Soft-routing Mixture of Experts for Autoregressive Language Model Pre-training

Anonymous Authors[1]

Abstract

Mixture-of-expert (MoE) models enable scaling model sizes with little increase of computation. Recently, fully differentiable MoE architectures have been proposed to address training difficulties via end-to-end gradient propagation. Despite their promising results in computer vision and text classification tasks, how to pre-train autoregressive language models (LMs) with such architectures remains an open question. In this work, we present SOAP: Soft-Routing Mixture of Experts for Autoregressive Language Model Pre-training, a novel approach for efficient pre-training of MoE LMs. SOAP consists of two key solutions: (1) We first propose a segment-level routing strategy, in which the previous segment is used to route the next segment in an autoregressive manner; (2) We pre-train our MoE models by concatenating similar documents sequentially to make our segment-level router more effective in expert specialization. Experimentally, we train soft-routing MoE models with up to 32 experts and 30B (1.5B active) parameters, and show that SOAP leads to significant performance gains over parameter-matched dense models on various tasks, including language modeling (+13.9%), commonsense reasoning (+3.7%), reading comprehension (+3.3%), closed-book QA (+1.5%), and text classification (+11.1%). Further analysis demonstrates that our trained experts can capture domain-level specialization without additional supervision.

1. Introduction

Mixture-of-experts (MoE) models route a given input to a small subset of model parameters (known as *experts*) for computation, realizing the strong performance of a large model size while staying with computational efficiency (Lepikhin et al., 2020; Fedus et al., 2022; Du et al., 2022; Zoph et al., 2022; Lewis et al., 2021; Zhou et al., 2022). These models are trained to route given inputs to a few experts, introducing the complexity of learning a non-differentiable, discrete decision-learning problem (Shazeer et al., 2017; Zoph et al., 2022).

Recently, fully differentiable MoE architectures based on soft-routing have been proposed (Puigcerver et al., 2023; Muqeeth et al., 2023), showing the promise of training models via end-to-end back-propagation. While these models achieve promising results on computer vision and text classification tasks, it remains challenging to pre-train autoregressive language models (LMs) with such architectures. Firstly, soft-routing MoE models introduces a computationally expensive merging operation (Muqeeth et al., 2023), making token-level routing infeasible. Secondly, existing input-level routing strategies make the routing decision on the entire input sequence, which disrupts the autoregressive nature intrinsic to decoder-only models. Moreover, during pre-training, the models that employ input-level routing are trained to route all documents in the same training sample together. This practice, coupled with the random concatenation of irrelevant documents within a training sample, can potentially leads to a scenario where experts are not sufficiently specialized.

In this paper, we propose SOAP (Soft-Routing Mixture of Experts for Autoregressive Language Model Pre-training) (Figure 1), to train autoregressive LMs with soft-routing MoE architectures. Our method consists of two key solutions. (1) We first propose the *causal segment routing* strategy. For a given input, we split it into multiple segments with a fixed length, where each segment is used to route the next segment in an autoregressive manner. This segment-level routing strategy maintains the autoregressive property of the models and also achieves high computational efficiency by merging experts only once per segment. (2) We also propose a similarity-based data batching method, inspired by Shi et al. (2023), which sequentially concatenates similar documents to construct training instances. This prevents the models from routing irrelevant documents together, encouraging experts to learn the specialization in specific domains or topics.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

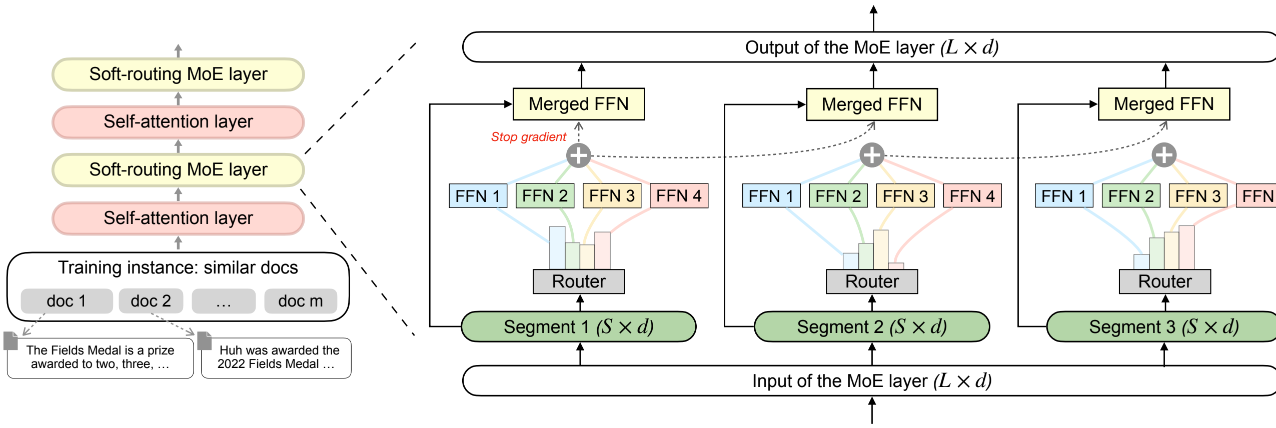


Figure 1: The illustration of the proposed training method SOAP. We adapt the fully differentiable soft-routing MoE architecture (Muqeeth et al., 2023) that merges multiple FFN networks into one using the routing weights at each MoE layer. SOAP is based on two key techniques: first, we use the *causal segment routing* strategy which does not introduce substantial compute overhead while preserving the autoregressive nature of the model; second, we employ the *similarity-based data batching* method to construct training samples by sequentially concatenating similar documents. L: sequence length of training samples; d: embedding dimension of the model; S: length of the segment.

We evaluate our approach by training different sizes of language models with 0.3B, 1.5B active parameters and with up to 32 experts. Our experimental results suggest that SOAP lead to soft-routing MoE models that significantly outperform parameter-matched dense baseline models trained with the same amount of data, achieving significant performance gains on language modeling (+13.9%), commonsense reasoning (+3.7%), reading comprehension (+3.3%), closed-book QA (+1.5%), and text classification (+11.1%). Furthermore, our analysis reveals that the experts trained through SOAP are able to capture domain-level specialization without additional supervision.

