LaDiMo: Layer-wise Distillation Inspired MoEfier

Anonymous ACL submission

Abstract

Increasing complexity of large language models has led to substantial training costs, resource demands, and environmental impacts. In response, sparse Mixture-of-Experts (MoE) models have emerged as a promising alternative to dense models. Since training MoE models from scratch can be prohibitively expensive, recent studies have explored leveraging knowledge from pre-trained non-MoE models. However, existing approaches have limitations, such as requiring significant hardware resources and data. We propose a novel algorithm, LaDiMo, which efficiently converts a Transformer-based non-MoE model into an MoE model with minimal additional training cost. LaDiMo consists of two stages: layer-wise expert construction and routing policy decision. By harnessing the concept of knowledge distillation, we compress the model and rapidly recover its performance. Furthermore, we develop an adaptive router that optimizes inference efficiency by profiling the distribution of routing weights and determining a layer-wise policy. We validated the effectiveness of our method by converting the LLaMA 3 8B model to an MoE model with just 2M tokens, reducing the number of activated parameters by 20% and FLOPs by 21% while maintaining accuracy. Our approach offers a flexible and efficient solution for building and deploying MoE models.

1 Introduction

The rise of Large Language Models (LLMs) has transformed the natural language processing (NLP) landscape, with their immense capacity to capture complex patterns and relationships in human language. However, this surge in model scale has also led to a concomitant increase in training costs, serving resources, and environmental footprints. In response, the sparse Mixture-of-Experts (MoE) model has recently garnered significant attention as an alternative to dense models (Cai et al., 2024). The Feed-Forward Networks (FFNs) in Transformers are replaced by a set of experts, where only a subset of these experts are activated for each input token, thereby achieving computational efficiency (Shazeer et al., 2017). For example, Mixtral 8x7B model (Jiang et al., 2024), a prominent large-scale model adopting the MoE architecture, reduces the number of active parameters by forwarding each input token only through the top two most relevant experts out of eight.

Since training an MoE model from scratch can be prohibitively expensive, recent studies have focused on leveraging knowledge from pre-trained non-MoE models. Moeficiation (Zhang et al., 2021) constructs experts based on neuron co-activation patterns, successfully converting T5 (Raffel et al., 2020) and BERT (Devlin et al., 2018) models, which use ReLU activation (Nair and Hinton, 2010) into MoE models. However, this approach is limited in that it is challenging to apply to recent state-of-the-art models that employ different activation functions, such as SwiGLU (Shazeer, 2020). Meanwhile, Zhu et al. (2024) proposed a method for transforming modern models like LLaMA (Touvron et al., 2023) into MoE models, but it requires relatively large hardware resources and data for training.

Furthermore, in many MoE-based models including Mixtral 8x7B, each token passed through the router is forwarded to a fixed number of experts. Nevertheless, since tokens exhibit varying levels of uncertainty, routing all tokens to the same number of experts at every layer can be inefficient (Wu et al., 2024; Huang et al., 2024). Researchers have recently explored dynamic routing approaches that enable tokens to be routed to multiple experts adaptively, thus enhancing performance and optimizing efficiency (Li et al., 2023; Huang et al., 2024; Zeng et al., 2024; Lu et al., 2024). While these existing methods require some level of training or fine-tuning, to our knowledge, no proposal

has been made for adaptive routing strategies that do not necessitate additional training.

To address the aforementioned issues, we introduce LaDiMo, a novel algorithm that constructs an MoE model, namely *MoEfy*, from a Transformer-based non-MoE model at a minimal additional training cost. LaDiMo consists of two stages: expert construction and routing policy decision. First, we construct an MoE model by leveraging the concept of knowledge distillation (Hinton et al., 2015), which utilizes the softmax output of a pre-trained model to train a more compact model. As shown in Figure 1, through layer-wise distillation, where each expert learns to approximate the original layer's results, we achieve efficient model compression and rapid performance recovery. Subsequently, to optimize inference efficiency, we deploy an adaptive router. By profiling the distribution of routing weights computed by the router for input tokens, we determine a layer-wise policy that minimizes accuracy degradation and reduces inference latency. The contributions of this study can be summarized as follows:

- **Conversion to MoE with fast training, small data**: LaDiMo accelerates the transformation of a non-MoE model into an MoE model using knowledge distillation-based training. Applying our methodology to the LLaMA 3 8B model, we successfully converted 12 layers into MoE layers using only 2M tokens. Our model achieved an MMLU (Hendrycks et al., 2020) accuracy of over 98% compared to the original model and even surpassed its TruthfulQA (Lin et al., 2021) scores, while reducing the number of activated parameters by 20% and FLOPs by 21%, resulting in significant computational cost savings.

- **Layer-wise Model Optimization**: By selectively training and converting less influential layers of the original model into MoE layers, we preserve the original model's capability. After training, we set the layer-wise routing policy based on the behavior of each MoE layer, enabling additional throughput improvements.

2 Backgrounds

2.1 Mixture-of-Experts

The concept of MoE, wherein certain components of a model specialize in distinct tasks or knowledge domains, was initially introduced by Jacobs et al. (1991). With the increasing scale of deep learning models, the sparse MoE was proposed aiming to reduce computational costs by activating only a subset of experts (Shazeer et al., 2017). Following this, the incorporation of MoE into Transformer-based LLMs has yielded impressive performance gains (Lepikhin et al., 2020; Fedus et al., 2022), leading to diverse research endeavors in this area. Notably, several industrial-scale LLMs incorporating MoE architectures, including Mixtral 8x7B, DeepSeek-V2 (DeepSeek-AI, 2024), DBRX (Databricks, 2024), Grok-1 (xAI, 2024), and Skywork-MoE (Wei et al., 2024), have been released.

