DenseMixer: Improving MoE Post-Training with Precise Router Gradient

Anonymous Author(s)
Affiliation
Address
email

Abstract

Mixture-of-Experts (MoE) models are notoriously harder to train compared with dense models. Existing approaches either rely on imprecise router gradient or freeze router parameters entirely, limiting training effectiveness. We introduce DenseMixer, a novel MoE post-training technique that trades one extra forward pass on inactive experts for a more precise router gradient estimation. Our method consistently outperforms conventional methods across different MoE scales (7B, 14B, 30B), architectures (with/without shared experts), pre-training methods (from scratch/up-cycling), and post-training data types (instruction/long CoT data). It is universally applicable to any MoE using Top-K routing and can be used in a plug-and-play manner, compatible with existing training libraries and parameter-efficient methods like LoRA, introducing no changes to inference. We provide comprehensive empirical validation showing DenseMixer's effectiveness in improving MoE post-training quality while maintaining practical computational overhead.

1 Introduction

Mixture-of-Experts (MoE) models have emerged as a powerful paradigm for scaling neural networks while maintaining computational efficiency. By activating only a subset of experts for each input, MoE architectures achieve the capacity of dense models with significantly lower computational cost during inference. However, despite their promising potential, MoE models are notoriously harder to train compared with dense models [Fedus et al., 2022, Lepikhin et al., 2021].

The primary challenge lies in MoE's sparse routing mechanism—typically implemented via a Top-K router—which is mathematically non-differentiable [Shazeer et al., 2017]. This fundamental issue blocks straightforward back-propagation and significantly complicates gradient computation, particularly for the router parameters that determine expert selection. The non-differentiability problem forces existing training methods to rely on imprecise gradient approximations, where the problematic Top-K operation is treated as having zero gradient during automatic differentiation, or to freeze router parameters entirely during fine-tuning. [Unsloth AI, 2025]

To address the non-differentiability problem, we introduce DenseMixer, a novel MoE post-training technique that trades additional compute (on inactive experts during the forward pass) for more precise router gradient estimation. Specifically, DenseMixer employs straight-through estimators [Bengio et al., 2013] to provide better gradient approximations for the Top-K routing operation, requiring outputs from all experts during forward propagation while maintaining sparse computation during inference.

DenseMixer consistently outperforms conventional methods across different MoE scales (7B, 14B, 30B), architectures (with/without shared experts), pre-trained methods (from scratch/up-cycling), and post-training data types (instruction/long CoT data). The method is universally applicable to any MoE using Top-K routing and can be used in a plug-and-play manner, compatible with existing

