ModularMoE: Fast LLM Customization with Parameter-Sharing Mixture-of-Experts for Low-Resource Settings

Jiaxing Liu[1], Qi Qi[1]*, Haifeng Sun[1], Dunjun Li[1], Zirui Zhuang[1], Bo He[1], Xiang Yang[1], Cong Liu[2], Jianxin Liao[1], Jingyu Wang[1]*

[1]State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications,
[2]China Mobile Communications Company Limited Research Institute

Correspondence: {jxing,qiqi8266,hfsun,lidunjun,zhuangzirui,hebo,yangxiang,jxlbupt,wangjingyu}@bupt.edu.cn

Abstract

The massive size of Large Language Models (LLMs) imposes substantial computational and storage burdens, particularly on devices with limited hardware resources. Compared to foundation models, smaller and more specialized models are often more suitable for practical deployment. Existing customization approaches, such as the conventional "prune-then-finetune" paradigm or task-agnostic deployment strategies, either incur excessive computational costs or lead to suboptimal task performance. The recently popular Mixture-of-Experts (MoE) architecture exhibits a strong ability to mitigate inter-task interference, offering a new perspective on model deployment. In this paper, we introduce ModularMoE, a training framework that converts pre-trained LLMs into parameter-sharing MoE models for lightweight deployment. Exploiting the emergent modularity within LLMs, we split the feed-forward layers into multiple disjoint modules. Each expert is then constructed as a combination of such modules, enabling knowledge sharing across experts and thereby improving parameter efficiency within MoEs. Extensive experiments across multiple downstream tasks demonstrate that ModularMoE outperforms other state-of-the-art baselines at the same sparsity level, achieving an average performance improvement of 4.10% to 28.75% while delivering up to 2.71× inference speedup.

1 Introduction

Large Language Models (LLMs) (Achiam et al., 2023; Touvron et al., 2023; GLM et al., 2024) have demonstrated exceptional performance across a wide range of natural language tasks, revolutionizing people's lives in unprecedented ways. With the increasing number of models being proposed, there is growing interest in deploying them for personalized services. However, their vast number of parameters results in significant storage and computational overhead. Equipped with consumer-grade graphics processing units (GPUs), personal devices typically possess limited memory and computational power. Moreover, real-world scenarios demand higher performance on specific tasks. Compared to foundation models, smaller and more specialized models are more aligned with the requirements of practical deployment.

As shown in Figure 1, existing research generally adopts two approaches to achieve the customization. *Task-Specific Deployment* compresses and fine-tunes models both on the target task. It often achieves superior performance, but at the cost of increased computational time and expensive resources. Therefore, it is not suitable for practical scenarios, which often exhibit varying computational capabilities and task requirements. *Task-Agnostic Deployment* performs customization on public datasets, aiming to retain the general-purpose capabilities of LLMs. However, fine-tuning on task-agnostic datasets may cause negative interference (McCloskey and Cohen, 1989) between tasks and catastrophic forgetting of existing knowledge (French, 1999), preventing the model from achieving optimal performance on target tasks. **The challenge is how to rapidly achieve both resource-oriented and task-oriented customization for diverse low-resource scenarios.**

The recently popular Mixture-of-Experts (MoE) architecture offers a new perspective on model deployment. The MoE comprises a series of experts, each responsible for handling a different aspect of the input space. The independence of experts avoids negative interference between tasks and the forgetting of previously acquired knowledge, enabling the model to store multiple domain-specific information without distortion. However, compared to dense models, MoEs typically contain more parameters, which contradicts the objective of lightweighting. Therefore, **the subsequent chal-**

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 3562–3575
July 2-7, 2026

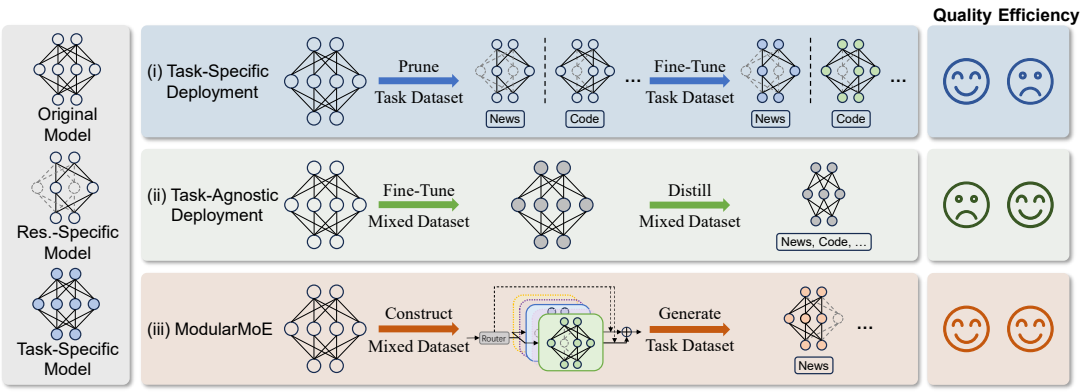


Figure 1: Comparison of different deployment strategies: (i) Task-Specific Deployment: compress and fine-tune the model for each specific scenario; (ii) Task-Agnostic Deployment: perform compression and fine-tuning on task-agnostic public datasets; (iii) ModularMoE: transform the vanilla LLM into the MoE model and ensemble relevant experts for deployment.

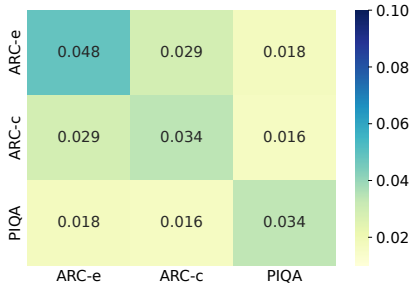


Figure 2: The average distribution similarity of functional neurons for different tasks on LLaMA2-7B. A larger value means a higher similarity in the distributions of functional neurons across tasks.

lenge lies in how to reduce the size of MoEs.

Our solution is parameter sharing between experts. Zhang et al. (2023) denoted the crucial neurons for a task as the *functional neurons*, and studied the similarity in the distributions of functional neurons for different tasks. As shown in Figure 2, the large overlays of the same task indicate that there are some groups of neurons activating jointly in response to specific features, structurally suggesting the emergence of modularity (i.e., *Emergent Modularity*). The high overlap between similar tasks (e.g., ARC-e and ARC-c) indicates that there are some neurons capable of multiple functions. This phenomenon inspired the idea that neurons could be shared among different experts. Specifically, we define experts as combinations of modules, enabling parameter sharing across experts and thereby reducing the size of MoEs.

In this paper, we propose ModularMoE, a training framework that converts pre-trained LLMs into parameter-sharing MoE models for lightweight deployment. ModularMoE consists of three main stages. First, we partition the feed-forward layers of the model into multiple disjoint modules. Next, we construct task-specific experts by sparsely activating a subset of modules for each task. Finally, we perform multi-task fine-tuning over all experts to enhance their specialization and complete the construction of the MoE model. Extensive experiments across multiple tasks demonstrate that ModularMoE outperforms other state-of-the-art methods at the same sparsity level, achieving an average performance improvement of 4.10% to 28.75% and delivering up to 2.71× inference speedup.