The most prevalent architecture for integrating MoE into Transformer-based models involves substituting the FFN within each Transformer block with a parallel N FFNs $\{E_1, E_2, ..., E_N\}$, each constituting an individual expert, accompanied by a gating network, namely router. Specifically, for each input token x, the embedding vector is fed into the router R, determining which experts to forward it to. A significant reduction in the computational cost of the FFN in dense models is achieved by activating a few experts.

2.2 Knowledge Distillation

Knowledge distillation is a prominent approach for compressing cumbersome pre-trained models into more compact and rapid models by leveraging their knowledge. The class probabilities generated by a teacher model with a large parameter set are utilized as soft targets, enabling a smaller student model to learn from these outputs. By employing high-entropy soft targets, more knowledge can be distilled compared to exploiting hard targets.

As the complexity and scale of models increase, researchers have studied utilizing not only the output of the last layer but also intermediate representations from hidden layers (Romero et al., 2014). Recently, studies on layer-wise distillation have been gaining traction, particularly with Transformer-based LLMs. For instance, Sun et al. (2019) selectively leveraged hidden layers from a teacher model to fine-tune a smaller model for NLP tasks. Similarly, TinyBERT (Jiao et al., 2019) adopted attention-based distillation and embedding layer-based distillation to reduce the computational requirements of the BERT model. Furthermore, Liang et al. (2023) proposed a layer-wise distillation method that calculates the discrepancy be-

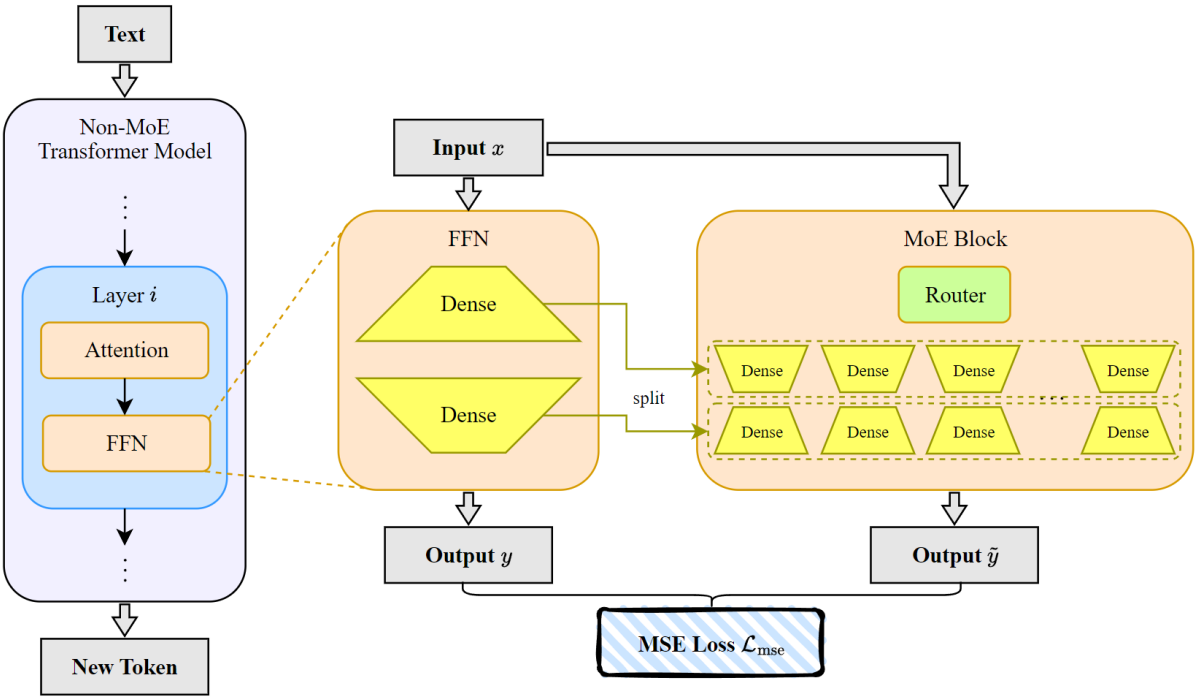

Figure 1: The main framework of LaDiMo. The MoE block consists of a gating router and experts, which are FFNs whose weights are initialized by splitting the reference FFN's weight matrices. Input x is obtained during inference tasks on a small text dataset. The collected inputs are then used as the dataset for training the MoE block. Furthermore, we apply auxiliary loss and adaptive router, which will be explained in Sections 4.3 and 4.4.

tween teacher model layers and student model layers using mean squared error (MSE) loss.

3 Related Works

3.1 From Dense to Sparse MoE Model

When building an MoE model from the dense Transformer-based models, it is essential to determine which network components (*e.g.*, FFNs, attention layers) to be replaced with experts and how (*e.g.*, the total number of MoE layers, the number of experts per MoE layer) (Cai et al., 2024). While some researches have been conducted on converting attention layers into MoE structures (Zhang et al., 2022; Shen et al., 2024), most studies have focused on converting FFNs. This is because FFNs account for a significant proportion of the overall FLOPs, and traditional ReLU-based models exhibit high activation sparsity in FFNs (Zhang et al., 2021; Li et al., 2022; Liu et al., 2023; Zheng et al., 2024; Pan et al., 2024). Recent studies have explored converting models employing soft activation functions with relatively low activation sparsity (*e.g.*, LLaMA's SwiGLU into MoE models (Zhu et al., 2024; Zheng et al., 2024)).