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

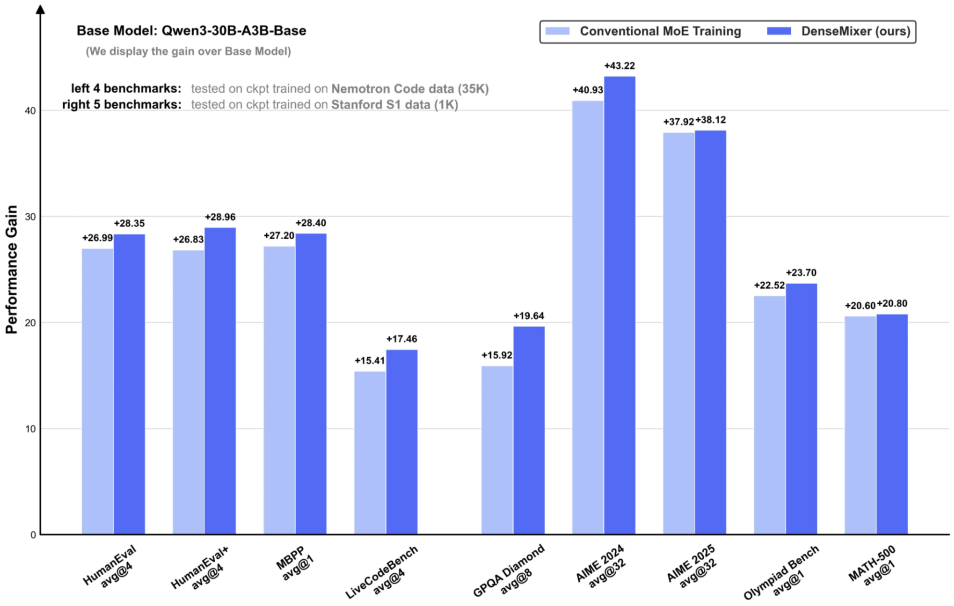

Figure 1: Overview of the DenseMixer framework. Performance gains of Qwen3-30B MoE after post-training using conventional method vs. DenseMixer. Results show consistent improvements across multiple evaluation metrics and decoding configurations.

training libraries (transformers, llama-factory, open-instruct, verl) and parameter-efficient methods (e.g., LoRA) [Hu et al., 2022].

Our comprehensive evaluation demonstrates DenseMixer's effectiveness across three representative MoE models: OLMoE-1B-7B [Muennighoff et al., 2025a], Qwen1.5-MoE-A2.7B, and Qwen3-30B-A3B-Base [Yang et al., 2025]. We compare against multiple baselines including conventional MoE training, Expert-Specialized Fine-Tuning (ESFT) [Wang et al., 2024], and frozen router approaches across diverse downstream tasks.

To summarize, our contributions are three-fold:

- We propose DenseMixer, a novel MoE post-training technique that provides a more precise router gradient estimation thus better performance through the straight-through estimator.
- We provide comprehensive empirical validation across multiple MoE scales, architectures, and training scenarios, demonstrating consistent improvements over conventional methods.
- We offer a plug-and-play implementation that requires minimal code changes and is compatible with existing training frameworks, enabling easy adoption for practitioners.

2 Background

In this section, we provide background on MoE training challenges, review existing approaches and their limitations, and formalize the problem that motivates our work.

2.1 MoE Training Challenges

Mixture-of-Experts (MoE) models replace traditional feed-forward networks with multiple expert networks and a router that selects which experts to activate for each input [Jacobs et al., 1991, Jordan and Jacobs, 1994, Shazeer et al., 2017]. While this architecture enables efficient scaling, it introduces fundamental training difficulties absent in dense models.

The core challenge stems from the non-differentiable Top-K routing mechanism. In MoE forward propagation, a router network computes routing weights $\pi \in \mathbb{R}^N$ for N experts, applies Top-K selection to activate only the highest-scoring experts, and computes the final output as a weighted combination of active expert outputs. However, the Top-K operation introduces a discrete selection

step that is mathematically non-differentiable, blocking standard gradient-based optimization for router parameters [Shazeer et al., 2017, Lepikhin et al., 2021, Fedus et al., 2022].

This non-differentiability problem manifests during backpropagation when computing gradients with respect to router parameters. The gradient computation requires the derivative of the Top-K selection mask, which is undefined at the boundaries where expert rankings change. This fundamental issue forces existing methods to rely on imprecise approximations or avoid training the router entirely.

2.2 Limitations of Existing Approaches

Current approaches to MoE training handle the non-differentiability problem through several paradigms, each with inherent limitations:

- **Conventional Training with Zero Gradients**. The most common approach treats the Top-K operation as having zero gradient during backpropagation, effectively ignoring the non-differentiable term. While simple to implement, this approximation provides imprecise gradient signals that can hinder training effectiveness, particularly for router parameters that determine expert selection quality [Lepikhin et al., 2021, Fedus et al., 2022].
- **Frozen Router Training**. Some practitioners freeze router parameters during fine-tuning to avoid the gradient computation issue entirely. [Unsloth AI, 2025] While this eliminates the non-differentiability problem, it prevents the router from adapting to new tasks or data distributions, potentially limiting model performance.
- **Expert-Specialized Fine-Tuning (ESFT)**. Recent work proposes selecting and updating only a subset of experts for specific tasks, reducing the routing complexity. However, expert selection strategies can be delicate and task-specific, requiring careful tuning and prior knowledge about expert specializations [Wang et al., 2024].