Our contributions are summarized as follows:

- We innovatively formulate experts as compositions of modules, enabling parameter sharing across experts to significantly improve parameter efficiency in MoEs.
- We develop an efficient masking-based MoE computation method, achieving 5.26× speedup during multi-task fine-tuning.
- We propose ModularMoE for rapid model customization, with extensive experiments across diverse models and tasks validating its effectiveness and practicality.

2 Preliminaries

2.1 Mixture-of-Experts for Transformers

The feed-forward network (FFN) layers, accounting for nearly two-thirds of the total model parameters, are crucial components storing a substantial amount of language and factual knowledge (Geva et al., 2021). A traditional FFN layer consists of two linear layers, processing input $\boldsymbol{x}$ from the input space $\mathcal{X} \subset \mathbb{R}^d$ as follows:

$$F(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{W}^I)\boldsymbol{W}^O, \quad (1)$$

where $\boldsymbol{W}^I \in \mathbb{R}^{d\times d_{ff}}, \boldsymbol{W}^O \in \mathbb{R}^{d_{ff}\times d}$ are the weight matrices, $\sigma(\cdot)$ is a non-linear activation function, d is the input dimension, d_{ff} is the dimension of intermediate hidden states. For brevity, we omit the bias vectors. The neuron n_i is defined as a column vector $\boldsymbol{W}^I_{:,i}$ and a row vector $\boldsymbol{W}^O_{i,:}$. The output of FFN can be viewed as the sum of the outputs of d_{ff} neurons:

$$
\begin{aligned}
a_i(\boldsymbol{x}) &= \sigma(\boldsymbol{x}\cdot\boldsymbol{W}^I_{:,i}),\\
F(\boldsymbol{x}) &= \sum_{i=1}^{d_{ff}} a_i(\boldsymbol{x})\boldsymbol{W}^O_{i,:},
\end{aligned}
\tag{2}
$$

where a_i is the activation value of neuron n_i.

The traditional approach to constructing MoEs involves replacing the FFN layers with MoE layers. An MoE layer consists of n experts $f_i : \mathcal{X} \to \mathbb{R}^d, i \in \{1, 2, \ldots, n\}$ and a routing function $r : \mathcal{X} \to \mathbb{R}^n$. The routing function r assigns weights to these experts, and the output of MoE is the weighted sum of the outputs from experts. For any input $\boldsymbol{x} \in \mathcal{X}$, the output of an MoE layer is:

$$
\sum_{i=1}^{n} f_i(\boldsymbol{x})r(\boldsymbol{x})_i, \quad \text{s.t. } r(\boldsymbol{x}) \geq 0. \tag{3}
$$

The Top-k routing function is calculated as follows:

$$
r\left(\boldsymbol{x}\right) = \text{Softmax}(\text{TopK}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})), \tag{4}
$$

where $\boldsymbol{W} \in \mathbb{R}^{d\times n}, \boldsymbol{b} \in \mathbb{R}^n$ are trainable parameters. The $\text{TopK}(\cdot)$ sets all elements in the vector to negative infinity except the largest k elements. After processing with the $\text{Softmax}(\cdot)$, at most k experts have non-zero weights, ensuring the computational efficiency of MoE.

2.2 Emergent Modularity in LLMs

Modularity is defined as the correspondence between strongly interconnected components of a system and the functions they perform (Baldwin and Clark, 2000). It can enhance a system's adaptability, robustness and interpretability, and is commonly found in complex artificial and biological systems. The MoE architecture aligns with the modularization paradigm, where the parameters within each expert are coupled as a module. Traditional MoEs adopt an explicit modular structure, where predefined experts are randomly initialized and specialized for different functionalities during pre-training. Instead of training from scratch, Zhang et al. (2023) discovered some groups of neurons may be co-activated in response to specific input features, indicating the emergence of spontaneous modularity in pre-trained transformer models. They partitioned the parameters of FFNs into multiple functional segments as experts, and such MoEs are referred to as Emergent MoEs (EMoEs).

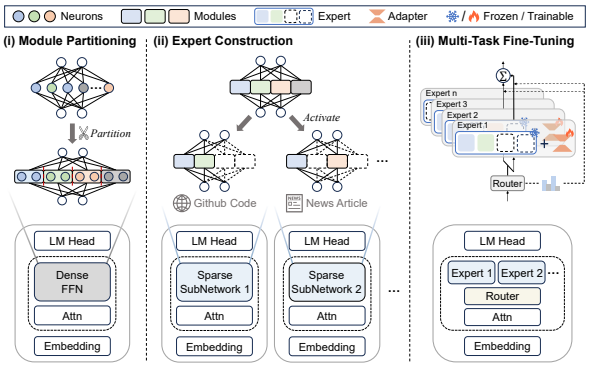


Figure 3: Overview of ModularMoE: (i) partition the original FFNs into multiple modules; (ii) sparsely activate modules based on task requirements to construct experts; (iii) perform multi-task fine-tuning to enhance the specialization of experts.

3 ModularMoE

3.1 Overview

As shown in Figure 3, the overall workflow of ModularMoE comprises three steps: module partitioning, expert construction, and multi-task fine-tuning.

3.2 Co-Activation-Based Module Partitioning

To cluster neurons specialized for similar functionalities, we collect neuron activation patterns across a wide range of tasks and then group neurons that are frequently co-activated. As shown in Eq. (2), we denote the activation value of neuron n_i as a_i. For models with ReLU activation, neuron n_i is considered activated when $|a_i| > 0$. Neurons n_i and n_j are considered co-activated if $|a_i \cdot a_j| > 0$. We utilize the co-activation feature to construct a neuron co-activation graph G. In this graph, each node represents a neuron. Each pair of nodes is connected by an edge, whose weight represents the co-activation value, computed as follows:

$$
w_{i,j} = \sum_{\boldsymbol{x}} \left|a_i(\boldsymbol{x}) \cdot a_j(\boldsymbol{x})\right|. \tag{5}
$$

Then, we employ the graph partitioning algorithm (Karypis and Kumar, 1998) to split the graph G into n groups. The edges within each group have high weights, indicating that the neurons within the same group are frequently co-activated. To improve the compatibility with inference frameworks,

we set the size of each module to be identical. If the number of modules is n, then the intermediate dimension of each module is $d_e = d_{ff}/n$. The parameters of the i-th module are denoted as:

$$\boldsymbol{W}_i^I \in \mathbb{R}^{d \times d_e}, \boldsymbol{W}_i^O \in \mathbb{R}^{d_e \times d}, \tag{6}$$

where d is the dimensionality of the hidden states in the model. Following Zhang et al. (2022), we use a permutation matrix $\boldsymbol{P} \in \mathbb{R}^{d_{ff} \times d_{ff}}$ to represent the partitioning process:

$$\begin{aligned} \boldsymbol{W}^I \boldsymbol{P} &= [\boldsymbol{W}_1^I, \boldsymbol{W}_2^I, ..., \boldsymbol{W}_n^I], \\ (\boldsymbol{P}^\top \boldsymbol{W}^O)^\top &= [(\boldsymbol{W}_1^O)^\top, (\boldsymbol{W}_2^O)^\top, ..., (\boldsymbol{W}_n^O)^\top], \end{aligned} \tag{7}$$

and it is worth noting that the permutation matrix does not influence the output of FFN:

$$\begin{aligned} \sigma(\boldsymbol{x}\boldsymbol{W}^I)\boldsymbol{W}^O &= \sigma(\boldsymbol{x}\boldsymbol{W}^I)\boldsymbol{P}\boldsymbol{P}^\top \boldsymbol{W}^O, \\ &= \sigma(\boldsymbol{x}\underline{\boldsymbol{W}^I \boldsymbol{P}})\underline{\boldsymbol{P}^\top \boldsymbol{W}^O}. \end{aligned} \tag{8}$$