Determining the number of layers to be replaced with MoE layers, the number of experts per MoE layer, and the size of each expert is also crucial for designing MoE models. These hyperparameters directly impact the model's performance, including accuracy and system overheads such as memory requirements (Yun et al., 2024; Krajewski et al., 2024; Cai et al., 2024). For instance, MoE layers can replace either the entire model layers (Fedus et al., 2022; Jiang et al., 2024; Dai et al., 2024) or only specific layers (Lepikhin et al., 2020; Zoph et al., 2022), and the position of the replaced layers can also affect performance (see Section 5.2).

Various approaches have been developed to construct MoE models by leveraging pre-trained weights from dense checkpoints. Komatsuzaki et al. (2022) constructed an MoE model whose initial experts are copied from the original dense model's FFNs. Building upon this concept, Wei et al. (2024) empirically demonstrated that exploiting the original dense model's weights is more efficient when the budget for training an MoE model is limited. Zhang et al. (2021) clustered and partitioned intermediate FFNs based on co-activation patterns to construct experts. Inspired by this, Zheng et al. (2024) proposed a method to learn non-ReLU activation models with an MoE structure efficiently. Szatkowski et al. (2024) presented a method for in-

dividual regression-based training of routers. Zuo et al. (2022) built experts based on FFN neurons according to their importance scores and performed layer-wise knowledge distillation from BERT models. Furthermore, Zhu et al. (2024) suggested dividing original FFNs of SwiGLU-based models into multiple experts, although this approach requires relatively high costs for continued training. Considering these aspects, we propose a methodology that constructs inference-efficient sparse MoE models through layer-wise distillation while keeping training costs low.

3.2 Expert Choice Strategies

The efficiency of MoE models relies on the activation of only a subset of experts, and the expert selection strategy has a significant impact on model performance. Generally, for a given input i-th x_i, the output of the router R can be represented as follows:

$$R(x_i) = (r_{i1}, \ldots, r_{iN}) \tag{1}$$

where r_{ij} denotes the probability of assigning the i-th token to expert j, also known as the routing weight (Shazeer et al., 2017). Conventionally, a static value k is set to be smaller than N, and the top-k experts with the highest routing weights are selected (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Wei et al., 2024), such as in the case of the Mixtral 8x7B model where $k = 2$ (Jiang et al., 2024).

Intuitively, if the maximum routing weight is sufficiently large, selecting only a single expert may have a negligible impact on the result while reducing computational costs. Conversely, if the distribution of routing weights is uniform, choosing multiple experts may benefit model accuracy (See Section 4.4). This implies that each token does not necessarily need to be forwarded to an equal number of experts, and recent studies have proposed dynamic routing strategies. Li et al. (2023) optimized the training process by routing to either the top-1 or top-2 experts based on the weight difference between the highest expert and the second-highest one. Similarly, Lu et al. (2024) and Zhong et al. (2024) chose only the top expert if the ratio of the highest weight exceeds a certain threshold. Wu et al. (2024) trained all experts during fine-tuning for tokens with uniform routing weight distributions and maintained the conventional approach at inference time, thereby improving model performance without incurring additional computational costs. Researches that select multiple experts until the sum of their routing weights exceeds a certain threshold have also been conducted (Huang et al., 2024; Yang et al., 2024).

Besides optimizing routing strategies, efforts have been made to improve the architecture of experts. Li et al. (2023) achieved the top-all effect by merging experts according to their routing weight ratios, leading to improved model accuracy. Zeng et al. (2024) boosted efficiency by introducing a FLOP-free null expert set and increasing the top-k. However, most existing studies suffer from the limitation of requiring additional training, whereas our proposed method determines the layer-wise routing policy in a training-free manner.

4 Methodology

LaDiMo focuses on mimicking a given non-MoE model by constructing an MoE model and training to approximate the output using a limited text dataset and computing resources. Under such limited environment, training the entire layers with their FFNs replaced with MoE blocks as in previous approaches may degrade the performance due to the underfitted result (Zhang et al., 2021; Zhu et al., 2024; Komatsuzaki et al., 2022), which motivated us to substitute FFNs to MoE blocks for only some layers. To do so, we needed to consider how many and which layers to be replaced. FLOPs decreased as the number of chosen layers increases, which results in better throughputs, while the accuracy also decreased under restricted training resources (See 5.2). This trade-off has to be treated more carefully. By training some MoE blocks independently and assembling them, one can find an optimal composition of original layers from the reference model and newly trained layers.

4.1 Continued Pre-training

Given a layer from the Transformer-based non-MoE model, the FFN in the layer consists of two dense projections. Some models like LLaMA use SwiGLU as their activation function, whose FFN contains three dense projections. The FFN transforms an input $x \in \mathbb{R}^{d_h}$ into

$$(\mathbf{x}W_u \odot \text{Swish}(\mathbf{x}W_g))\, W_d, \tag{2}$$

where $W_u \in \mathbb{R}^{d_h \times d_i}, W_g \in \mathbb{R}^{d_h \times d_i}, W_d \in \mathbb{R}^{d_i \times d_h}$ denote the up, gate, and down projection weights respectively, d_h, d_i denote the dimension of hidden and intermediate state vector respectively, and $\odot$ denotes element-wise product.

To construct an MoE block, LaDiMo starts with splitting those weight matrices into N submatrices where N is the number of experts. The resulting submatrices are $W_u^{(i)} \in \mathbb{R}^{d_h \times d_i'}, W_g^{(i)} \in \mathbb{R}^{d_h \times d_i'}, W_d^{(i)} \in \mathbb{R}^{d_i' \times d_h}$ with $d_i' = \frac{d_i}{N}$ and $1 \leq i \leq N$, whose selected indices set is $I_i = \{(i-1)d_i' + 1, (i-1)d_i' + 2, \cdots, id_i'\}$. The i-th expert is namely an FFN which transforms an input $x \in \mathbb{R}^{d_h}$ into

$$\left(\mathbf{x}W_u^{(i)} \odot \text{Swish}(\mathbf{x}W_g^{(i)})\right) W_d^{(i)}. \tag{3}$$

Starting from the initial values from a pre-trained model's weight recovered the performance rapidly as in most similar approaches (Komatsuzaki et al., 2022; Kim et al., 2023; Wei et al., 2024). We verified that this approach boosts the training, as described in Section A.