Additionally, existing methods in the literature that attempt to address MoE gradient computation face practical limitations:

- **Sampling-Based Methods**. Approaches like SparseMixer and GrinMoE use numerical methods with sampling over routing distributions. However, these methods become computationally challenging when many experts are activated (e.g., 8 or more), limiting their applicability to modern MoE architectures [Liu et al., 2023, 2024].
- **Dense Training Methods**. Some approaches like Default MoE employ dense computation during both forward and backward passes. While effective, they require significantly more computation and may not be practical for large-scale training scenarios [Pan et al., 2024].

2.3 Problem Formulation

Task Definition. We focus on MoE *post-training* scenarios where a pre-trained MoE model is further fine-tuned on task-specific data. Post-training represents a critical phase where the additional computational cost of our method is more acceptable compared to pre-training, while still providing significant performance benefits.

Evaluation Framework. Our evaluation addresses the limitations of existing approaches through comprehensive comparison across: (1) *multiple model scales*: from 7B to 30B parameters to assess scalability, (2) *diverse architectures*: models with and without shared experts to test generalizability, (3) *various pre-training strategies*: both from-scratch and up-cycling approaches, and (4) *different data types*: instruction tuning and long reasoning chain datasets to evaluate versatility.

3 Methodology

In this section, we present DenseMixer, our solution to address the non-differentiability problem in MoE training. We first provide a detailed mathematical analysis of the root problem (§ 3.1), then explain how conventional methods handle this issue (§ 3.2), and finally introduce our DenseMixer approach with straight-through estimators (§ 3.3).

3.1 The Root Problem: Non-Differentiable Routing

MoE Forward Propagation. In Transformer-based MoE models, the standard feed-forward network (FFN) layer is replaced with a Mixture-of-Experts (MoE) layer, consisting of a set of N parallel FFNs referred to as experts: $\{E_0(x), E_1(x), \ldots, E_{N-1}(x)\}$, and a lightweight router network that dynamically selects a subset of these experts to activate for each input x.

The forward propagation of MoE layer involves the following steps:

1. **Compute routing weights** — Compute a score vector $\pi \in \mathbb{R}^N$ via a linear mapping and apply a softmax to obtain a probability distribution:

$$\pi = \text{Router}(x; \theta) = \text{softmax}(x \cdot \theta^T) \tag{1}$$

where θ is the router parameter.

2. **Compute expert output** — Compute the weighted sum over the Top-K selected experts' output with the corresponding routing weights:

$$y = \sum_{i=0}^{N-1} \pi_i \cdot \text{TopK}(\pi)_i \cdot \text{Expert}_i(x) \tag{2}$$

where:

$$\text{TopK}(\pi)_i = \begin{cases} 1, & \text{if } \pi_i \text{ is among the top-k values of } \pi, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

MoE Backward Propagation — the non-differentiability problem. To backpropagate from the expert output y to router parameter θ, we compute:

$$\nabla_\theta y = \sum_{j=0}^{N-1} \frac{\partial y}{\partial \pi_j} \frac{\partial \pi_j}{\partial \theta} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \text{Expert}_i(x) \cdot \frac{\partial(\pi_i \cdot \text{TopK}(\pi)_i)}{\partial \pi_j} \cdot \frac{\partial \pi_j}{\partial \theta} \tag{4}$$

Here: 1. The last term is straightforward to compute:

$$\frac{\partial \pi_j}{\partial \theta} = \frac{\partial \, \text{softmax}(x \cdot \theta^T)_j}{\partial \theta} \tag{5}$$

2. The middle term expands to:

$$\frac{\partial(\pi_i \cdot \text{TopK}(\pi)_i)}{\partial \pi_j} = \pi_i \cdot \frac{\partial \text{TopK}(\pi)_i}{\partial \pi_j} + \text{TopK}(\pi)_i \cdot \delta_{ij} \tag{6}$$

where:

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases} \tag{7}$$

Clearly, the Top-K mask is non-differentiable — specifically, the term $\frac{\partial \text{TopK}(\pi)_i}{\partial \pi_j}$ is not defined. Therefore, the router's gradient is not straightforward to compute.

