StableMoE: Stable Routing Strategy for Mixture of Experts

Anonymous ACL submission

Abstract

The Mixture-of-Experts (MoE) technique can scale up the model size of Transformers with an affordable computational overhead. We point out that existing learning-to-route MoE methods suffer from the routing fluctuation issue, i.e., the target expert of the same input may change along with training, but only one expert will be activated for the input during inference. The routing fluctuation tends to harm sample efficiency because the same input updates different experts but only one is finally used. In this paper, we propose **StableMoE** with two training stages to address the routing fluctuation problem. In the first training stage, we learn a balanced and cohesive routing strategy and distill it into a lightweight router decoupled from the backbone model. In the second training stage, we utilize the distilled router to determine the token-to-expert assignment and freeze it for a stable routing strategy. We validate our method on language modeling and multilingual machine translation. The results show that StableMoE outperforms existing MoE methods in terms of both convergence speed and performance.

1 Introduction

In recent years, large-scale Transformers (Devlin et al., 2019; Dong et al., 2019; Raffel et al., 2020; Clark et al., 2020; Bao et al., 2020; Brown et al., 2020) have shown a striking ability to model languages. However, with the model scale growing, the training speed will go slower, and the extremely large memory requirement also introduces a heavy burden of engineering. Mixture of Experts (MoE) (Jacobs et al., 1991; Jordan and Jacobs, 1994; Shazeer et al., 2017), in a much easier way, enables Transformers to scale up the number of parameters meanwhile introducing an affordable computational overhead. MoE-based Transformers have a set of expert modules, and only a few experts will be activated for each input token. In this way, we can expand the model scale by adding expert modules, which will keep the computational and memory overhead within a tolerable range.

Most existing MoE methods (Lepikhin et al., 2021; Fedus et al., 2021; Lewis et al., 2021) decide the token-to-expert routing according to the dynamically changing token representations. However, we point out that they face the routing fluctuation problem. As shown in Figure 1, the same input may be assigned to different experts along with training. However, during inference, only one expert will be activated for the input. The routing fluctuation problem tends to harm sample efficiency because the same input updates different experts while only one is finally used.

Taking BASE Layer (Lewis et al., 2021) as an example, during the whole training process, we examine the token-to-expert assignment for tokens in the validation set. For an input token, we define the last fluctuation step as the last step where its target expert is different from the final step. We plot the cumulative token percentage with regard to the last fluctuation step (annotated as its percentage accounting for all training steps) in Figure 2. We find that the last fluctuation step of 40.9% tokens exceeds 20%, which means 40.9% tokens do not have a stable target expert when 20% of all training steps have been done. Furthermore, 29.1% tokens still change their target experts after half of the whole training process, and 15.4% tokens even change the target expert after 80% of all training steps, which is nearing the training ending. These statistics prove that the routing fluctuation problem indeed exists in previous MoE methods.

In this paper, we propose **StableMoE** with two training stages to address the routing fluctuation problem. In the first training stage, we follow the learning-to-route paradigm and aim to learn a balanced and cohesive routing strategy. We design a balance loss to guarantee the assignment is balanced. In addition, inspired by Lewis et al. (2021),