4.2 Layer-wise Distillation

LaDiMo trains the MoE block to mimic an FFN as in layer-wise distillation methods. Unlike the original knowledge distillation, which compares the last layer's output state (Hinton et al., 2015), LaDiMo trains the hidden states at each layer, which has been proven to improve the generalization performance (Sun et al., 2019; Jiao et al., 2019; Liang et al., 2023). Inspired by this approach, we trained the MoE block by setting the loss function as

$$\mathcal{L}_{\text{mse}} = \text{MSE}(\tilde{f}(\mathbf{x}), f(\mathbf{x})) \tag{4}$$

provided the FFN and MoE block are considered as functions f and $\tilde{f}$ respectively. Here, $\text{MSE}(\cdot, \cdot)$ is the mean squared error of two vectors.

In this scheme, the training data should consist of hidden state inputs of the FFN from the original layer. The hidden tensors can be gathered from a sampled text dataset during inference tasks.

4.3 Auxiliary Loss

In general, sparse computation using routing functions has a common issue of load imbalance over experts. This imbalance makes only a few experts to be utilized, resulting in poor performance due to the limited active parameters (Lepikhin et al., 2020). To mitigate this issue, Shazeer et al. (2017) suggested adding auxiliary losses to penalize the imbalance and encourage uniform routing. Some variations, such as a more straightforward form from Switch transformer (Fedus et al., 2022), were also employed.

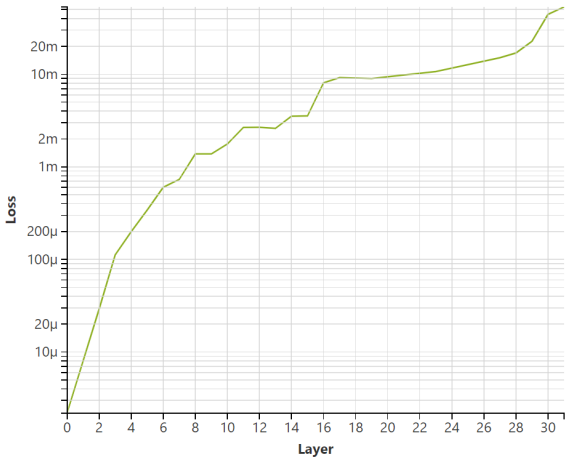

Figure 2: Training loss $\mathcal{L}_{\text{mse}}$ for each layer.

The auxiliary loss of a given MoE layer can be written as

$$\mathcal{L}_{\text{aux}} = \sum_{i=1}^{N} f_i P_i, \tag{5}$$

where f_i is the fraction of tokens out of the current batch dispatched to the i-th expert, and P_i is the fraction of the router probability given to the expert. Switch Transformer combines two types of loss with an adjustment hyper-parameter α as follows:

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{mse}} + \alpha \mathcal{L}_{\text{aux}}. \tag{6}$$

However, the coefficient α is not necessarily identical over layers (Wei et al., 2024), regarding that the order of $\mathcal{L}_{\text{mse}}$ increases as the position of the layer varies from top to bottom, as shown in Figure 2. Based on this phenomenon, we modified the loss scheme to

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{mse}} + \alpha \|\mathcal{L}_{\text{mse}}\|_2 \mathcal{L}_{\text{aux}} \tag{7}$$

where $|\cdot|_2$ denotes the L^2 norm, so that the scale of coefficient of $\mathcal{L}_{\text{aux}}$ is adaptively adjusted to maintain the balance of incorporating two losses. In our experiments, we set the empirical default value $\alpha = 0.1$.

4.4 Adaptive Router

Most MoE models are implemented with various versions of the top-k router. Recently, some researchers focused on adaptive routing, where the number of experts to be activated differ layer-by-layer, token-by-token, or both (Li et al., 2023; Huang et al., 2024; Guo et al., 2024; Zeng et al., 2024). Most approaches required training or fine-tuning for their newly designed router, which might

not be feasible under limited resources. To fully exploit throughput of LaDiMo we invented a novel approach which adaptively and dynamically routes experts in a training-free manner to mitigate the lack of training data.

We assumed that (1) when a router assigns a large logit to a single expert, dispatching the second-largest expert becomes unnecessary, and (2) if experts are assigned evenly distributed logits, then it may be necessary to consider more than two experts. To determine which case applies, we focused on the maximal routing weights over the experts, since these values tend to be large in the former case and vice versa. Our decision was based on the result of our experiment described in Section B. Furthermore, we assumed that the reference MoE model is employing static top-2 policy.

As noted in Section 4.2, the maximal routing weight for each layer and each token can be accumulated during inference tasks on a sampled text dataset. We utilized these profiled weights to determine each layer's top-k policy. This decision was based on our observation that while serving the LLM model, users' input prompts tended to form a word pool with their own characteristic distribution of maximal routing weights. For more detailed explanations, see Section B.

1. Given the hyper-parameter p_u and p_e, which are set to be the ratio of extremely uneven and even routing weight distributions respectively, find two *global quantiles* $\alpha, \beta \in [0, 1]$ such that the probability $\mathop{\mathbb{P}}_{i \in \mathscr{L}}(w_m \geq \alpha)$ of the maximal routing weight w_m being greater than or equal to α over all layers $i \in \mathscr{L}$ is equal to p_u and $\mathop{\mathbb{P}}_{i \in \mathscr{L}}(w_m \leq \beta) = p_e$. For example, if one set $p_u = p_e = 0.25$, the two parameters would equal the third and the first quartiles.