3.2 How Conventional Method Handles This

Conventional MoE training sidesteps this issue by treating $\text{TopK}(\pi)$ as constant during backpropagation, thus neglecting the non-differentiable term $\frac{\partial \text{TopK}(\pi)_i}{\partial \pi_j}$, namely:

$$\frac{\partial \text{TopK}(\pi)_i}{\partial \pi_j} \approx 0 \tag{8}$$

Thus:

$$\frac{\partial(\pi_i \, \text{TopK}(\pi)_i)}{\partial \pi_j} \approx \text{TopK}(\pi)_i \cdot \delta_{ij} \tag{9}$$

And finally:

$$\nabla_\theta y \approx \nabla_{\text{Conventional}} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \text{Expert}_i(x) \cdot \text{TopK}(\pi)_i \cdot \delta_{ij} \cdot \frac{\partial \pi_j}{\partial \theta} \tag{10}$$

$$= \sum_{j=0}^{N-1} \text{Expert}_j(x) \cdot \text{TopK}(\pi)_j \cdot \frac{\partial \pi_j}{\partial \theta} \tag{11}$$

3.3 DenseMixer: Trade Compute for Gradient Precision

DenseMixer's Solution. To make the gradient approximation more accurate, DenseMixer adopts the **straight-through estimator** (STE) [Bengio et al., 2013]. This amounts to a first-order approximation where:

$$\frac{\partial \text{TopK}(\pi)_i}{\partial \pi_j} \approx \delta_{ij} \tag{12}$$

This means:

$$\frac{\partial(\pi_i \ \text{TopK}(\pi)_i)}{\partial \pi_j} \approx (\pi_i + \text{TopK}(\pi)_i) \cdot \delta_{ij} \tag{13}$$

Finally:

$$\nabla_\theta y \approx \nabla_{\text{DenseMixer}} = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} \text{Expert}_i(x) \cdot (\pi_i + \text{TopK}(\pi)_i) \cdot \delta_{ij} \cdot \frac{\partial \pi_j}{\partial \theta} \tag{14}$$

$$= \sum_{j=0}^{N-1} \text{Expert}_j(x) \cdot (\pi_j + \text{TopK}(\pi)_j) \cdot \frac{\partial \pi_j}{\partial \theta} \tag{15}$$

Intuitively, STE pretends that the Top-K selection is an identity function with respect to the router logits during backpropagation. In practice, we implement this by: 1. Forward — computing with the original hard Top-K as before; 2. Backward — overriding the gradient of the Top-K node to be the identity.

The Side Effect of DenseMixer. An obvious drawback of using DenseMixer's gradient approximation ($\nabla_{\text{DenseMixer}}$) is that it requires the outputs of all experts (i.e., $\text{Expert}_j(x)$ for all $j \in [0, N-1]$) for a given input x, while conventional method's gradient approximation ($\nabla_{\text{Conventional}}$) only requires those of the Top-K selected experts. This means that DenseMixer requires the MoE layer to be densely activated during forward pass, which introduces more compute than the normal case.

However, for MoE post-training, where the compute cost is not as critical as in pre-training, we can employ DenseMixer to trade compute for performance. In practice, DenseMixer only requires one extra forward pass on those inactive experts, which is different from setting the Top-K value to the total experts number.

Handling Normalized Top-K. Some recently released MoE models [Yang et al., 2025] adopt `normalized_topk_prob` implementation, which normalizes the expert weights as follows:

$$y_{\text{normalized}} = \sum_{i=0}^{N-1} \frac{\pi_i \cdot \text{TopK}(\pi)_i}{\sum_{k=0}^{N-1} \pi_k \cdot \text{TopK}(\pi)_k} \cdot \text{Expert}_i(x) \tag{16}$$

Such normalization makes the gradient computation even more complicated as both the numerator and denominator have the non-differentiable TopK term. We address this by distinguishing the TopK and non-TopK experts' output during backpropagation, providing a practical solution for modern MoE architectures.

4 Experiments

In this section, we verify the effectiveness of DenseMixer across multiple MoE models and training scenarios. We first introduce the experimental setup (§ 4.1), then present comparative results (§ 4.2) and ablation studies (§ 4.3), and conclude with efficiency analysis (§ 4.4).