2. Preliminaries

2.1. Background: Sparsely-activated MoE

Transformer-based sparse MoE models (Shazeer et al., 2017; Fedus et al., 2022; Zoph et al., 2022) replace feed-forward network (FFN) layers with MoE layers. Each MoE layer consists of E expert FFNs, parameterized as $\text{FFN}(\cdot;\theta_1), \ldots, \text{FFN}(\cdot;\theta_E)$, where $\text{FFN} : \mathbb{R}^{L\times d} \to \mathbb{R}^{L\times d}$. Given an input x, a conventional sparse MoE layer computes the output y by sparsely activating one or more experts in this layer:

$$y = \sum_i e_i \cdot \text{FFN}(x; \theta_i),$$

where e_i represents the routing weight for the i-th expert of the input at this MoE layer. The routing weight e_i is typically measured by a *routing network* or *router* R, which takes a hidden representation as input and employs a linear operation followed by a softmax, $e_i = \text{Softmax}(R(x))_i$. For the i-th expert, if the corresponding routing weight $e_i = 0$, we do not need to compute $\text{FFN}(x;\theta_i)$, i.e., the i-th expert is not activated. In practice, only k experts with top routing weights are activated at each layer, i.e., $e_i = \text{Top-}k(\text{Softmax}(R(x)))_i$.

2.2. Soft-routing MoE

Training sparsely-activated MoE models has been shown to be difficult (Fedus et al., 2022; Zoph et al., 2022) Recently, fully differentiable MoE architectures have been proposed (Muqeeth et al., 2023; Puigcerver et al., 2023), with the promise of training entire models via end-to-end gradient back-propagation. Our work focused on the soft-routing MoE architecture, proposed by Muqeeth et al. (2023). Instead of activating a limited number of experts at each layer, we compute a weighted average of all expert FFNs with routing weights in the parameter space, yielding a *merged FFN*. The input x is passed into the merged FFN to compute the output y:

$$y = \text{FFN}(x; \sum_i e_i \cdot \theta_i). \quad (1)$$

In soft-routing MoE models, as the input x is only fed into the merged FFN network, the computational overhead compared to a dense model comes from computing the routing weights e_i and merging the experts into one FFN.

Applicability to autoregressive model training Here, we discuss the possibility of applying the existing training technique to train autoregressive LMs with the soft-routing MoE architecture (Muqeeth et al., 2023). As Equation 1 shows, these soft-routing MoE models require the computation of a weighted average of all expert

Algorithm 1 Pseudocode of causal segment routing.

```
# B: batch size (number of training instances)
# L: length of each training instance
# d: hidden dimension
# E: number of experts
# S: length of each segment
# R: routing network (input: hidden rep, output:
    routing weights)

input x # x: input tensor (BxLxd)

N = L // S # number of segments in each sample
seg_x = x.view(B*N, S, d) # split x into segments

# representation of each segment (BNxE)
repr = mean(seg_x, dim=1)

# routing results (not causal) (BNxE)
e = softmax(R(repr), dim=-1)

# routing results for the first segment
e_first = e.view(B, N, E)[:, 0]

# make causal routing results (shift 1)
e = roll(e, 1) # shift by 1

# set routing results of the first segment
e = e.view(B, N, E) # back to the instance view
e[:, 0] = stop_grad(e_first) # assign w/ stop gradient
e = e.view(B*N, E)

# MoE FFN forward with expert weights e
seg_y = moe_ffn(seg_x, e) # seg_y: B*N x S x d

# back to the instance view
y = seg.y.view(B, L, d)

return y
```

`moe_ffn`: compute the merged expert and process the input (equation 1).

parameters with each routing decision made. Although processing the input in the merged expert leads to comparable computation costs to using a single expert, the averaging operation that merges expert parameters may introduce a substantial computational overhead when the routing network is run at each position in the sequence. Muqeeth et al. (2023) approaches this challenge by simply making a single routing choice for the entire input example. This strategy enables the training of a text classification model in downstream tasks (Muqeeth et al., 2023), it does not apply to the training of autoregressive LMs. First, the sentence-level routing strategy assumes that the router network has access to the entire input sentence, even when computing the loss at the middle positions. This disrupts the autoregressive nature of trained LMs. Second, during pre-training, short documents are randomly concatenated to form a training instance. Making routing decisions based on the entire input encourages the merged expert to adopt a "generalist" approach to process all input documents, which may undermine the specialization of expert FFNs.

3. Our Method: SoAP

In order to train autoregressive language models with a soft-routing arthicture, we propose SoAP (Soft-Routing Mixture of Experts for Autoregressive Language Model Pre-training) (Figure 1), which is based on two key techniques: causal segment routing and similarity-based data batching.

3.1. Causal Segment Routing

As discussed above, token-level routing leads to a large computational overhead, while routing based on the entire input disrupts the autoregressive nature of the model. We propose the *causal segment routing* strategy to realize the computational efficiency of routing while ensuring that the model's computations are conditioned on preceding positions, thus preserving its autoregressive integrity. Algorithm 1 shows the pseudocode of our routing strategy. Given a training instance consists of L tokens (e.g., $L = 4096$), we split the training instance into multiple segments, each of which contains S (e.g., $S = 256$) consecutive tokens. During training, for each segment within an instance except for the first segment, we compute the mean hidden representation of the preceding segment and feed this as input to the routing network for each MoE layer. For the first segment, the representation of the segment itself is used to compute the routing weights. However, this may cause the model to leak information through the router; thus, we apply a stop-gradient operation on top of the routing results of the first segment to avoid the model from optimizing based on potential information leakage. Such segment-level routing avoids merging FFNs at each position, largely reducing computational overhead. In Appendix A, we analyze the computational overhead that the causal segment routing strategy introduces.

3.2. Similarity-based Data Batching

When training language models on large corpora, irrelevant documents are randomly concatenated to form a training instance with a fixed context window length. However, in segment-level routing strategy, routing irrelevant documents together may lead to less specialized experts, as it trains the model to process those documents of various domains with the same merged FFN. To address this issue, we adapt the idea of data batching in in-context pre-training (Shi et al., 2023). Instead of randomly batching irrelevant documents to form training samples, we measure the similarity between documents using Contriever (Izacard et al., 2021) and sequntially concatenate similar documents to construct training instances. Similar to Shi et al. (2023), we employ a greedy algorithm to order all documents and construct training instances (see Appendix B for more details).