2. For each layer i, find two *local quantiles* $\alpha_i, \beta_i \in [0, 1]$ such that $\mathop{\mathbb{P}}_{i}(w_m \geq \alpha) = p_u$ where the probability is over only the i-th layer and $\mathop{\mathbb{P}}_{i}(w_m \leq \beta) = p_e$. Figure 3 illustrates how the quantiles are chosen.

3. Decide the layer policy as in Table 1. $\alpha_i > \alpha$ means the layer has relatively many uneven distributions, and $\beta_i < \beta$ means that the layer has relatively many even distributions. If $\alpha_i > \alpha$ and $\beta_i < \beta$ hold, which means that both extremely even and uneven distributions can occur within the layer, then the value of k is determined token-wise; top-1 if $w_m \geq \alpha_i$, top-3 if $w_m \leq \beta_i$, and top-2 otherwise.

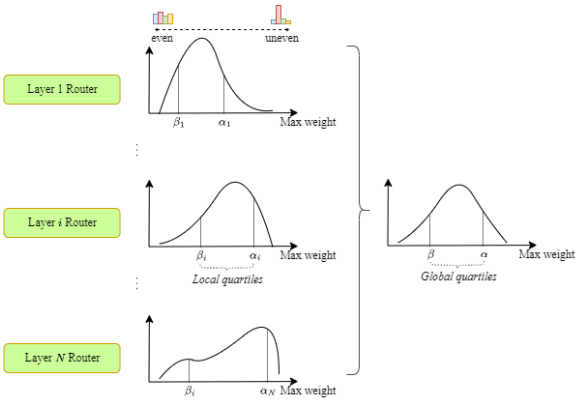

Figure 3: How the global quartiles α, β and local quartiles α_i, β_i are decided.

	$\alpha_i < \alpha$	$\alpha_i > \alpha$
$\beta_i > \beta$	Top-2	Top-1
$\beta_i < \beta$	Top-3	Dynamic (token-wise)

Table 1: Layer-wise top-k policy decision rule.

Here, token-wise dynamic top-k is chosen in the case $\alpha_i > \alpha$ and $\beta_i < \beta$ only. This is because the static top-k has less latency than the dynamic one, which not only calls the top-k function but also has to compare the maximal routing weight with α_i, β_i.

5 Experiments

We performed a series of experiments under 8 NVIDIA A100 SXM4 GPUs. We made use of DeepSpeed (Rasley et al., 2020) to train MoE blocks, vLLM (Kwon et al., 2023) and Hugging Face Transformers (Wolf et al., 2020) to measure performances of inference task, and Language Model Evaluation Harness (Gao et al., 2024) to evaluate the models.

5.1 Dataset

To train MoE blocks in our LaDiMo scheme, we needed to prepare the training dataset. Since the inputs of both FFN f and MoE block $\tilde{f}$, defined in Section 4.2, should be hidden states, it was necessary to gather such inputs for each FFN selected to be transformed into a MoE block. We leveraged the Chatbot Instruction Prompts dataset (Palla, 2023) which consists of English prompts to obtain 2M input vectors for each layer. For each training of the MoE block, we utilized these vectors as a train-

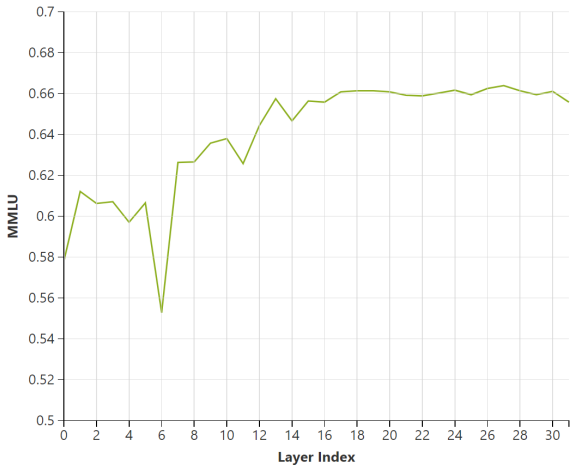

Figure 4: Relation between the MMLU score and LLaMA 3 8B models with each different single layer MoEfied. The x axis refers to the layer index from 0 to 31. All measurements were performed in each single run.

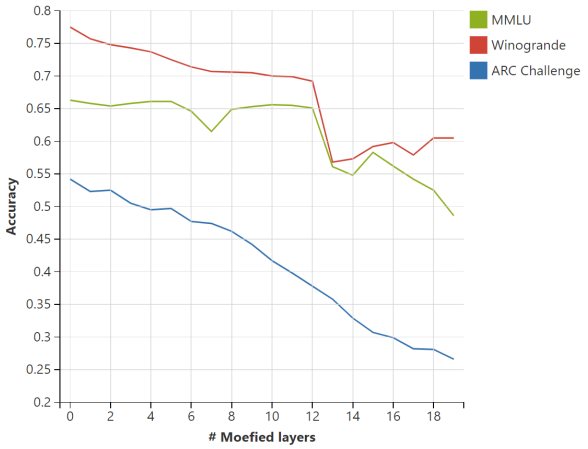

Figure 5: Effects of changes in the number of MoEfied layers to the MMLU, Winogrande, and ARC Challenge scores for LLaMA 3 8B model. All evaluations were run with five shots in each single run.

ing dataset with batch size 32 and trained for 16 epochs.