For models with non-ReLU activation, we first replace the activation with ReLU, followed by a rapid post-training to restore model performance. The resulting models retained over 97% of the original performance, with task-specific results detailed in Appendix D. The minor performance decrease is compensated by the restored sparsity and significant acceleration potential. For models replacing the FFN with a GLU variant (Shazeer, 2020), the activation of neuron n_i will be:

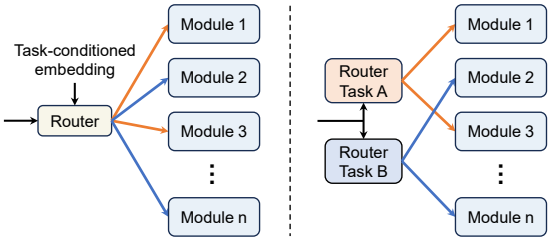
$$a_i(\boldsymbol{x}) = \sigma(\boldsymbol{x} \cdot \boldsymbol{W}_{:,i}^G) \odot (\boldsymbol{x} \cdot \boldsymbol{W}_{:,i}^I), \tag{9}$$

where $\boldsymbol{W}^G \in \mathbb{R}^{d \times d_{ff}}$ is the gating matrix in the GLU. The remaining computation remains the same as in the standard FFN.

3.3 Expert Construction via Sparse Routing

When constructing task-specific experts, ModularMoE employs a task-level routing function to sparsely activate the required modules on each task. The classical Top-k routing is not suitable for two reasons: (a) the lack of smoothness can lead to issues with weight oscillation and convergence (Hazimeh et al., 2021); (b) performing a weighted sum of the module outputs would alter the original behavior of FFNs.

As an alternative, we implement the DSelect-k routing (Hazimeh et al., 2021) to construct experts. DSelect-k is a continuously differentiable sparse gate, using a learnable binary encoding mechanism to implicitly impose cardinality constraints. If the



(a) Task-conditioned MoE layer (b) Multi-gate MoE layer

Figure 4: The two variants of Multi-Task MoE layers. (a) All tasks share a single router, while a task-specific embedding determines the assignment of tokens. (b) Each task activates the modules using its own router.

number of modules is n, then each module is represented by a binary code of $m = \log_2(n)$ bits. It is implemented as follows:

$$r(\boldsymbol{Z}) = \sum_{i=1}^{k} d\left(S\left(\boldsymbol{Z}_{i,:}\right)\right), \tag{10}$$

where $\boldsymbol{Z}_{i,:}$, identifying the i-th module, is the i-th row of the trainable parameter $\boldsymbol{Z} \in \mathbb{R}^{k \times m}$. The smoothing function $S(\cdot)$ maps elements to the interval $[0, 1]$, extending the range of $\boldsymbol{Z}$ into the real number domain, implemented as follows:

$$S(t) = \begin{cases} 0, & \text{if } t < -\gamma/2, \\ -\frac{2}{\gamma^3}t^3 + \frac{3}{2\gamma}t + \frac{1}{2}, & \text{if } -\gamma/2 \le t \le \gamma/2, \\ 1, & \text{if } t > \gamma/2. \end{cases} \tag{11}$$

γ is a hyperparameter, which was set to 0.1 in our experiments. $S\left(\boldsymbol{Z}_{i,:}\right) \in \mathbb{R}^m$ is an m-bit binary code of a module and $d(\cdot)$ decodes the binary encoding into an n-bit one-hot representation. Finally, $r(\boldsymbol{Z})$ will be a k-hot vector (similar to one-hot but with k bits set to 1), ensuring that k modules are activated. To illustrate with an example: suppose $n = 8, k = 2$, then the module's binary encoding length $m = log_2(n) = 3$. Suppose $S(\boldsymbol{Z}) = [[0, 0, 1]\,[0, 1, 1]] \in \mathbb{R}^{2\times 3}$, then $r(\boldsymbol{Z}) = [0, 1, 0, 1, 0, 0, 0, 0]$, indicating the expert selects the 2nd and 4th modules.

Given an input space $\mathcal{X}$, an output space $\mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathbb{R} \longrightarrow \mathbb{R}$, and a dataset $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}_{i=1}^N$ with N training samples. The training objective is to minimize the model loss, formulated as:

$$\min_{\boldsymbol{Z}} \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \ell\left(y, \sum_{i=1}^{n} f_{m_i}(\boldsymbol{x})\, r(\boldsymbol{Z})_i\right), \tag{12}$$

where $f_{m_i}(\boldsymbol{x})$ is the output of the i-th module. However, the original DSelect-k suffers from excessive sparsity and difficulties in converging to a

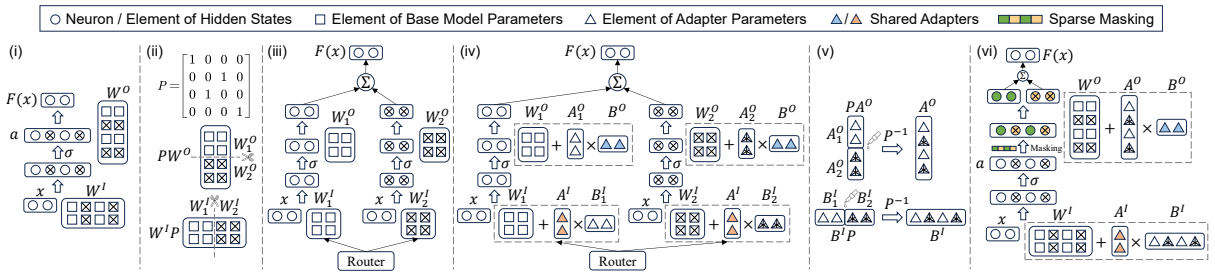


Figure 5: The overall process of the masking-based MoE implementation. (i) The FFN computation process. (ii) Construct experts via module partitioning. (iii) The base MoE layer. (iv) Attach adapters to all linear layers, with some parameters shared across modules. (v) Merge adapters along the module-partitioning dimension. (vi) Reparameterize module weights into dense FFNs, and leverage masking to enable efficient MoE computation.

binary state. To address this, we apply the TopK$(\cdot)$ function after training, setting the largest k weights to 1 and all others to 0.

Microcosmically, various experts are different combinations of modules, sharing parameters flexibly according to the relationship between tasks; macroscopically, they are different subnetworks extracted from the foundation model. Moreover, since the experts are built on separate datasets, they can be constructed in parallel and explicitly added or removed to modify the model's coverage.