3.3. Routing During Downstream Inference

During inference for downstream tasks, we are given a prompt text and would like to generate the continuations. We consider two routing strategies during inference. First, we follow the same schema as our training method. We split the input prompt into multiple segments with the same fixed length as the pre-training and use each segment to compute the routing weights for the next segment[1]. During generation, once the current segment is filled with newly generated tokens, we perform the routing again and compute new merged experts. Second, we only make a single routing choice once based on the entire input prompt. All generations are based on the merged experts computed upon the input prompt at the beginning, and the merged experts are not refreshed during generation. In Appendix E.1, we show that these two inference routing strategies do not make substantial differences in the downstream tasks we evaluated. We then use the prompt-routing strategy by default as it requires only a one-time routing choice. Note that after the one-time routing choice, we process the entire input only on the merged experts, making the generation procedure as simple and efficient as dense models.

4. Experiments

In this section, we conduct experiments to evaluate our approach on training autoregressive language models.

4.1. Setup

Models We evaluate our approach by training decoder-only Transformer models which consists of active parameters of 0.3B and 1.5B[2]. For each FFN layer in the Transformer model, we replace it with MoE layers with E ($E \in \{8, 16, 32\}$) experts with exactly the same architecture. Appendix C shows the configuration of model architectures as well as the total parameter count. We follow LLaMA (Touvron et al., 2023a) and use SwiGLU (Shazeer, 2020) as the activation function in FFNs. We use the same tokenizer as the LLaMA models (Touvron et al., 2023a;b). All models are trained with a 4096-token context window. In the causal segment routing strategy, we set the length of each segment to be $S = 256$.

Training details We employ the AdamW optimizer (Loshchilov & Hutter, 2017) with $\beta_1 = 0.9$ and $\beta_2 = 0.95$ and use a learning rate of $2e-4$ with a cosine learning rate scheduler. All 0.3B models are trained using 32 A100 GPUs with a batch size of 1 million tokens; 1.5 models are trained using 64 GPUs with a batch size of 1 million tokens.

Warmup and initialization At the beginning of training, we train a parameter-matched dense model and duplicate the FFN layers as initialization of the MoE model. In our experiments, we use the first 5% training steps as the warmup to initialize the MoE weights. We find that without warmup training, there may be more experts under-utilized (see Appendix E.2 for an ablation study). We also apply a linear warmup to the learning rate scheduler for the first 5% training steps.

Training datasets We follow Shi et al. (2023) and use the Commoncrawl dataset (Wenzek et al., 2019). We randomly sample a subset of Commoncrawl, which consists of 150 billion tokens. We apply the similarity-based data batching method on this subset to construct all training instances.

Evaluation datasets We evaluate the pre-trained models on language modeling tasks. We measure the perplexity of trained models on held-out evaluation datasets sampled from arXiv, Books Corpora, Wikipedia, C4 (Raffel et al., 2020), and Python code (a Python subset of Github). Each evaluation dataset contains 1K samples, each of which consists of 2048 tokens.

We also evaluate models in downstream tasks with in-context learning (Brown et al., 2020), including common sense reasoning: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrand (Sakaguchi et al., 2021); reading comprehension: RACE (Lai et al., 2017), ARC (Clark et al., 2018)); closed-book QA: Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017); and text classification: AGNews (Zhang et al., 2015), SST-2 (Socher et al., 2013), Amazon and Yelp (Zhang et al., 2015), FEVER (Thorne et al., 2018), MRPC (Dolan & Brockett, 2005).

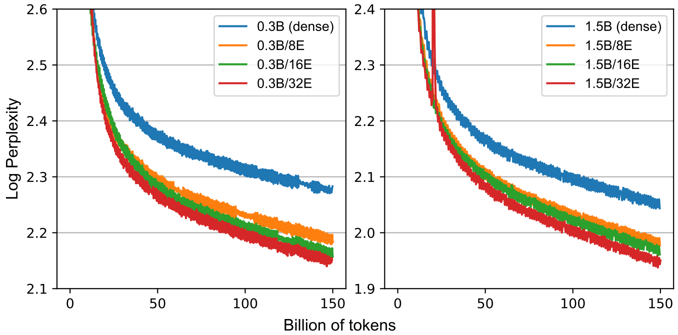
4.2. Main Results

Training efficiency and convergence Figure 2 (left) shows the training loss curves of the dense model and our MoE models with different model sizes. First, we find that with the same amount of training tokens, our models clearly achieve better training loss compared to the dense model baseline. For the 0.3B and 1.5B models, our models with 32 experts achieve the same level of loss with fewer than half of the training tokens. This indicates that our approach achieves much better performance with the same training FLOPs (see analysis of additional FLOPs from MoE layers in Appendix A). We also observe that when using more experts, we are able to gain more improvement.

Language modeling We evaluate trained models on language modeling evaluation sets. As shown in Figure 2 (right), our MoE models outperform the dense baseline in all domains, significantly reducing perplexity. For example, our 0.3B/32E model achieves a relative

[1]Similar to training, we use the first segment to compute the routing weights for itself.

[2]In Appendix D, we additionally conduct experiments on a 7B dense model and a 7B/4E MoE model *without* using similarity-based data batching. Due to the limited computing resources, we are not able to train 7B models on the batched dataset.



Model	arXiv	Books	Wiki	C4	Python
0.3B	8.4	18.0	10.3	13.8	15.2
0.3B/8E	7.4	16.0	9.2	13.3	12.5
0.3B/16E	**7.2**	15.7	9.1	13.1	12.2
0.3B/32E	**7.2**	**15.5**	**8.9**	**13.0**	**11.7**
1.5B	6.6	13.6	7.8	10.7	10.4
1.5B/8E	6.2	12.8	7.6	10.6	10.1
1.5B/16E	6.0	12.4	**7.1**	10.6	8.9
1.5B/32E	**5.8**	**12.3**	**7.1**	**10.4**	**8.7**

Figure 2: Left: training curves (log perplexity) of models with different sizes and experts. Right: Perplexity of trained models on different evaluation sets (arXiv, Books, Wikipedia, C4, and Python).

improvement of 13.9% on Books compared to the 0.3B dense model. We observe that the improvement is especially large in test domains that are markedly different from the domains of the training dataset (e.g. Python). We consider this as a strong indication of expert specialization in specific domains (we further study expert specialization in Section 5.4).