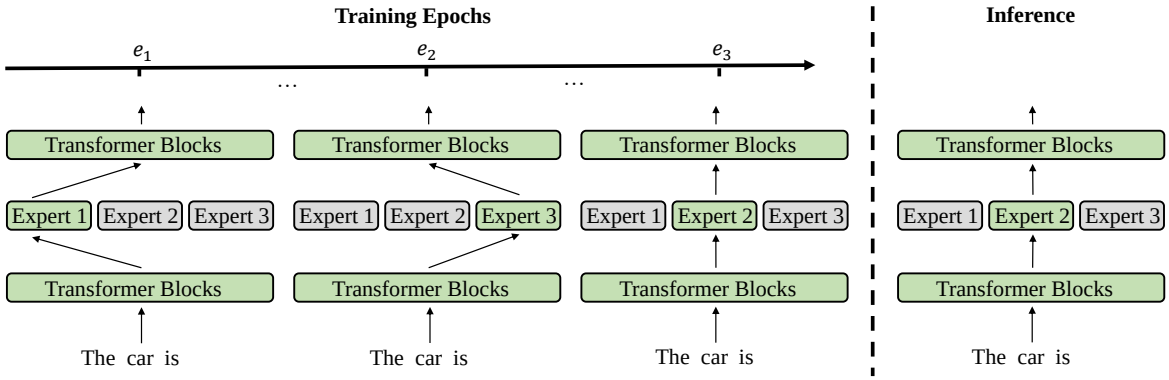

Figure 1: Illustration of the routing fluctuation problem. The same input is assigned to different experts along with training. However, during inference, only one expert is sparsely activated for the input. The routing fluctuation tends to harm sample efficiency because the same input updates different experts while only one is used.

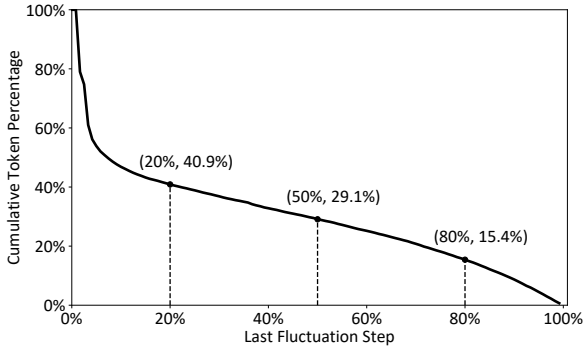

Figure 2: Cumulative token percentage with regard to the last fluctuation step of tokens for BASE Layer (Lewis et al., 2021). A substantial portion of tokens still change their target experts even if the training is nearing the end.

we adopt a sigmoid gating mechanism, which enables the task objective to propagate supervised signal back to the routing strategy, to facilitate learning a more cohesive assignment. As the routing strategy is being learned, we synchronously distill it into a lightweight router decoupled from the backbone model. In the second training stage, we utilize the distilled router to determine the token-to-expert assignment. The distilled router is frozen in this stage to provide a stable routing strategy, which addresses the routing fluctuation problem in the remaining training. We conduct experiments on language modeling and multilingual machine translation. The results show that StableMoE outperforms existing MoE methods in terms of both convergence speed and performance.

Our contributions are summarized as follows: (1) We point out the routing fluctuation problem in existing learning-to-route MoE methods. (2) We propose StableMoE to address the routing fluctuation problem. (3) We conduct substantial experiments under various settings to show the advantages of StableMoE over existing MoE methods.

2 Background: Mixture-of-Experts for Transformers

We first introduce the MoE mechanism designed for Transformers (Vaswani et al., 2017). Given a standard L-layer Transformer model and an input sequence X containing T tokens, the Transformer output H^L is calculated by

$$H^L = [\mathbf{h}_1^L; \mathbf{h}_2^L; ...; \mathbf{h}_T^L], \quad (1)$$

$$\mathbf{h}_t^l = \mathrm{FFN}\left(\mathbf{u}_t^l\right) + \mathbf{u}_t^l, \quad (2)$$

$$\mathbf{u}_{1:T}^l = \text{self-att}\left(\mathbf{h}_{1:T}^{l-1}\right) + \mathbf{h}_{1:T}^{l-1}, \quad (3)$$

where $\mathbf{h}_t^l$ is the hidden state of t-th token after the l-th layer, $\mathrm{Self\text{-}Att}(\cdot)$ is the self-attention module, and $\mathrm{FFN}(\cdot)$ is short for the feed-forward network. For simplicity, we omit the layer normalization.

We implement MoE for Transformers by inserting MoE layers, that are composed of a set of FFNs, into two neighboring Transformer blocks. At an MoE layer, for each input token, only a few or one expert will be activated, controlled by a gating function $g(\cdot)$:

$$\mathbf{h}_t^l = \sum_{i=1}^{N} g_i\left(\mathbf{h}_t^{l-1}\right) \mathrm{FFN}_i\left(\mathbf{h}_t^{l-1}\right) + \mathbf{h}_t^{l-1}, \quad (4)$$

where N is the total number of experts, and FFN_i is the i-th expert. Here, the gating function $g_i(\cdot)$ is sparse for computational efficiency. For simplicity, we omit the layer normalization.