4.1 Experimental Setup

We evaluate DenseMixer across three representative MoE models that cover a broad range of scales, architectures, and pre-training strategies. Each model is evaluated on downstream tasks relevant to its training domain.

Models and Datasets. We select the following three base models for conducting post-training experiments:

Table 1: Summary of MoE models used in our experiments.

Model Name	Active Param.	Total Param.	Active/Total Expert Num.	Shared Expert Num.	Context Length	Normalize TopK Prob	Training Strategy
OLMoE-1B-7B	1B	7B	8 / 64	0	4k	False	from scratch
Qwen1.5-MoE-A2.7B	2.7B	14B	8 / 64	4	8k	False	up-cycling
Qwen3-30B-A3B-Base	3B	30B	8 / 128	0	32k	True	from scratch

For OLMoE-1B-7B and Qwen1.5-MoE-A2.7B, we adopt the training and testing datasets from DeepseekAI's ESFT paper, which include diverse tasks such as GSM (math) [Cobbe et al., 2021], MBPP [Austin et al., 2021]and HumanEval (coding) [Chen et al., 2021], Intent classification, Law, Summary, and Translation [Wang et al., 2024].

For Qwen3-30B-A3B-Base, we found that training it with the above data hardly improves or even degrades its downstream performance. Therefore, we train it on long reasoning data and test it on challenging math/coding benchmarks:

- **Math domain**: We post-train on a filtered subset of the Stanford S1 dataset [Muennighoff et al., 2025b], which has around 1K training samples with reasoning trajectories distilled from Deepseek-R1 [DeepSeek-AI, 2025].
- **Coding domain**: We post-train on a filtered subset of Llama-Nemotron-Post-Training-Dataset [Nathawani et al., 2025], containing around 35K training samples.

Baselines. We compare DenseMixer against several state-of-the-art MoE training methods:

- **Conventional**: Standard MoE training where the non-differentiable Top-K operation is treated as having zero gradient in automatic differentiation.
- **ESFT**: Expert-Specialized Fine-Tuning proposed by DeepseekAI, which updates only a pre-selected subset of experts for a given task. We include both ESFT-gate (selection based on average gating scores) and ESFT-token (selection based on token-selection ratios).
- **Frozen Router**: Freeze the router and only update other components in MoE, as suggested by practitioners for stability.

We also compare these methods in parameter-efficient fine-tuning (PEFT) settings, where we apply LoRA to all modules except the router.

Evaluation. For each method, we conduct a grid search on training hyperparameters (learning rate and batch size) and report the best performance. We evaluate models across multiple downstream tasks relevant to each model's domain, measuring task-specific metrics such as accuracy for classification tasks and pass rates for coding tasks.

4.2 Main Results

Here we present comprehensive experimental results across all three MoE models. Tables 2, 3 show the results on Qwen3-30B-A3B-Base for math and coding benchmarks respectively, while Tables 7,

8 in the appendix show the results on OLMoE-1B-7B and Qwen1.5-MoE-A2.7B respectively. These results demonstrate DenseMixer's consistent improvements over baseline methods.

Table 2: Results on Qwen3-30B-A3B-Base across different decoding configurations (Math Benchmarks). DenseMixer consistently outperforms conventional training.

Method	Temp & Top-p	GPQA	AIME 2024	AIME 2025	Olympiad	MATH-500
Base Model	0.6 & 0.95	38.88	20.63	7.71	34.81	72.80
	0.7 & 0.8	39.89	20.53	8.33	33.92	75.40
	1.0 & 0.7	36.36	18.75	8.75	31.70	68.00
Conventional	0.6 & 0.95	54.80	61.56	45.63	57.33	93.40
	0.7 & 0.8	54.23	61.67	44.27	55.41	92.20
	1.0 & 0.7	56.55	63.65	46.15	59.11	93.00
DenseMixer	**0.6 & 0.95**	**58.52**	**63.85**	**45.83**	**58.51**	**93.60**
	0.7 & 0.8	**55.80**	**63.13**	**45.31**	**57.18**	**93.00**
	1.0 & 0.7	**58.14**	**62.71**	**47.50**	**57.77**	**93.80**

Table 3: Results on Qwen3-30B-A3B-Base across different decoding configurations (Code Benchmarks and Average). DenseMixer consistently outperforms conventional training.