Downstream tasks Table 1 shows the model performance on downstream tasks. We observe significant performance across all tasks. For example, our 0.3B/32E model achieves an average performance improvement of $+3.7\%$ in common sense reasoning, $+3.3\%$ in reading comprehension, $+1.5\%$ in reading comprehension, and $+11.1\%$ in text classification. This suggests that our MoE models are better at memorizing knowledge, understanding and reasoning over the context, and classifying sentences based on different criteria.

5. Analysis and Ablation Studies

In this section, we conduct ablation studies and analysis to understand the essence of each component of our approach.

5.1. Importance of Causal Segment Routing

We study the importance of our causal segment routing strategy. As during inference, we first encode the entire input prompt and compute the routing weights at each MoE layer. A natural alternative strategy for training routers is to regard a prefix of the training instance as the "prompt" and route the entire training instance using the prefix. Specifically, we implement a *prefix routing* strategy where we use the first segment as the prefix prompt. Similarly, we apply a stop-gradient operation when processing the first segment on the merged FFNs. As shown in Figure 3, although we use similarity-based data batching to construct training instances, only using a prefix for routing leads to much worse performance compared to using causal segment routing. These results suggest the importance of using every segment to provide strong training signals for routers.

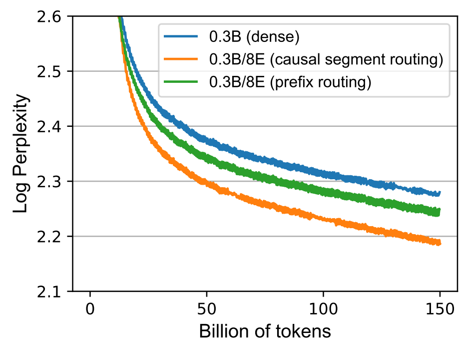


Figure 3: Training curves of using different routing strategies. We investigate the importance of using causal segment routing. We compare with prefix routing, a straightforward segment-level routing strategy that uses the prefix of the input (the first segment).

5.2. Importance of Similarity-based Data Batching

To investigate the importance of using similarity-based data batching, we compare the improvement we gain from training MoE models with and without the similarity-based batching method. First, Figure 4 (a) shows the training loss of the dense models and MoE models with eight experts when using the similarity batched data (*sim batch*) and the original randomly batched data (*rand batch*). We find that on both training sets, our MoE models clearly outperform the dense models, achieving a lower training loss. Furthermore, we compare the loss improvement (i.e., the difference between the loss of dense models and the loss of MoE models) of the MoE models in Figure 4 (b). We observe that with the similarity-based batching method, the loss improvement is much larger, and it is further enlarged when training with more update steps. These results clearly suggest that it is crucial to apply similarity-based batching in order to train effective soft-routing MoE models.

Table 1: We compare the MoE models trained with SOAP with the parameter-matched dense models on downstream tasks, including commonsense reasoning, reading comprehension, closed-book QA, and text classification.

	Commonsense Reasoning					Reading Comprehension			
Model	**PIQA**	**SIQA**	**BoolQ**	**HellaSwag**	**WinoGrande**	**RACE-m**	**RACE-h**	**ARC-e**	**ARC-c**
0.3B	65.8	42.7	44.6	34.6	51.2	41.7	30.9	51.5	21.3
0.3B/8E	67.5	41.2	41.2	34.8	**54.4**	43.1	31.4	52.4	22.1
0.3B/16E	67.2	**44.1**	56.6	**34.9**	54.1	**43.9**	31.1	54.8	24.9
0.3B/32E	**68.2**	43.0	**58.0**	34.7	53.4	42.7	**32.0**	**57.4**	**26.3**
1.5B	71.2	45.0	54.0	**43.9**	60.9	50.1	36.7	65.0	31.0
1.5B/8E	**72.1**	45.2	**62.0**	43.6	**63.7**	51.2	36.5	66.3	32.5
1.5B/16E	71.3	45.0	56.0	43.7	61.5	**51.7**	**37.3**	66.3	**32.7**
1.5B/32E	**72.1**	**47.1**	59.9	43.8	61.9	51.5	32.4	**66.7**	**32.7**

	Closed-book QA		Text Classification						Avg
Model	**NQ**	**TQA**	**AGNews**	**Amazon**	**SST-2**	**Yelp**	**Fever**	**MRPC**	
0.3B	4.7	8.8	30.3	53.6	54.6	66.0	47.6	62.0	41.8
0.3B/8E	5.3	9.0	38.4	52.3	54.6	62.6	56.6	59.0	42.7
0.3B/16E	**6.0**	**10.2**	36.3	**75.6**	53.3	64.0	**57.0**	**65.0**	45.8
0.3B/32E	5.3	**10.2**	**47.3**	64.0	**55.3**	**73.3**	55.7	56.0	**46.0**
1.5B	**7.6**	23.8	64.0	65.3	80.0	58.6	**59.0**	**66.7**	51.9
1.5B/8E	7.3	24.2	**65.0**	94.0	80.0	88.3	57.0	64.0	56.1
1.5B/16E	7.3	**25.6**	61.6	78.3	84.6	93.6	57.3	63.6	55.1
1.5B/32E	7.0	25.4	62.3	**94.7**	**85.0**	**95.3**	56.3	**66.7**	**56.5**

5.3. Comparison with Existing MoE Models

We compare our approach with a state-of-the-art MoE method Expert Choice (EC) (Zhou et al., 2022), where each expert selects top-k inputs according to the routing weights to ensure the balanced load during training. We consider two variants of the EC MoE models. In both variants, we set the capacity factor of experts to 1 to ensure that the amount of computation is roughly the same as our MoE models. First, we follow our segment routing strategy and train a sparse MoE model with the EC method. During training, each expert selects the top segments, and all tokens in this segment are fed into this expert. This variant is to investigate the improvement that we can get by merely using a fully differentiable soft-routing MoE with the same routing strategy. Second, we consider the original setting of EC models where token-level routing is used. Here, the goal is to have an end-to-end comparison to the existing SoTA MoE models with the same amount of training computation.