3.4 Multi-Task Fine-Tuning with Masking

To further enhance the specialization of experts, we fine-tune the MoE model under a multi-task setting, where the inputs of each task are routed to corresponding modules. This conditional activation is implemented via a static, task-conditioned routing function, as illustrated in Figure 4a. To reduce memory overhead, we employ the Low-Rank Adaptation (LoRA) (Hu et al., 2021). Each trainable parameter in the model is denoted as $\boldsymbol{W}$, and its update $\Delta\boldsymbol{W}$ can be decomposed as: $\Delta\boldsymbol{W} = \boldsymbol{A}\boldsymbol{B} \in \mathbb{R}^{d_i \times d_o}$, where $\boldsymbol{A} \in \mathbb{R}^{d_i \times r}, \boldsymbol{B} \in \mathbb{R}^{r \times d_o}$, and the rank $r \ll \min\{d_i, d_o\}$. During training, $\boldsymbol{W}$ is frozen, while $\boldsymbol{A}$ and $\boldsymbol{B}$ contain the trainable parameters. For an input $\boldsymbol{x}$, the forward computation can then be expressed as: $f(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{x}\Delta\boldsymbol{W} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{x}\boldsymbol{A}\boldsymbol{B}$.

Traditionally, we attach an adapter to each linear layer within the MoE layer, as illustrated in Figure 5(iv). However, when the number of modules becomes large, it will incur substantial scheduling overhead and low device utilization. To address this, we employ sparse masking to enable efficient MoE computation. For task t_j, the neurons selected by its expert can be represented by a sparse mask $\boldsymbol{M}_j$, and the computation is formulated as:

$$f_j(\boldsymbol{x}) = \left(\sigma(\boldsymbol{x}\boldsymbol{W}^I) \odot \boldsymbol{M}_j\right) \boldsymbol{W}^O. \quad (13)$$

We then merge all adapters within the same layer into a single unified adapter, as illustrated in Figure 5(v). Since the outputs of adapters are added to the output of the original matrix, the mask will also be applied to adapters, ensuring that the task-conditioned updates of module weights work correctly. Taking the first linear layer of FFN as an example, we apply the permutation matrix $\boldsymbol{P}$ from Eq. (7) to represent the merging process:

$$\boldsymbol{A}^I[\boldsymbol{B}_1^I, \boldsymbol{B}_2^I, ..., \boldsymbol{B}_n^I] = \boldsymbol{A}^I\boldsymbol{B}^I\boldsymbol{P}, \quad (14)$$

which can be viewed as the inverse of partitioning.

The only side effect is that all modules share the same $\boldsymbol{A}$ matrix[1]. However, previous studies (Tian et al., 2024) have shown that the $\boldsymbol{A}$ matrix tends to capture the commonalities across all data, making this sharing have minimal impact on model performance. The overall process of the masking-based MoE implementation is illustrated in Figure 5.

During deployment, we train a sparse gate for each target task to select modules on demand, as shown in Figure 4b. Since the weights of all activated modules are set to 1, the final MoE layer can be further reduced to a computationally efficient dense FFN layer, whose weights are given by:

$$\begin{aligned} \boldsymbol{W}^I_{new} &= [\boldsymbol{W}^I_{i_1}, \boldsymbol{W}^I_{i_2}, ..., \boldsymbol{W}^I_{i_k}], \\ (\boldsymbol{W}^O_{new})^\top &= [(\boldsymbol{W}^O_{i_1})^\top, (\boldsymbol{W}^O_{i_2})^\top, ..., (\boldsymbol{W}^O_{i_k})^\top], \\ &\text{with } r^j(\boldsymbol{x})_{i_1,i_2,...,i_k} = 1, \end{aligned} \quad (15)$$

where r^j is the routing function for task t_j. Finally, except for the intermediate dimension of FFNs, the compressed model is structurally identical to the original model, ensuring strong compatibility with various inference frameworks. Furthermore, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices can be reparameterized as $\Delta\boldsymbol{W}$ and fused into $\boldsymbol{W}$, eliminating additional latency.

[1]The $\boldsymbol{A}$ matrix here refers broadly to the matrix in the adapter that is not zero-initialized, since exchanging the initialization of matrices may swap their roles.

Method	Sparsity	Params (B)	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
ReLU-LLaMA2-7b											
Dense	0%	6.74	74.07	43.60	66.69	69.22	42.60	78.89	0.90/89.40	40.96	63.18
Wanda	2:4	6.74	46.89	25.26	47.83	55.96	27.60	63.77	0.75/73.97	34.07	46.92
Wanda	4:8	6.74	47.52	25.34	47.77	56.75	29.60	65.18	0.77/76.37	33.59	47.76
SparseGPT	2:4	6.74	57.74	31.66	54.40	60.54	34.00	69.53	0.83/82.54	37.13	53.44
SparseGPT	4:8	6.74	59.39	34.04	56.63	61.56	36.40	71.16	**0.83/82.69**	36.17	54.76
LLM-Pruner	50%	4.84	63.51	33.53	53.49	59.43	31.00	71.65	0.72/71.46	29.38	51.68
NAEE	50%	4.84	53.28	28.07	50.26	58.41	29.60	67.36	0.76/75.09	32.06	49.26
UMC	50%	4.84	47.05	25.60	49.00	57.77	30.20	65.56	0.69/68.46	34.55	47.27
M-SMoE	50%	4.84	46.13	25.34	46.77	54.70	29.40	62.24	0.61/60.43	32.54	44.69
ModularMoE	50%	4.84	**66.54**	**36.43**	**57.11**	**65.19**	**41.40**	**74.27**	0.81/80.26	**39.14**	**57.54**
ReLU-ChatGLM3-6b											
Dense	0%	6.24	62.16	35.92	56.20	61.41	31.80	72.52	0.81/80.74	36.27	54.63
Wanda	2:4	6.24	48.15	29.61	48.94	55.33	28.00	66.43	0.72/70.74	**34.26**	47.68
Wanda	4:8	6.24	51.26	30.55	48.89	57.22	28.60	66.27	0.74/73.09	34.07	48.74
SparseGPT	2:4	6.24	49.87	31.83	48.89	56.35	28.20	66.10	0.74/73.00	33.97	48.53
SparseGPT	4:8	6.24	53.33	30.03	**50.43**	57.93	30.20	66.98	**0.76/75.00**	34.07	49.75
LLM-Pruner	50%	4.22	54.42	30.21	48.29	57.22	28.60	67.68	0.72/71.43	30.43	48.53
NAEE	50%	4.22	46.80	29.44	45.06	53.51	26.20	65.07	0.65/64.20	31.39	45.21
UMC	50%	4.22	40.74	23.98	44.37	55.64	24.60	62.24	0.62/61.74	29.47	42.85
M-SMoE	50%	4.22	40.53	24.49	42.29	51.93	26.00	59.36	0.54/53.06	29.38	40.88
ModularMoE	50%	4.22	**59.34**	**35.24**	50.29	**60.85**	**33.00**	**68.83**	0.73/72.46	**34.26**	**51.78**

Table 1: Zero-shot performance of different model compression methods on ReLU-LLaMA2-7B and ReLU-ChatGLM3-6B. "Sparsity" indicates the sparsity of FFN, and "Params" denotes the parameter count of the compressed model. "**Bold**" indicates the best performance for each task.