Method	Temp & Top-p	HumanEval	HumanEval+	MBPP	LiveCodeBench	Avg
Base Model	0.6 & 0.95	65.24	60.06	53.60	16.85	48.94
	0.7 & 0.8	62.80	61.12	55.60	16.48	49.00
	1.0 & 0.7	62.65	59.14	49.80	13.26	46.21
Conventional	0.6 & 0.95	92.23	86.89	80.80	32.26	67.21
	0.7 & 0.8	91.01	85.37	76.80	29.39	65.59
	1.0 & 0.7	90.85	86.59	79.00	33.42	67.59
DenseMixer	**0.6 & 0.95**	**93.59**	**89.02**	**82.00**	**34.31**	**68.80**
	0.7 & 0.8	**91.92**	**86.89**	**80.80**	**31.89**	**67.32**
	1.0 & 0.7	**93.29**	**88.87**	**84.39**	**34.40**	**68.99**

4.3 Impact of Straight-Through Estimator.

To validate the effectiveness of our straight-through estimator approach, we compare DenseMixer with a variant that computes dense activations but uses conventional gradient approximation. The results show that the improved gradient estimation is crucial for performance gains.

Table 4: Ablation study on OLMoE-1B-7B showing the importance of precise gradient estimation.

Method	Avg Performance	Improvement
Conventional	35.44	-
Dense Forward + Conv. Gradient	36.12	+0.68
DenseMixer (Full)	**38.35**	**+2.91**

4.4 Efficiency Analysis

Computational Overhead. Though DenseMixer requires extra computuation on inactive experts, the time consumption does not grow linearly with the number of experts. It only requires an extra forward pass and does not require gradient computation and backward parameter update on inactive experts.

For Qwen3-30B-A3B, DenseMixer incurs approximately 1.46× the FLOPs compared to conventional training. The detailed FLOP analysis shows:

Training Time Comparison. We provide actual training time comparisons across different datasets:

Table 5: FLOP analysis for Qwen3-30B-A3B showing manageable computational overhead.

Method	Forward TFLOPs/layer	Total TFLOPs/layer
Conventional	16.85	50.54
DenseMixer	40.04	73.74
Ratio	**2.37×**	**1.46×**

Table 6: Training time comparison showing acceptable overhead for post-training scenarios.

Model	Dataset	Conventional	DenseMixer	Overhead
Qwen1.5-MoE (14B)	Intent (7K)	22 min	24 min	9%
	Law (1K)	8.5 min	9.5 min	12%
	Summary (19K)	1.2 h	1.4 h	17%
	Translation (11K)	39 min	45 min	15%
Qwen3-MoE (30B)	S1 (1K)	2.8 h	3.6 h	29%
	Nemotron-Code (35K)	21 h	28 h	33%

The results demonstrate that DenseMixer consistently outperforms conventional MoE training methods across different model scales, architectures, and training scenarios, while maintaining reasonable computational overhead suitable for post-training applications.

5 Conclusion

We confirm that the non-differentiability of Top-K routing is a key obstacle to achieving more effective MoE training. By trading an extra forward pass on inactive experts, **DenseMixer** enables more precise routing gradients, consistently improving MoE post-training quality as measured by downstream performance, beyond conventional MoE training approaches.

Our comprehensive experimental evaluation across three MoE models (7B, 14B, 30B parameters) and diverse architectures demonstrates DenseMixer's effectiveness and generalizability. The method achieves consistent improvements across different pre-training strategies (from scratch vs. up-cycling), various downstream tasks (math, coding, language understanding), and both full fine-tuning and parameter-efficient settings.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL `https://arxiv.org/abs/2108.07732`.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. URL `http://arxiv.org/abs/1308.3432`.

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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL `https://arxiv.org/abs/2107.03374`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

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

Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. Poster.