The training loss curves are shown in Figure 5. First, we observe that our approach (blue curve) significantly outperforms segment-level EC (orange curve) with the same routing setting. This suggests that with the same routing strategy, using a fully differentiable architecture is more effective than a sparse MoE, thanks to the end-to-end gradient back-propagation. On the other hand, the token-level EC model leads to a loss curve similar to that of our model. This indicates that although our models make a coarser-grained routing decision (segment-level), they are able to achieve the same level of performance compared to token-level MoE models.

Table 2: Perplexity of our trained MoE model and EC models on evaluation sets. We instantiate EC methods with our segment-level routing and the original token-level routing.

Model	arXiv	Books	Wiki	C4	Python
0.3B/8E (SOAP)	**7.4**	**16.0**	**9.2**	13.3	**12.5**
0.3B/8E (EC, segment-level)	7.9	17.6	10.5	14.1	20.8
0.3B/8E (EC, token-level)	7.5	17.0	**9.2**	**12.8**	23.7

Table 2 shows the perplexity of the models on held-out evaluation sets. We find that the token-level EC model performs better than our model on the C4 evaluation data, which is likely on the most similar distribution to the training set (Commoncrawl); on arXiv, Books, and Wikipedia, EC performs similarly or slightly worse than our model. Surprisingly, on the Python evaluation set, the token-level EC model performs particularly badly, achieving even worse perplexity than the segment-level EC model. We think this suggests that segment-level routing models are particularly better at learning domain-level specialization, as segment-level global features are captured by routing networks. The better domain-level expert specialization makes the model achieve good performance on out-of-domain evaluation data (we assume that there is only a very small part of Python code in Commoncrawl). We further investigate the expert specialization in Section 5.4

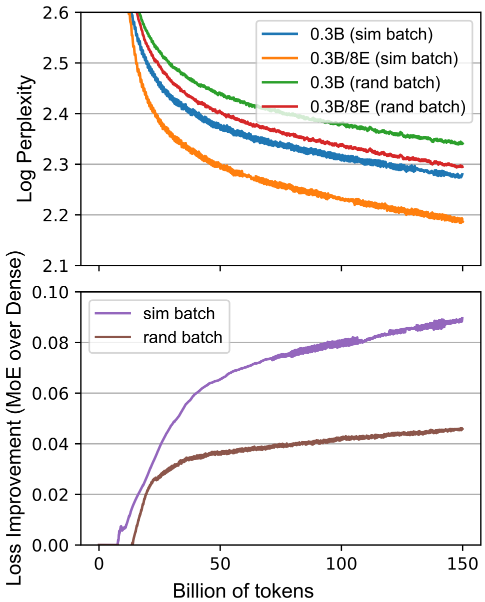


Figure 4: We study the effects of using the similarity-based data batching method. Top: we show the training curves of the 0.3B dense model and the 0.3B/8E model with similarity-based data batching (*sim batch*) or the standard random batching (*rand batch*). Bottom: we show the training loss improvement of the MoE model over the dense counterpart in different scenarios. Using similarity-based batching leads a much larger loss improvement.

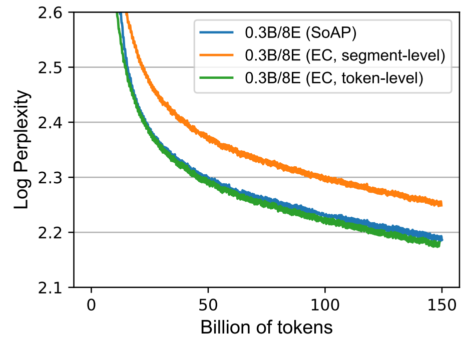


Figure 5: Comparison with the state-of-the-art MoE training technique Expert Choice (EC). We implement two variants of EC: one trained with the original token-level routing strategy, where each expert selects top tokens; the other trained with the segment-level routing strategy, where each expert selects top segments. For both EC models, we use the capacity factor of 1 with the same amount of FLOPs as our training method for the fair comparison.

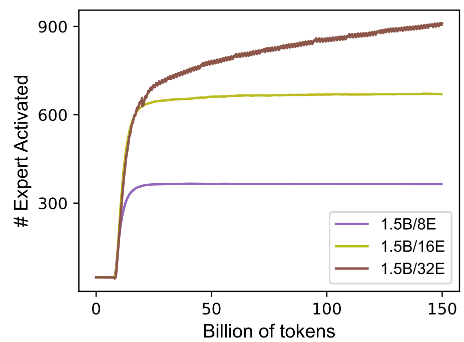


Figure 6: We show how many experts are actively utilized every 10 training steps during training. We define an expert is activated if the weight is larger than $\frac{E}{2}$, where E denotes the number of experts at each MoE layer.

5.4. Expert Utilization and Specialization

Utilization: How many experts are actively utilized? One potential issue of training MoE models is the models may collapse to dense models because most experts are under-utilized (e.g., some experts have never been activated). Here, we investigate the expert utilization of our soft-routing MoE models. We define an expert that is activated for a given input if the routing weight is larger than $\frac{E}{2}$. In Figure 6, we plot the number of experts that are activated at least once among 10 training steps when training 1.5B MoE models (with 48 layers; therefore, the 1.5B/8E, 1.5B/16E, 1.5B/32E models have 384, 768, 1536 experts in total, respectively). We see that after the warmup phase at the beginning, the expert utilization quickly increases. 1.5B/8E and 1.5B/16E models have quickly utilized most of the experts; while the expert utilization of the 1.5B/32E model continues to increase until the end of the training. This indicates that our approach is able to prevent the MoE models from collapse to dense models and achieves high expert utilization. However, when training with a large number of experts, training the model to activate all experts is more challenging.

Specialization: What do experts learn? In order to study the expert specialization, we investigate the averaged routing weights at different layers of the 0.3B/8E model, on different domains (Books, arXiv, Python, and Wikipedia). Figure 7 shows the routing weights at layer 0, 11, and 23 (the first, middle, and last layer) of the 0.3B/8E model. First, we find that there exists clear domain-level expert specialization in our trained MoE models, even though no additional domain-level supervision is used during training. For instance, expert 7 at layer 11 is specialized to process inputs in the arXiv domain. We also observe that routing weights on arXiv and Python code are more similar compared to Books and Wikipedia, likely because LaTex code and Python code are dissimilar to natural language. Second, experts at the middle or high layers are more specialized in specific domains, while the routing weights at lower layers are similar and flat across domains.