5.2 Layer Decision

We observed that when replacing the FFN from a single layer with a trained MoE block, the adverse effect on its MMLU tended to decline as the layer index increases, as shown in Figure 4. Regarding this observation, we conducted experiments by replacing the bottom-most layers with MoE blocks, varying the number of chosen layers.

5.3 Layer-wise Training

After training the MoE blocks for the bottom-most layers in a layer-by-layer manner, we observed that as the number of MoE layers increased, the MMLU, Winogrande (Sakaguchi et al., 2021), and ARC Challenge (Clark et al., 2018) scores tended to decrease as shown in Figure 5. The complete list of evaluation scores can be found in Table 2. Note that FLOPs decrease linearly as the number of MoEfied layers increases. Training for each individual layer took approximately 100 minutes.

One can choose a suitable model based on the trade-off between text quality and throughput. For instance, the partially MoEfied model, where the last 12 layers are MoEfied, reduced FLOPs by 21%, from 61.48 to 48.49 TFLOPs and latency by 10%, from 12.81 to 11.62 ms/token, while maintaining 98% MMLU accuracy of the LLaMA 3 8B model. Some scores, such as TruthfulQA suites, were better than those of the original model. The resulting model has 8.0B parameters, almost identical to the original model, but the activated parameter size counts to 6.4B, which is reduced by 20%.

We also trained the same architecture in an end-to-end non-layer-wise manner in order to verify the effectiveness of our layer-wise training method. We replaced the last twelve FFNs with untrained MoE blocks, consisting of a randomly initialized gating router and experts whose weights were obtained by splitting the original FFNs. We trained the MoE blocks simultaneously using 2M tokens from the same text dataset for the same number of epochs. Table 2 lists the evaluation scores of this model as well. It shows that our layer-wise training method achieved the goal more efficiently under the limited data size and computing resources.

5.4 Applying Adaptive Router

We applied our adaptive router introduced in Section 4.4 to the MoE layers of our 6.4B model obtained from Section 5.3, by profiling maximal routing weights during inference runs with PILE (Gao et al., 2020) ArXiv dataset. We performed a grid search on the candidate sets $S_\alpha = \{0.1, 0.2, 0.3, 0.4\}$, $S_\beta = \{0.6, 0.7, 0.8, 0.9\}$, and $S_\delta = \{0.25, 0.5, 0.75, 1.0, 1.25\}$ for hyperparameters α, β, and δ respectively, where $\delta = \alpha_i/\alpha = \beta_i/\beta$. Among all the combinations, $\alpha = 0.3, \beta = 0.8, \delta = 0.25$ attained the best performance, in which the layer 20 and 21 were set to top-2 and the other ten layers to top-1. The resulting model has 6.2B active parameters. Note that this layer decision result also proves that the

	LLaMA 3 8B	LaDiMo 6.4B	LaDiMo 6.4B with adaptive router	End-to-end
Active parameters	8.0B	6.4B	6.2B	6.4B
TFLOPs	61.48	48.49	46.69	48.49
TruthfulQA Generation BLEU (5 shots)	0.430	**0.495**	0.459	0.393
TruthfulQA MC1 (5 shots)	0.271	**0.293**	0.282	0.267
TruthfulQA MC2 (5 shots)	0.439	**0.471**	0.453	0.420
MMLU (5 shots)	**0.663**	0.651	0.645	0.632
CommonsenseQA (0 shot)	**0.694**	0.679	0.668	0.673
LogiQA (5 shots)	**0.277**	0.255	0.260	0.246
Winogrande (0 shot)	**0.727**	0.662	0.661	0.597
SIQA (0 shot)	**0.471**	0.429	0.420	0.406
PIQA (5 shots)	**0.810**	0.669	0.677	0.677
ARC Challenge (25 shots)	**0.554**	0.364	0.364	0.363

Table 2: Properties and evaluation scores on English (TruthfulQA, MMLU, CommonsenseQA (Talmor et al., 2018), LogiQA (Liu et al., 2020), Winogrande, SIQA (Sap et al., 2019)), PIQA (Bisk et al., 2020), and ARC Challenge) of LLaMA 3 8B, LaDiMo 6.4B, LaDiMo 6.4B with adaptive router, and the model trained in the end-to-end way. Best results are marked in bold, and second best results are underlined. All evaluations were performed in each single run.

importance of the bottom-most layers is relatively small, which was also indirectly implied in Figure 4. The results under these hyper-parameters are also listed in Table 2. Applying the adaptive router accelerated the inference of LaDiMo 6.4B model in terms of both throughput (20.973 to 22.823 tokens per second) and FLOPs (48.49 to 46.69 TFLOPs).

6 Limitations

In summary, we have successfully trained several MoE blocks to mimic FFNs of a given non-MoE model and converted LLaMA 3 8B model into partially MoEfied 6.4B model using only 2M tokens. The composition of MoE layers and non-MoE layers can be customized to control the trade-off between throughput and accuracy. Nevertheless, some limitations remain, and further works can be extended.

- We have selected the last layers to be MoEfied based on our observation. However, the process could be accomplished more carefully regarding importance factors of the layers, such as the sensitivity of each layer to the output. A methodology that adaptively adjusts the configuration of each MoEfied layer, including the number and size of experts, can also be considered. Generalizing the adaptive router to top-k layers for $k \geq 4$ would be accompanied along with this extension.

- We have performed experiments on the small size of datasets. However, extensive studies with more massive datasets would be helpful to understand how our approach can be effective in various situations.

- We applied our adaptive router to LaDiMo 6.4B model and observed degradation of evaluation scores for some cases. Our adaptive routing method maintains the model's accuracy by assigning top-3 to some layers, which works properly when applied to all layers. However, we applied it to the last twelve layers since these are the only MoE layers.