Liyuan Liu, Jianfeng Gao, and Weizhu Chen. Sparsemixer: Mixture-of-experts via learned sparse backpropagation. *arXiv preprint arXiv:2310.00811*, 2023. URL https://arxiv.org/abs/2310.00811.

Liyuan Liu et al. Grinmoe: Gradient-informed mixture-of-experts. *arXiv preprint arXiv:2409.12136*, 2024. URL https://arxiv.org/abs/2409.12136.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Øyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations (ICLR)*, 2025a. URL https://openreview.net/forum?id=xXTkbTBmqq.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025b.

Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, 2025. URL https://huggingface.co/datasets/nvidia/Nemotron-Post-Training-Dataset-v1.

X. Pan et al. Dense backpropagation improves routing for sparsely activated mixture-of-experts. *arXiv preprint arXiv:2402.01652*, 2024. URL https://arxiv.org/abs/2402.01652.

Noam Shazeer, Amin Mirhoseini, Niki Maziarz, Andy Davis, Quoc V. Le, and Geoffrey Hinton. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=rJBT6GZRb. Workshop Track.

Unsloth AI. Qwen3: How to run & fine-tune, 2025. URL https://docs.unsloth.ai/basics/qwen3-how-to-run-and-fine-tune#qwen3-moe-models-fine-tuning. Section "Qwen3 MOE models fine-tuning" states router layer fine-tuning is disabled by default.

Zihan Wang, Deli Chen, Damai Dai, Runxin Xu, Zhuoshu Li, and Yu Wu. Expert-specialized fine-tuning for sparse architectural large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. URL https://aclanthology.org/2024.emnlp-main.46/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao

Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. Technical report, arXiv, 2025. URL https://arxiv.org/abs/2505.09388.

A Appendix

B More experiment results

Method	GSM	MBPP	HumanEval	Intent	Law	Summary	Translation	Avg
Base Model	15.85	19.80	10.97	0.20	5.70	7.40	11.09	10.14
Frozen Router	44.88	17.80	7.23	72.80	22.50	36.05	28.29	32.79
Conventional	45.94	23.40	18.92	74.60	22.35	35.99	26.89	35.44
DenseMixer	**49.00**	**25.12**	**20.73**	**77.40**	**23.02**	**40.64**	**32.55**	**38.35**
Frozen Router + LoRA	45.03	24.20	17.07	55.80	21.30	37.70	28.19	32.76
Conventional + LoRA	44.58	24.20	15.85	60.20	21.60	37.30	26.22	32.85
DenseMixer + LoRA	**45.38**	**26.20**	**16.48**	**66.60**	**24.70**	**40.80**	**29.43**	**35.66**
ESFT-gate	43.06	20.80	14.02	21.20	22.39	19.50	17.37	22.62
ESFT-token	43.82	19.60	12.80	20.80	22.60	17.80	16.67	22.01

Table 7: Results on OLMoE-1B-7B (Total Parameters 7B). DenseMixer consistently outperforms all baselines.

Method	GSM	MBPP	HumanEval	Intent	Law	Summary	Translation	Avg
Base Model	38.69	38.84	32.31	16.83	18.20	28.29	16.53	27.10
Frozen Router	53.37	35.20	37.10	82.20	33.01	38.29	32.75	44.56
Conventional	53.42	34.60	36.43	81.80	29.25	37.80	33.02	43.76
DenseMixer	**55.16**	**35.40**	**39.68**	**83.40**	**33.83**	**40.56**	**33.90**	**45.99**
Frozen Router + LoRA	46.77	31.40	36.58	71.00	30.30	30.19	28.08	39.19
Conventional + LoRA	43.89	34.00	38.41	64.80	28.80	37.99	26.14	39.15
DenseMixer + LoRA	**47.24**	**35.40**	**38.41**	**71.80**	**31.80**	**40.20**	**29.25**	**42.01**
ESFT-gate	50.72	34.00	36.59	76.40	27.10	35.89	28.49	41.31
ESFT-token	52.76	35.80	37.20	76.00	28.20	33.39	28.86	41.74

Table 8: Results on Qwen1.5-MoE-A2.7B (Total Parameters 14B). DenseMixer achieves the best performance across all metrics.