It is worth noting that our learned experts behave differently from those of prior token-level MoE models, where shallow token-level specialization is observed. For example, some experts are specialized for a specific type of word (e.g.,

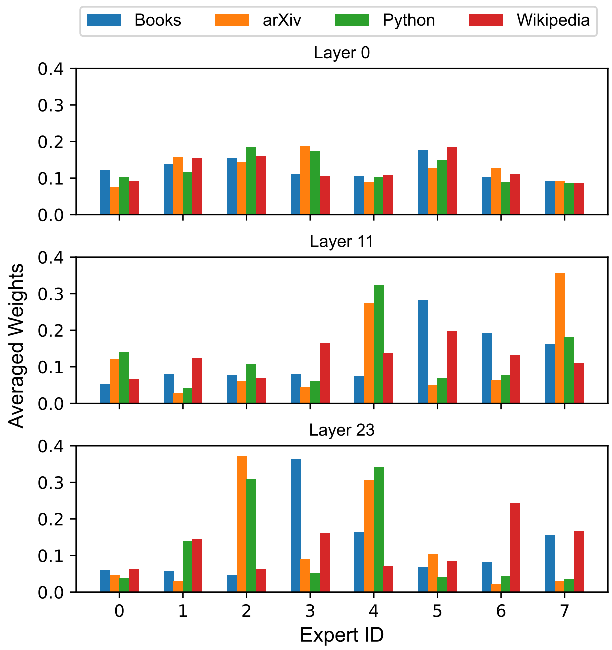


Figure 7: Averaged routing weights at layer {0, 11, 23} of the 0.3B/8E model on different domains (Books, arXiv, Python, Wikipedia). We observe that the experts in our MoE models learn domain-level specialization, especially at middle and higher layers.

punctuations, articles), and few deep semantic features are captured by the learned routers (Jiang et al., 2024; Lewis et al., 2021; Zoph et al., 2022; Shazeer et al., 2017). Our models learn domain-level specialization. We think this is due to the segment-level routing strategy we use during training, which enables the routers to capture global features beyond token level. This suggests that segment-/sentence-level routing strategies and token-level routing strategies capture complementary features, opening up opportunities to combine them to build even stronger models.

5.5. More Analysis

In Appendix E, we further show that (1) during inference of downstream tasks, routing the entire input prompt once or routing each segment does not make substantial differences on the tasks we evaluate; (2) warmup training is crucial to achieve high expert utilization, especially when training MoE models with a large number of experts.

6. Related Work

Mixture of Experts Sparsely gated MoE models (Shazeer et al., 2017) have been proposed to demonstrate the potential of massively scaling up model sizes. GShard (Lepikhin et al., 2020) adapts the sparse MoE architecture into Transformer models and achieves strong results on machine translation. Recent work has extended it to general language models (Fedus et al., 2022; Zoph et al., 2022; Jiang et al., 2024; Dai et al., 2024; Zhou et al., 2022; Du et al., 2022; Artetxe et al., 2022). Traditional MoE models are trained to route given inputs to one or a few specialized expert modules, which introduces a non-differentiable, discrete decision-learning problem. These existing models are trained with the top-1 or top-2 routing strategy on a carefully designed load balancing objective (Lepikhin et al., 2020; Fedus et al., 2022; Zoph et al., 2022), or employ complicated assignment algorithms to distribute inputs (Lewis et al., 2021; Roller et al., 2021; Zhou et al., 2022). Training MoE models has been shown to be difficult, facing the issues of training instability, expert under-specialization, poor training efficiency (Zoph et al., 2022).

Our approach enables end-to-end gradient back-propagation by employing fully differentiable MoE architectures (Muqeeth et al., 2023; Puigcerver et al., 2023). Soft MoE models (Puigcerver et al., 2023) softly merge visual tokens that are processed by an expert in computer vision tasks. SMEAR (Muqeeth et al., 2023) proposes softly merging experts by taking a weighted average on the parameter space. Our MoE models are built on the SMEAR architecture with all FFN layers replaced by MoE layers. However, existing differentiable MoE models only focus on an encoder architecture, while our proposed SoAP method enables training autoregressive soft-routing MoE models.

Similarity-based data batching There exists research that applies a similar data batching method during training. In-context pre-training (Shi et al., 2023) groups relevant documents together to encourage language models to leverage long-range contexts and improve the results of in-context learning and retrieval augmentation. TRIME (Zhong et al., 2022) batch documents with high lexical similarity to collect more positive pairs in a contrastive learning framework to provide stronger training signals. Although sharing the same idea, the goal of our data batching method is to avoid routing irrelevant documents together, which may hurt the expert specialization.

7. Conclusion

In this paper, we propose SoAP, a training method to train autoregressive MoE language models with soft routing. Our extensive experiments demonstrate that our MoE models significantly outperform baseline models on language modeling tasks and downstream applications. We also observe that trained experts are highly specialized and capable of capturing domain-level information. Future research includes further scaling up our MoE models, combine=ing token-level routing and segment-level routing, and developing efficient decoding methods for soft-routing MoE models.

Impact Statements

This paper presents a new approach for building large language models. We would like to note that, similar to existing language models, the language models trained with our approach may have the same potential societal consequences. For example, language models can produce factually inaccurate outputs (e.g., Min et al. (2023)), facing the risk of spreading misinformation; malicious users can extract training data that is used to train language models (Carlini et al., 2021), causing privacy and license problems. We acknowledge these potential negative consequences and caution those who use our approach to build powerful language models.

References

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., Anantharaman, G., Li, X., Chen, S., Akin, H., Baines, M., Martin, L., Zhou, X., Koura, P. S., O'Horo, B., Wang, J., Zettlemoyer, L., Diab, M., Kozareva, Z., and Stoyanov, V. Efficient large scale language modeling with mixtures of experts. In Goldberg, Y., Kozareva, Z., and Zhang, Y. (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11699–11732, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.804. URL https://aclanthology.org/2022.emnlp-main.804.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dai, D., Deng, C., Zhao, C., Xu, R. X., Gao, H., Chen, D., Li, J., Zeng, W., Yu, X., Wu, Y., Xie, Z., Li, Y. K., Huang, P., Luo, F., Ruan, C., Sui, Z., and Liang, W. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *CoRR*, abs/2401.06066, 2024. URL https://arxiv.org/abs/2401.06066.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. URL https://aclanthology.org/I05-5002.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

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

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. Unsupervised dense information retrieval with contrastive learning, 2021. URL https://arxiv.org/abs/2112.09118.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Min, S., Krishna, K., Lyu, X., Lewis, M., Yih, W.-t., Koh, P., Iyyer, M., Zettlemoyer, L., and Hajishirzi, H. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12076–12100, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.741. URL https://aclanthology.org/2023.emnlp-main.741.