- Since both MoEfying and adaptive routing may sacrifice the target model's accuracy, merely pipelining them would result in amplified damage to the accuracy. More careful implementation and modification of the adaptive router for partial MoE models might be one of feasible future works.

References

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*, pages 7432–7439.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*.

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, et al. 2024. Deepseekmoe: Towards ultimate expert specialization in

mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Databricks. 2024. Introducing dbrx: A new state-of-the-art open llm.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

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

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. 2024. A framework for few-shot language model evaluation.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, and Tao Lin. 2024. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. *arXiv preprint arXiv:2405.14297*.

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*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. 2024. Harder tasks need more experts: Dynamic routing in moe models. *arXiv preprint arXiv:2403.07652*.

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.

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*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, Changbae Ahn, Seonghoon Yang, Sukyung Lee, Hyunbyung Park, Gyoungjin Gim, Mikyoung Cha, Hwalsuk Lee, and Sunghun Kim. 2023. Solar 10.7b: Scaling large language models with simple yet effective depth upscaling. *arXiv preprint arXiv:2312.15166*.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. 2022. Sparse upcycling: Training mixture-of-experts from dense checkpoints. *arXiv preprint arXiv:2212.05055*.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. 2024. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

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*.

Jiamin Li, Qiang Su, Yitao Yang, Yimin Jiang, Cong Wang, and Hong Xu. 2023. Adaptive gating in mixture-of-experts based language models. *arXiv preprint arXiv:2310.07188*.

Zonglin Li, Chong You, Srinadh Bhojanapalli, Daliang Li, Ankit Singh Rawat, Sashank J Reddi, Ke Ye, Felix Chern, Felix Yu, Ruiqi Guo, et al. 2022. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. *arXiv preprint arXiv:2210.06313*.

Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2023. Less is more: Task-aware layer-wise distillation for language model compression. In *International Conference on Machine Learning*, pages 20852–20867. PMLR.

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

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.

Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. 2023. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR.

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. *arXiv preprint arXiv:2402.14800*.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Alessandro Palla. 2023. Chatbot instruction prompts.

Bowen Pan, Yikang Shen, Haokun Liu, Mayank Mishra, Gaoyuan Zhang, Aude Oliva, Colin Raffel, and Rameswar Panda. 2024. Dense training, sparse inference: Rethinking training of mixture-of-experts language models. *arXiv preprint arXiv:2404.05567*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

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.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

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. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. 2024. Jetmoe: Reaching llama2 performance with 0.1 m dollars. *arXiv preprint arXiv:2404.07413*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. 2024. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. In *Workshop on Efficient Systems for Foundation Models II@ ICML2024*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*.

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

Tianwen Wei, Bo Zhu, Liang Zhao, Cheng Cheng, Biye Li, Weiwei Lü, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Liang Zeng, Xiaokun Wang, Yutuan Ma, Rui Hu, Shuicheng Yan, Han Fang, and Yahui Zhou. 2024. Skywork-moe: A deep dive into training techniques for mixture-of-experts language models. *arXiv preprint arXiv:2406.06563*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

Haoze Wu, Zihan Qiu, Zili Wang, Hang Zhao, and Jie Fu. 2024. Gw-moe: Resolving uncertainty in moe router with global workspace theory. *arXiv preprint arXiv:2406.12375*.

xAI. 2024. Grok-1.

Yuanhang Yang, Shiyi Qi, Wenchao Gu, Chaozheng Wang, Cuiyun Gao, and Zenglin Xu. 2024. Xmoe: Sparse models with fine-grained and adaptive expert selection. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 11664–11674.

Longfei Yun, Yonghao Zhuang, Yao Fu, Eric P Xing, and Hao Zhang. 2024. Toward inference-optimal mixture-of-expert large language models. *arXiv preprint arXiv:2404.02852*.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. 2024. Adamoe: Token-adaptive routing with null experts for mixture-of-experts language models. *arXiv preprint arXiv:2406.13233*.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2022. Mixture of attention heads: Selecting attention heads per token. *arXiv preprint arXiv:2210.05144*.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Moefication: Transformer feed-forward layers are mixtures of experts. *arXiv preprint arXiv:2110.01786*.

Haizhong Zheng, Xiaoyan Bai, Beidi Chen, Fan Lai, and Atul Prakash. 2024. Learn to be efficient: Build structured sparsity in large language models. *arXiv preprint arXiv:2402.06126*.

Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. 2024. Adapmoe: Adaptive sensitivity-based expert gating and management for efficient moe inference. *arXiv preprint arXiv:2408.10284*.

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. *arXiv preprint arXiv:2406.16554*.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*.

Simiao Zuo, Qingru Zhang, Chen Liang, Pengcheng He, Tuo Zhao, and Weizhu Chen. 2022. Moebert: from bert to mixture-of-experts via importance-guided adaptation. *arXiv preprint arXiv:2204.07675*.

A Effect of continued pre-training

Figure 6 compares the losses with and without continued pre-training method. The experiment was conducted under single NVIDIA A100 SXM4 GPU with the Chatbot Instruction Prompts dataset and the LLaMA 2 7B model.

B Experiments on adaptive router

Figure 7 shows that the distribution of the maximal routing weights of Mixtral 8x7B differed layer-by-layer. This result implies that the decision of layer policy based on the maximal routing weight works. We used Chatbot Instruction Prompts dataset to collect routing weights.

Figure 8 shows the distribution of the maximal routing weights of Mixtral 8x7B with various PILE datasets. Datasets with similar text styles, *e.g.*, ArXiv and PhilPapers, showed a similar distribution, while Github and DM Mathematics showed different results.

