40.0
Vicuna-13B	25.8	56.3	41.1
Camel-13B	24.7	57.5	41.1
Camelidae-8×13B	26.8	59.4	43.1
Yi-34B	33.5	62.1	47.8
Yi-34B-Chat	23.7	52.3	38.0
SUSChat-34B	20.4	56.1	38.3
Camel-34B	31.6	63.3	47.5
Camelidae-8×34B	32.2	63.4	47.8
Camelidae-8×34B-pro	31.2	62.5	46.9

Table 11: Comparison on the exact match performance of world knowledge tasks.