Muqeeth, M., Liu, H., and Raffel, C. Soft merging of experts with adaptive routing. *arXiv preprint arXiv:2306.03745*, 2023.

Puigcerver, J., Riquelme, C., Mustafa, B., and Houlsby, N. From sparse to soft mixtures of experts. *arXiv preprint arXiv:2308.00951*, 2023.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

Roller, S., Sukhbaatar, S., Weston, J., et al. Hash layers for large sparse models. *Advances in Neural Information Processing Systems*, 34:17555–17566, 2021.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sap, M., Rashkin, H., Chen, D., LeBras, R., and Choi, Y. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

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

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shi, W., Min, S., Lomeli, M., Zhou, C., Li, M., Lin, V., Smith, N. A., Zettlemoyer, L., Yih, S., and Lewis, M. In-context pretraining: Language modeling beyond document boundaries. *arXiv preprint arXiv:2310.10638*, 2023.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. Fever: a large-scale dataset for fact extraction and verification. *arXiv preprint arXiv:1803.05355*, 2018.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A., and Grave, E. Ccnet: Extracting high quality monolingual datasets from web crawl data. *arXiv preprint arXiv:1911.00359*, 2019.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Zhong, Z., Lei, T., and Chen, D. Training language models with memory augmentation. In Goldberg, Y., Kozareva, Z., and Zhang, Y. (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5657–5673, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.382. URL https://aclanthology.org/2022.emnlp-main.382.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

Zoph, B., Bello, I., Kumar, S., Du, N., Huang, Y., Dean, J., Shazeer, N., and Fedus, W. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A. Computational Overhead of Routing and Merging

Here we investigate the computational overhead of our causal segment routing strategy. We consider an MoE layer and an input tensor x consisting of L tokens and d dimensions: $x : L \times d$. We assume that the model uses SwiGLU as the activation function in FFNs and it up-projects the input x to d'-dimensional activations in FFNs. In this case, processing the input on an FFN requires roughly $6 \times L \times d \times d'$ FLOPs (there are two up projections and one down projections in SwiGLU-based FFNs). The overhead of soft-routing MoE comes mainly from the merging operation. Suppose that there are E experts and that the model makes a routing decision for every segment of S tokens (equivalently, there are L/S routing decisions). Each merging operation on E experts takes $6 \times E \times d \times d'$ FLOPs (we compute three merged matrices). Therefore, the total overhead will be $\frac{L}{S} \times 6 \times E \times d \times d'$ FLOPs. This indicates that compared to a dense FFN layer, an MoE layer with E experts requires $\frac{E}{S}$ more FLOPs. In our experiments, we set $S = 256$; this suggests that using $E = 8$ experts introduces 3.1% more computations and using $E = 32$ experts introduces 12.5% more computations at the FFN/MoE layers. It is worth noting that the computations from FFN layers are only a subset of the full model computations, so 3.1% is an overhead upperbound when measuring on full models. In our experiments, our most straightforward implementation leads to a 15% or 20% slowdown of training efficiency when using 8 or 32 experts. We leave a better-optimized implementation for future work.

B. Details of Similarity-based Data Batching

We adapt the pipeline of in-context pre-training (Shi et al., 2023) in our approach. Given a set of documents $\mathcal{D}$, for each document $d \in \mathcal{D}$, we first use Contriever (Izacard et al., 2021) to retrieve top-k most similar documents $N(d)$. The similarity between the document d_i and d_j is defined as the cosine similarity of their Contriever embeddings, i.e., $\text{sim}(d_i, d_j) = \cos(C(d_i), C(d_j))$, where C denotes the Contriever encoder model. We implement an efficient approximate nearest-neighbors search based on the FAISS library (Johnson et al., 2019). Then, we sort all the documents according to the similarity and construct training instances by batch consecutive documents. We use the same greedy algorithm as Shi et al. (2023). We start from a single document and repeatedly add the document that has the highest similarity value and has not been added to the list; we restart the process with a new document if all documents that are connected to the last document of the list are selected. We repeat this process until there are no documents left.

Table 3: Model architectures and sizes used in our experiments. For MoE models, we replace each FFN layers with a MoE layer. kE (e.g., “16E” in “0.3B/16E”) represents the architecture in which each FFN layer is replaced with a MoE layer of k experts. N: number of layers; D: hidden dimension of the model; n_{head}: number of attention heads.

Model	n_{params}	N	D	n_{head}
0.3B	0.3B	24	1024	16
0.3B/8E	1.8B			
0.3B/16E	3.5B			
0.3B/32E	6.8B			
1.5B	1.5B	48	1536	24
1.5B/8E	7.8B			
1.5B/16E	15.0B			
1.5B/32E	29.5B			

C. Model Configurations

In our experiments, we employ SOAP to train decoder-only models which consists of effective parameters of 0.3B and 1.5B. For each FFN layer in the Transformer model, we replace it with MoE layers with E ($E \in \{8, 16, 32\}$) experts with exactly the same architecture. Table 3 shows the configurations of model architectures.

D. Experiments on 7B models

Experimental Setups We conduct experiments on a 7B architecture. Table 4 shows the configuration of the model architectures. We train a dense 7B model and a 7B/4E MoE model. For the 7B models, we follow LLaMA2 (Touvron et al., 2023b) and use a combination of several corpora as the training set. We down-sample the full training set to a subset of 200B tokens for 7B models. Due to limited resources, we only conduct experiments on randomly batched training data for 7B models and do not apply the similarity-based batching method.

Table 4: Model architectures and sizes used in our 7B experiments. For MoE models, we replace each FFN layers with a MoE layer. N: number of layers; D: hidden dimension of the model; n_{head}: number of attention heads.