4 Experiments

4.1 Experimental Setup

We conducted experiments on ReLU-LLaMA2-7B (Touvron et al., 2023) and instruction-tuned ReLU-ChatGLM3-6B (GLM et al., 2024). We used the Language Model Evaluation Harness[2] to perform zero-shot classification on eight tasks: ARC-e (Clark et al., 2018), ARC-c (Clark et al., 2018), HellaSwag (HS) (Zellers et al., 2019), WinoGrande (WG) (Sakaguchi et al., 2021), OpenBookQA (OBQA) (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), ReCoRD (Zhang et al., 2018) and RACE (Lai et al., 2017). The metrics are accuracy for WinoGrande and RACE, and F1 score and Exact Match for ReCoRD. Other tasks are evaluated using normalized accuracy. The detailed descriptions of the experimental setup and hyperparameters are provided in Appendix A.

4.2 Main Results

We first constructed MoEs on small-scale datasets composed of eight tasks, with the results summarized in Table 1. We begin by comparing our approach with traditional pruning methods, such as *Wanda* (Sun et al., 2024), *SparseGPT* (Frantar and Alistarh, 2023), and *LLM-Pruner* (Ma et al., 2023). For fairness, all methods retain 50% of the FFN parameters. Notably, Wanda and SparseGPT are semi-structured pruning methods that do not directly reduce the overall parameter count. ModularMoE achieves superior performance on most tasks, with average improvements ranging from 4.10% to 22.65%. Next, we compare our method with expert-pruning approaches designed for MoEs, such as *NAEE* (Lu et al., 2024), *UMC* (He et al., 2025), and *M-SMoE* (Li et al., 2024). As shown in the table, these methods generally perform worse than ModularMoE. This is likely because they target MoEs trained from scratch, where redundancy across experts is more pronounced, making them less suitable for pruning EMoEs.

n	k	Method	Avg. Perf.
8	4	Expert	49.30
		ModularMoE	57.20
16	8	Expert	51.90
		ModularMoE	**58.50**
32	16	Expert	53.13
		ModularMoE	57.54

(a) Varying Module Count

n	k_1	k_2	Avg. Perf.
32	4	16	54.76
32	8	16	56.30
32	16	16	**57.54**
32	24	16	56.49

(b) Varying Expert Size

Table 2: Results of ablation study on ReLU-LLaMA2-7B. "Expert" refers to naive experts without fine-tuning.

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

Method	ARC-e	ARC-c	HS	WG	OBQA	PIQA	Avg.
R+T	53.41	27.90	49.09	57.46	30.80	66.59	47.54
R+D	52.82	26.62	50.71	56.51	27.60	65.67	46.66
C+T	52.90	30.72	46.60	55.25	32.60	67.41	47.58
C+D	60.27	33.45	52.60	59.83	35.20	71.93	52.21
C+D+F	**66.54**	**36.43**	**57.11**	**65.19**	**41.40**	**74.27**	**56.82**

Table 3: Zero-shot performance of ModularMoE on ReLU-LLaMA2-7B under different partitioning and routing strategies. **R** denotes random partitioning, **C** denotes co-activation-based partitioning, **T** refers to Top-k routing, **D** refers to DSelect-k routing, and **F** indicates multi-task fine-tuning.

4.3 Ablation Study

Impact of Module Count. We conducted experiments with varying module counts n, and adjusted the number of activated modules k accordingly to ensure consistent sparsity across all methods. As shown in Table 2a, increasing n generally improved the performance of experts. This improvement is likely attributed to the fact that a larger number of modules leads to more fine-grained decoupling of functionality and less activation of unnecessary parameters. We observe that the best performance after fine-tuning is achieved when $n = 16$. This may be attributed to the increased complexity introduced by a larger number of modules, which can limit the learning efficiency of model.

Impact of Expert Size. To investigate the effect of expert size on multi-task fine-tuning, we varied the number of modules selected per expert k_1. At deployment time, we kept the number of activated modules per task k_2 fixed to ensure consistent sparsity across all settings. As shown in Table 2b, the best performance is achieved when $k_1 = 16$. We suppose that small expert sizes limit the capacity to capture task-specific knowledge, while large expert sizes lead to negative interference between tasks. Future work may investigate how heterogeneous expert sizes affects the multi-task fine-tuning.

Effect of Partitioning and Routing Strategy. To investigate the impact of module partitioning, routing function, and multi-task fine-tuning, we compared our method against variants using random partitioning and Top-k routing. As shown in Table 3, experts built with co-activation-based partitioning and DSelect-k routing achieve the best performance, with an average improvement of approximately 10% over other configurations. Furthermore, multi-task fine-tuning brings an additional gain of 8.84% in average performance.

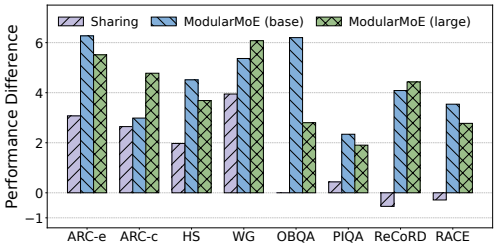


Figure 6: Performance gap of various configurations with respect to naive experts without fine-tuning.

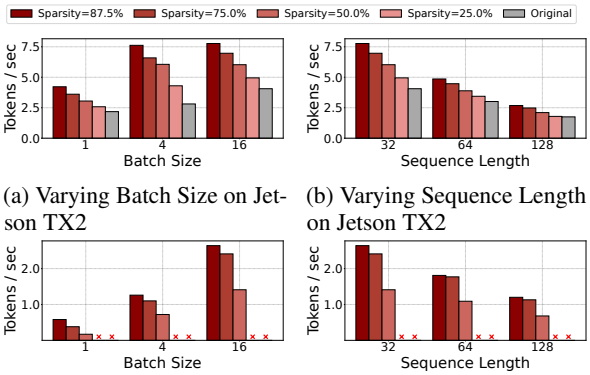
4.4 Further Analysis

Impact of Masking-Based MoE Implementation. As discussed in Section 3.4, the masking-based MoE implementation may result in the sharing of partial adapters across modules. In this section, we provide an in-depth analysis of its impact. For comparison, we implemented two baselines: (1) ***LoRA***: Each module is equipped with a dedicated adapter; (2) ***QLoRA*** (Dettmers et al., 2023): Similar to *LoRA*, except that the backbone model weights are quantized to 4-bit precision. For fairness, all models were trained for the same number of steps. As shown in Table 4, *LoRA* achieves the highest average performance, albeit with the longest training time. *QLoRA* benefits from quantization-induced memory savings, which enable larger batch sizes and help mitigate device underutilization. Our masking-based implementation achieves a 5.26× speedup compared to *LoRA* while retaining 98% of its average performance, demonstrating its effectiveness in accelerating fine-tuning of MoEs.