Finally, we verified the concept of our adaptive router to Mixtral 8x7B model using PILE Github dataset to profile maximal routing weights. We performed the grid search on the candidate sets $S_\alpha = \{0.1, 0.2, 0.3, 0.4\}$, $S_\beta = \{0.6, 0.7, 0.8, 0.9\}$, and $S_\delta = \{0.25, 0.5, 0.75, 1.0, 1.25\}$ for hyperparameters α, β, and δ respectively, where $\delta = \alpha_i/\alpha = \beta_i/\beta$. Among all of possible combinations we found the optimal setting $\alpha = 0.2, \beta = 0.8, \delta = 0.75$ with MMLU score 71.55 and throughput 13.92, which are better than the original model's 71.50 and 11.68. Figure 9 shows the resulting layer-wise routing policies.

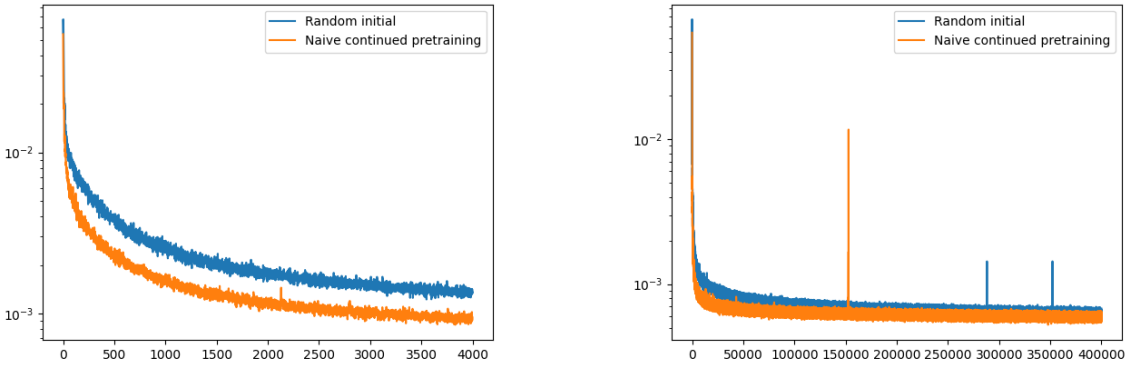

Figure 6: Continued pre-training gives a smaller loss than starting from random initial weights.

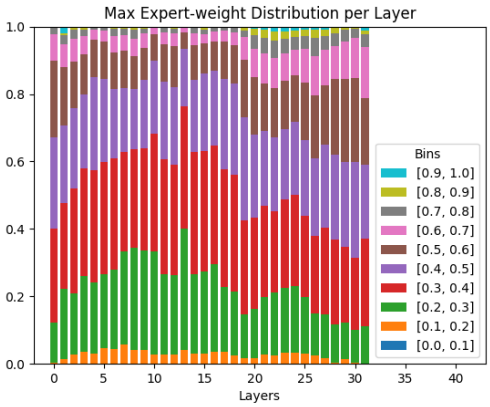

Figure 7: Maximal routing weights distribution per layer of Mixtral 8x7B, gathered during inference tasks.

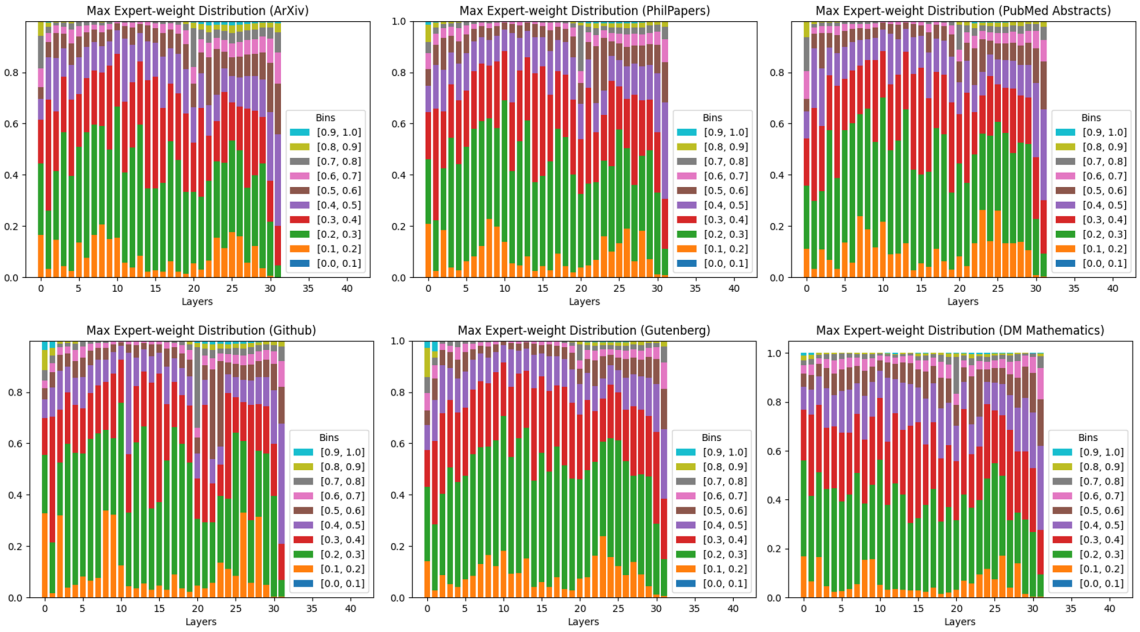

Figure 8: Maximal routing weights distribution of Mixtral 8x7B with various PILE datasets.

Figure 9: Layer-wise routing policy obtained from Mixtral 8x7B model and PILE Github dataset with hyperparameters $\alpha = 0.2, \beta = 0.8, \delta = 0.75$ for each layer 0 to 31. 1, 2, 3, and D stands for top-1, top-2, top-3, and dynamic top-k respectively.