Model	n_{params}	N	D	n_{head}
7B	7B	32	4096	32
7B/4E	19.7B			

Language Modeling Results We show the training loss curves in Figure 8 and the perplexity on held-out evaluation sets in Table 6. We find that compared to the 0.3B and 1.5B models (see Section 4), the improvement of the 7B/4E model is less significant. We think it is because (1) the similarity-based batching method is not applied in this case,

Table 5: Downstream performance of using different inference methods. We study two routing strategy for inference. *prompt*: we make the routing decision once on the entire input prompt; *segment*: we re-route and get new merged FFNs every segment.

Model	PIQA	SIQA	BoolQ	HellaSwag
1.5B/8E (prompt)	**72.1**	45.2	**62.0**	43.6
1.5B/8E (segment)	**72.1**	**45.6**	60.2	**43.9**
1.5B/16E (prompt)	71.3	45.0	**56.0**	**43.7**
1.5B/16E (segment)	**72.9**	**45.4**	55.2	43.6
Model	**Wino**	**NQ**	**TQA**	**Avg**
1.5B/8E (prompt)	**63.7**	**7.3**	24.2	**45.4**
1.5B/8E (segment)	61.8	**7.3**	**24.4**	45.1
1.5B/16E (prompt)	61.5	7.3	**25.6**	44.4
1.5B/16E (segment)	**62.4**	**7.6**	25.5	**44.7**

making the experts under-utilized; (2) we only use four experts in the MoE model. We leave the experiments with the similarity-based batching method on MoE models with more experts as future work.

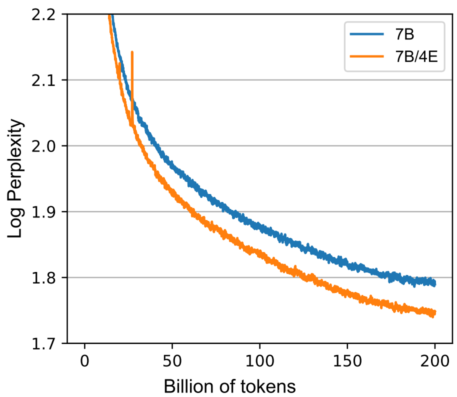


Figure 8: Training curves (log perplexity) of the 7B dense model and the 7B/4E MoE model. Note that when training the 7B/4E model, we do not apply the similarity-based batching method.

Table 6: Perplexity of trained models on different evaluation sets (arXiv, Books, Wikipedia, C4, and Python). Note that when training the 7B/4E model, we do not apply the similarity-based batching method.

Model	arXiv	Books	Wiki	C4	Python
7B	2.3	9.1	5.9	8.0	2.3
7B/4E	**2.2**	**8.7**	**5.7**	**7.7**	**2.2**

Performance on Downstream Tasks Table 7 shows the performance of the models on downstream tasks. We find that although the similarity-based batching method is not used when training the 7B/4E model, it still achieves clearly better results on various tasks compared to the dense 7B model. This further suggests the effectiveness of our causal routing strategy.

E. More Analysis and Ablation Studies

E.1. Inference Methods

During inference of downstream tasks, by default, we take the task input prompt as the input of the routers in each layer and make the routing decision once. This inference method enables the decoding process to be simple and achieves low latency, since after encoding and routing the input, we do not need to use the routers again – the rest generation can be run in a (merged) dense model. As such an inference method introduces a train-test discrepancy, we study the method that routes every segment as we do during training. As shown in Table 5, routing the input once or routing each segment does not make substantial differences in the downstream tasks we evaluate. Due to simplicity and efficiency, we use the entire prompt as the routing input and perform routing only once.

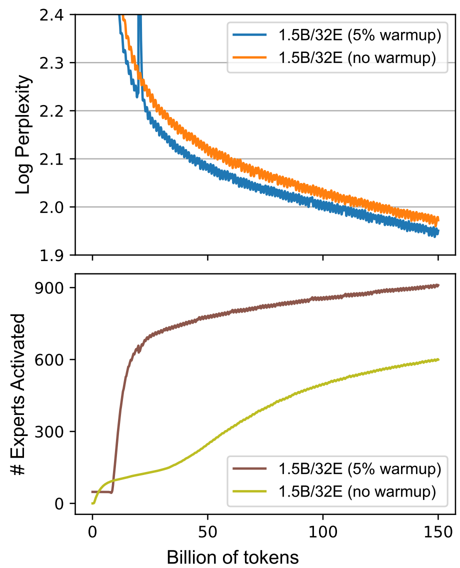


Figure 9: Training curves and expert utilization of employing a warmup phrase or not. We find without a warmup phrase, training leads to a worse MoE model (top) and worse expert utilization (bottom).

E.2. Warmup Training

At the beginning of training (i.e., the first 5% training steps), we train a dense LM with the same configuration before training the MoE model. We initialize the MoE layers by duplicating the FFN layers of the warmup trained model. We find that this warmup phase is crucial for achieving high expert utilization especially when there is a large number of experts. Figure 9 visualizes the training loss curves and expert utilization of the 1.5B/32E model (with or without warmup training). As shown in the figure, without the warm-up phrase, the model achieves worse performance and much

Table 7: We compare the 7B/4E MoE models trained with our routing strategy (without using the similarity-based batching method) with the parameter-matched dense models on downstream tasks, including commonsense reasoning, reading comprehension, closed-book QA, and text classification.

	Commonsense Reasoning					Reading Comprehension			
Model	PIQA	SIQA	BoolQ	HellaSwag	WinoGrande	RACE-m	RACE-h	ARC-e	ARC-c
7B	76.9	**50.2**	65.2	52.6	66.2	55.3	40.5	73.0	38.5
7B/4E	**77.7**	50.1	**67.6**	**54.8**	**67.3**	**57.0**	**41.3**	**73.5**	**39.6**

	Closed-book QA		Text Classification						Avg
Model	NQ	TQA	AGNews	Amazon	SST-2	Yelp	Fever	MRPC	
7B	17.3	42.5	80.6	94.3	92.7	**98.3**	53.7	67.0	62.5
7B/4E	**18.8**	**44.5**	**81.7**	**95.7**	**93.1**	96.7	**57.7**	**69.7**	**63.9**

fewer experts are utilized.