Experiments on Larger-Scale Datasets. To evaluate the scalability of ModularMoE, we extended the training set from 8 tasks to a larger collection of 26 tasks. We first clustered all samples into eight domains using a Gaussian Mixture Model (Aharoni and Goldberg, 2020). Then, we constructed expert for each domain using the same procedure as discussed in Section 3.3 and 3.4. For comparison, we implemented a dense baseline called ***Sharing***, where all tasks share a common set of adapters without any task-conditioned weight updates. As shown in Figure 6, ModularMoE maintains strong performance on the large-scale dataset, and even outperforms its small-scale variant on some tasks, demonstrating excellent stability and scalability. We attribute this to its strong ability to mitigate inter-task interference. This modular architecture enables compact and efficient storage of multi-domain knowledge within a single model.

Method	Training Time (h)	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
Expert	None	60.27	33.45	52.60	59.83	35.20	71.93	0.77/76.17	35.60	53.13
LoRA	9.31	67.97	37.29	**57.86**	65.59	**43.60**	**75.52**	**0.82/81.20**	**40.86**	**58.74**
QLoRA	5.23	**68.18**	**38.48**	57.40	**66.77**	41.60	75.08	0.82/81.06	40.29	58.61
Masking	**1.77**	66.54	36.43	57.11	65.19	41.40	74.27	0.81/80.26	39.14	57.54

Table 4: Training time and performance of different multi-task fine-tuning strategies on ReLU-LLaMA2-7B. "Expert" denotes naive experts without fine-tuning. All methods were trained for the same number of steps.



(a) Varying Batch Size on Jetson TX2

(b) Varying Sequence Length on Jetson TX2

(c) Varying Batch Size on Raspberry Pi 4B

(d) Varying Sequence Length on Raspberry Pi 4B

Figure 7: Throughput of original ReLU-LLaMA2-7B and its compressed variants by ModularMoE on Jetson TX2 and Raspberry Pi 4B. Figure (a), (c) report results under fixed sequence length = 32 with varying batch sizes, while Figure (b), (d) report results under fixed batch size = 16 with varying sequence lengths. Red crosses indicate settings that exceeded the memory.

Inference Efficiency on Real Hardware. To investigate the practical inference efficiency of models, we deployed the 4-bit quantized versions of the compressed models by ModularMoE on NVIDIA Jetson TX2 and Raspberry Pi 4B using llama.cpp[3]. We measured the end-to-end generation throughput (tokens per second) of the original model and its varying compressed counterparts across different batch sizes and sequence lengths. As demonstrated in Figure 7, ModularMoE consistently maintains generation speed improvements across all compression ratios. Notably, the performance gain becomes more pronounced with larger batch sizes and shorter sequence lengths, resulting in a maximum speedup of 1.53× under the 25% sparsity, 2.15× under the 50% sparsity, 2.35× under the 75% sparsity, and 2.71× under the 87.5% sparsity. These results highlight the effectiveness of ModularMoE in improving the inference efficiency of LLMs on real-world, resource-constrained hardware.

[3] https://github.com/ggml-org/llama.cpp/tree/b5556

5 Related Work

Jacobs et al. (1991) first introduced the concept of MoE, constructing a system with multiple independent sub-networks. Shazeer et al. (2016) combined MoE with LSTM to construct the first large-scale MoE language model. GShard (Lepikhin et al., 2020), Switch-Transformer (Fedus et al., 2022) further explored building large-scale Transformer-based MoEs. However, training MoEs from scratch requires significant computational resources and training time. In response, some works (Zhang et al., 2022; Zhu et al., 2024; Szatkowski et al., 2024) investigated converting the pre-trained model into its MoE version. However, their objective is to reduce the computational efficiency of the model rather than its memory overhead.

Beyond larger model size, expert specialization is also a crucial factor in enhancing the capability of MoEs. Branch-Train-Merge (Li et al., 2022), DEMix (Gururangan et al., 2022) explicitly fine-tuned experts in specific domains, ensuring each expert learns distinct knowledge. DeepSeekMoE (Dai et al., 2024) further formalized the issue in terms of *knowledge hybridity* and *redundancy*, and addressed it by implementing fine-grained expert segmentation and setting up shared experts. Our approach combines the strengths of existing works by constructing experts in a modular fashion. This design not only eliminates parameter redundancy across experts but also enhances their specialization through explicit multi-task fine-tuning.

6 Conclusions

In this paper, we propose ModularMoE, a training framework that converts pre-trained LLMs into parameter-sharing MoEs for lightweight deployment. It employs a modular architecture to pre-store multi-domain knowledge, thereby enabling rapid model customization for diverse low-resource scenarios. Extensive experiments on various models and tasks demonstrate that ModularMoE effectively accelerates model inference on edge devices while achieving superior task performance.

Limitations

Due to resource constraints, this study only conducted experiments on two models. Given that the phenomenon of emergent modularity has been observed across various LLMs, future studies could extend our experiments to a broader range of models. Although we investigated the impact of expert size on multi-task fine-tuning, in a single experiment the expert sizes across different domains were kept identical. The effects of heterogeneous experts with varying sizes remain to be explored. Moreover, we constructed experts tailored for specific downstream tasks and groups of tasks, which can be expanded to domains defined by language, topic, or modality in future work.

Acknowledgments

This work was supported in part by the National Key R&D Program of China 2024YFE0200800,the National Natural Science Foundation of China under Grants (62471055, U23B2001, 62401080, 62406039, 62321001, 62101064, 62171057, 62201072), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM107), the High-Quality Development Project of the MIIT (2440STCZB2584), the Ministry of Education and China Mobile Joint Fund (MCM20200202, MCM20180101), the 2025 Education and Teaching Reform Project Funding at Beijing University of Posts and Telecommunications (2025YZ005).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Roee Aharoni and Yoav Goldberg. 2020. Unsupervised domain clusters in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7747–7763.

Carliss Y Baldwin and Kim B Clark. 2000. *Design rules, Volume 1: The power of modularity*. MIT press.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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*.

Damai Dai, Chengqi Deng, Chenggang Zhao, Rx Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, and 1 others. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36:10088–10115.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*.

Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495.

Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, and 1 others. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A Smith, and Luke Zettlemoyer. 2022. Demix layers: Disentangling domains for modular language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5557–5576.

Hussein Hazimeh, Zhe Zhao, Aakanksha Chowdhery, Maheswaran Sathiamoorthy, Yihua Chen, Rahul Mazumder, Lichan Hong, and Ed Chi. 2021. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347.

Shwai He, Daize Dong, Liang Ding, and Ang Li. 2025. Towards efficient mixture of experts: A holistic study of compression techniques. *Transactions on Machine Learning Research*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794.

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. In *International Conference on Learning Representations*.

Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A Smith, and Luke Zettlemoyer. 2022. Branch-train-merge: Embarrassingly parallel training of expert language models. In *First Workshop on Interpolation Regularizers and Beyond at NeurIPS 2022*.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. 2024. Merge, then compress: Demystify efficient smoe with hints from its routing policy. In *The Twelfth International Conference on Learning Representations*.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, and 1 others. 2023. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *The Eleventh International Conference on Learning Representations*.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. 2024. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6159–6172.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in Neural Information Processing Systems*, 36:21702–21720.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391.

Seyed Iman Mirzadeh, Keivan Alizadeh-Vahid, Sachin Mehta, Carlo C del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. Relu strikes back: Exploiting activation sparsity in large language models. In *The Twelfth International Conference on Learning Representations*.

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.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2016. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Chenyang Song, Xu Han, Zhengyan Zhang, Shengding Hu, Xiyu Shi, Kuai Li, Chen Chen, Zhiyuan Liu, Guangli Li, Tao Yang, and 1 others. 2025. Prosparse: Introducing and enhancing intrinsic activation sparsity within large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 2626–2644.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. 2024. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. *Advances in Neural Information Processing Systems*, 37:43245–43273.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Cheng-Zhong Xu. 2024. Hydralora: An asymmetric lora architecture for efficient fine-tuning. *Advances in Neural Information Processing Systems*, 37:9565–9584.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Sheng Zhang, Xiaodong Liu, Jingjing Liu, Jianfeng Gao, Kevin Duh, and Benjamin Van Durme. 2018. Record: Bridging the gap between human and machine commonsense reading comprehension. *arXiv preprint arXiv:1810.12885*.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890.

Zhengyan Zhang, Zhiyuan Zeng, Yankai Lin, Chaojun Xiao, Xiaozhi Wang, Xu Han, Zhiyuan Liu, Ruobing Xie, Maosong Sun, and Jie Zhou. 2023. Emergent modularity in pre-trained transformers. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. 2024. Llama-moe: Building mixture-of-experts from llama with continual pre-training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15913–15923.

A Training and Hardware Details

In this section, we describe the experimental setup, hyperparameters, and baseline methods.

Testbed. The offline compression stage was performed on a machine equipped with an Intel i9-13900K processor, and two NVIDIA RTX 4090 GPUs. We deployed the compressed models on two representative edge devices: NVIDIA Jetson TX2 (GPU) and Raspberry Pi 4B (CPU). The software environment includes JetPack 4.6.6, Ubuntu 18.04.6 LTS, CUDA 10.2, Raspberry Pi OS, and llama.cpp (b5556).

Method	Sparsity	Params (B)	ARC-e	ARC-c	HS	WG	Average
Dense	0%	6.74	74.07	43.60	66.69	69.22	63.40
LLM-Pruner	25.0%	5.79	70.83	39.42	60.86	64.56	58.92
ModularMoE	25.0%	5.79	**74.33**	**45.73**	**64.49**	**70.80**	**63.84**
LLM-Pruner	50.0%	4.84	63.51	33.53	53.49	59.43	52.49
ModularMoE	50.0%	4.84	**66.54**	**36.43**	**57.11**	**65.19**	**56.32**
LLM-Pruner	75.0%	3.90	46.00	26.45	43.60	53.20	42.31
ModularMoE	75.0%	3.90	**58.46**	**33.28**	**47.83**	**59.12**	**49.67**
LLM-Pruner	87.5%	3.42	34.68	26.20	35.91	50.43	36.81
ModularMoE	87.5%	3.42	**48.86**	**27.13**	**41.80**	**54.62**	**43.10**

Table 5: Zero-shot performance of various compression methods under different sparsity levels on the ReLU-LLaMA2-7B model.

Sparsity	Params (B)	FLOPs (T)	MACs (G)	Memory (GiB)
0%	6.74	1.69	845.71	12.55
25.0%	5.79	1.45	724.51	10.79
50.0%	4.84	1.21	603.31	9.11
75.0%	3.90	0.96	482.11	7.30
87.5%	3.42	0.84	421.51	6.38

Table 6: Statistics of the compressed ReLU-LLaMA2-7B model under different sparsity levels.

Hyperparameters. We set the number of modules n to 32, and the number of activated modules k to 16. We employed the AdamW optimizer with its default configuration. The maximum learning rate is 10^{-3} for the training of the routing function, 3.0×10^{-4} for multi-task instruction fine-tuning, and the final learning rate decays to 3.0×10^{-5} with cosine scheduling. LLM-Pruner indicates that the first three and the last layers significantly impact the model performance, thus we did not modify these layers. The LoRA modules were configured with rank $= 8$, alpha $= 16$ and dropout $= 0.05$.

Comparisons. We compared ModularMoE with six baselines. (1) *Wanda* (Sun et al., 2024) prunes the model based on weight magnitudes and input activations. (2) *SparseGPT* (Frantar and Alistarh, 2023) proposes a one-shot pruning approach for LLMs. (3) *LLM-Pruner* (Ma et al., 2023) leverages first-order information and approximate Hessian information to prune the model. (4) *NAEE* (Lu et al., 2024) prunes experts by minimizing the token reconstruction loss. (5) *UMC* (He et al., 2025) prunes experts based on the outputs of the routing function. (6) *M-SMoE* (Li et al., 2024) merges similar experts by considering both expert similarity and activation frequency. For a fair comparison, we adapted these methods as task-specific semi-structured or structured pruning approaches tailored for FFNs.

Bits	Memory (GiB)	ARC-e	ARC-c	HS	WG	Average
16	9.11	66.54	36.43	57.11	65.19	56.32
8	4.86	66.41	36.35	57.14	65.04	56.24
4	2.71	65.74	36.35	57.14	64.48	55.93

Table 7: Impact of 4-bit and 8-bit quantization on the compressed ReLU-LLaMA2-7B model.

B The Performance under Varying Sparsity Levels

Since different devices vary significantly in memory capacity, the maximum model size they can accommodate also differs. To evaluate the robustness of different compression methods under varying memory constraints, we compared performance across four sparsity levels. Because semi-structured pruning methods such as Wanda and SparseGPT only support 50% sparsity, and expert-pruning methods for MoEs perform poorly, we conducted comparisons primarily against LLM-Pruner. Experiments are conducted on four tasks, and results are reported in Table 5. Our method consistently outperforms LLM-Pruner across all sparsity levels. Notably, at a sparsity level of 25.0%, the compressed model achieves performance nearly equivalent to the original dense model. We further evaluated the floating-point operations (FLOPs), multiply-add operations (MACs), and peak memory consumption of the deployed models under varying sparsity levels. The input sequence length was set to 128 tokens, and the model weights were loaded in FP16 precision. The results are summarized in Table 6.

C Integrate ModularMoE with Quantization

Quantization can maintain the comparable performance of the original models while significantly reducing memory costs. ModularMoE focuses on conditional activation of model parameters, making it compatible with quantization. Specifically, we first applied ModularMoE to generate a series of target models with 50% sparsity and subsequently quantized these models to 4-bit and 8-bit precision using GPTQ (Frantar et al., 2023). The baseline model weights are loaded in FP16 precision. As shown in Table 7, quantizing the model to 4-bit precision reduces memory usage by approximately 70% while retaining over 99% of the original performance. These results demonstrate that quantization is an effective approach for deploying resulting models on resource-constrained devices.

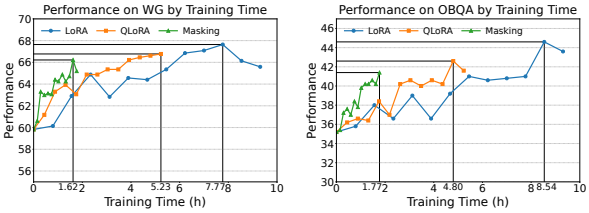


Figure 8: Task performance over training time under different training strategies.

D The Effects of ReLU*fication*

Sparse Activation, i.e., only a small subset of neurons is activated for a single input, has been widely observed in ReLU models (Li et al., 2023) and offers a promising paradigm for accelerating model inference. However, many LLMs (Touvron et al., 2023; GLM et al., 2024) adopt activation functions without intrinsic activation sparsity, such as GELU (Hendrycks and Gimpel, 2016) and SiLU (Elfwing et al., 2018). Several studies (Zhang et al., 2022; Mirzadeh et al., 2024; Song et al., 2025) have explored replacing the activation functions with ReLU to restore this sparsity. In this work, we conducted experiments based on these publicly released models and provided a detailed analysis of the effects of activation functions. As shown in Table 8, ReLU variants retain over 97% of the original performance.

E Ablation and Analysis Details

E.1 Impact of Varying Module Count

In Section 4.3, we present the average performance of ModularMoE across varying numbers of modules, while Table 9 provides detailed results for each individual task.

E.2 Impact of Varying Expert Size

In Section 4.3, we present the average performance of ModularMoE across varying size of experts, while Table 10 provides detailed results for each individual task.

E.3 Acceleration Effects of Masking-Based MoE Implementation

In Section 4.4, we present the training time and task performance of three strategies: **LoRA**, **QLoRA**, and **Masking**. For fairness, all strategies were trained for the same number of steps. To further analyze the training process, we illustrate the performance trends on the WG and OBQA tasks with respect to training time in Figure 8. While

Algorithm 1: Overall Process of ModularMoE

Input: Mixed dataset $\mathcal{D}^M$, task-specific dataset $\mathcal{D}^t$, loss function ℓ, learning rate η, neuron activation value a

Initialize co-activate graph $G = 0$, adapter parameters $A \sim \mathcal{N}(0, \sigma^2)$, $B = 0$;
for *each* $x \in \mathcal{D}^M$ **do**
  for *each neuron pair* (i, j) *from FFN* **do**
    $G[i, j] = G[i, j] + |a_i(x) \cdot a_j(x)|$; // Computing the co-activate graph
  end
end
for *each module* $i = 1, ..., n$ **do**
  $f_{m_i} \leftarrow \text{GRAPHSPLIT}(G)_i$; // Module partitioning
end
for *each task* $t = 1, ..., T$ **do**
  for *each* $(x, y) \in \mathcal{D}^t$ **do**
    $r^t \leftarrow r^t - \eta \cdot \nabla_{r^t} \ell(y, \sum_{i=1}^{n} f_{m_i}(x) r_i^t)$;
  end
  $M^t \leftarrow \mathbb{I}(r^t = 1)$; // Generate task-specific mask
end
for *each task* $t = 1, ..., T$ **do**
  for *each* $(x, y) \in \mathcal{D}^t$ **do**
    $A \leftarrow A - \eta \cdot \nabla_A \ell(y, FFN(x) \odot M^t)$; // Multi-task fine-tuning
    $B \leftarrow B - \eta \cdot \nabla_B \ell(y, FFN(x) \odot M^t)$;
  end
end

the peak performance shows a slight drop, Masking achieves about 3× speedup over QLoRA and about 5× speedup over LoRA during training. With training time comparable to standard single-task fine-tuning, our method attains performance approaching that of other multi-task training methods, achieving a desirable trade-off between performance and efficiency.

E.4 Numerical Results on Larger-Scale Datasets

In Figure 6, we illustrate the performance differences between various fine-tuning strategies and naive experts. Detailed numerical results for various configurations are provided in Table 11.

F Algorithm of ModularMoE

The full procedure of ModularMoE is summarized in Algorithm 1.

Model	Activation	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
LLaMA2-7B	SiLU	74.58	46.25	76.37	68.98	44.20	79.11	0.92/90.91	39.52	65.08
	ReLU	74.07	43.60	66.69	69.22	42.60	78.89	0.90/89.40	40.96	63.18
ChatGLM3-6B	SiLU	59.55	37.63	59.57	62.43	36.60	72.25	0.75/74.63	38.28	55.18
	ReLU	62.16	35.92	56.20	61.41	31.80	72.52	0.81/80.74	36.27	54.63

Table 8: Zero-shot performance of pre-trained models and their corresponding ReLU variants.

n	k	Method	Sparsity	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
8	4	Expert	50%	58.29	28.67	50.43	54.93	32.60	68.83	0.70/68.77	31.87	49.30
		ModularMoE	50%	66.04	36.86	56.69	64.72	38.60	74.21	0.82/81.94	38.57	57.20
16	8	Expert	50%	56.27	34.90	51.09	60.30	33.40	71.55	0.75/74.09	33.59	51.90
		ModularMoE	50%	**68.81**	**38.91**	56.91	**66.22**	40.40	**74.92**	**0.83/82.49**	**39.33**	**58.50**
32	16	Expert	50%	60.27	33.45	52.60	59.83	35.20	71.93	0.77/76.17	35.60	53.13
		ModularMoE	50%	66.54	36.43	**57.11**	65.19	**41.40**	74.27	0.81/80.26	39.14	57.54

Table 9: Zero-shot performance of ModularMoE with varying numbers of modules on the ReLU-LLaMA2-7B model. n denotes the total number of modules, and k indicates the number of selected modules.

n	k_1	k_2	Sparsity	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
32	4	16	50%	63.43	33.28	54.37	62.35	36.80	72.85	0.80/79.34	35.69	54.76
32	8	16	50%	65.11	35.32	55.80	65.90	38.20	72.85	0.80/79.63	37.61	56.30
32	16	16	50%	**66.54**	**36.43**	**57.11**	65.19	**41.40**	**74.27**	**0.81/80.26**	**39.14**	**57.54**
32	24	16	50%	64.94	36.01	56.94	**66.54**	39.00	72.63	0.79/78.34	37.51	56.49

Table 10: Zero-shot performance of ModularMoE with varying expert sizes on the ReLU-LLaMA2-7B model. k_1 denotes the number of modules activated per expert, which determines expert size, and k_2 indicates the number of modules activated at deployment, which determines the final model sparsity.

Method	ARC-e	ARC-c	HS	WG	OBQA	PIQA	ReCoRD	RACE	Average
Expert	60.27	33.45	52.60	59.83	35.20	71.93	0.77/76.17	35.60	53.13
Sharing	63.34	36.09	54.57	63.77	35.20	72.36	0.76/75.63	35.31	54.53
Masking (base)	**66.54**	36.43	**57.11**	65.19	**41.40**	**74.27**	0.81/80.26	**39.14**	**57.54**
Masking (large)	65.78	**38.23**	56.29	**65.90**	38.00	73.83	**0.81/80.60**	38.37	57.13

Table 11: Comparison on a large-scale dataset using the ReLU-LLaMA2-7B model. "Expert" denotes the performance without fine-tuning. "Sharing" refers to a dense baseline where all tasks share a common set of adapters.