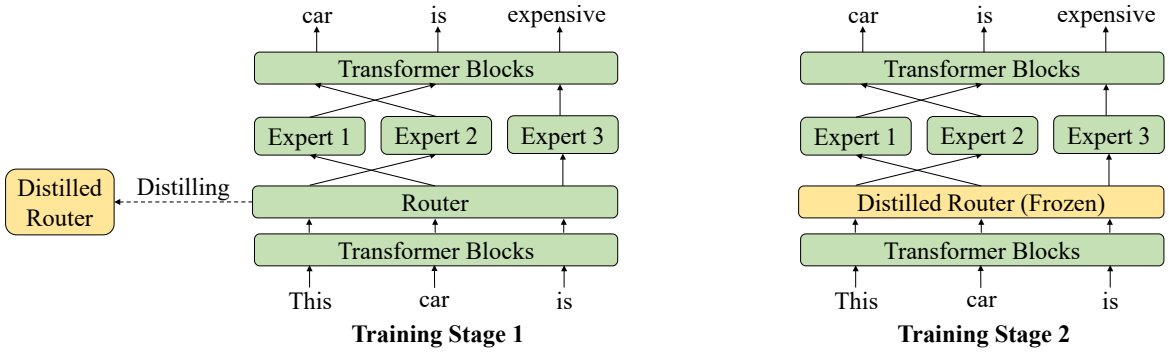

Figure 3: Illustration of two training stages in StableMoE. In training stage 1, we learn a routing strategy and distill it into a lightweight router. Then, we freeze the distilled router for stable routing in training stage 2.

3 Method

StableMoE has two training stages as illustrated in Figure 3. In the first training stage, we follow the learning-to-route paradigm and aim to learn a balanced and cohesive routing strategy. As the routing strategy is being learned, we synchronously distill it into a lightweight router decoupled from the backbone model. In the second training stage, we utilize the distilled router to determine the token-to-expert assignment. The distilled router is frozen in this stage to provide a stable routing strategy. During inference, we also use the frozen distilled router for consistent routing.

3.1 Training Stage 1: Learn Routing Strategy

Let $\mathbf{h}_t^{l-1} \in \mathbb{R}^d$ be the input representation of token t and $E \in \mathbb{R}^{N\times d}$ be the centroids of N experts. For each MoE layer, we assign each token to one expert FFN (Fedus et al., 2021; Lewis et al., 2021; Roller et al., 2021). The assignment score is:

$$s_{t,i} = E_i^\top \mathbf{h}_t^{l-1}, \tag{5}$$

where $s_{t,i}$ is the assignment score between token t and expert i, indicating their affinity. We use a greedy assignment algorithm, i.e., sending each token to the expert with the highest affinity. Then, we calculate the expert FFN output as:

$$a_t = \arg\max_i (s_{t,i}), \tag{6}$$

$$\mathbf{h}_t^l = \sigma\left(s_{t,a_t}\right) \mathrm{FFN}_{a_t}\left(\mathbf{h}_t^{l-1}\right) + \mathbf{h}_t^{l-1}, \tag{7}$$

where a_t is the the expert index that token t is sent to, and σ is the sigmoid gate (Lewis et al., 2021). Considering the sigmoid gate $\sigma\left(s_{t,a_t}\right)$, if FFN_{a_t} is beneficial for token t, optimizing the training objective (e.g., minimizing the cross-entropy loss for language modeling) will urge the gate to be greater; otherwise, the gate will tend to be smaller. The gate signal urges similar tokens to be assigned to the same expert that is beneficial to them, thus producing cohesive token-to-expert assignments.

Balance Loss We design a balance loss $\mathcal{L}_{bal}$ to avoid imbalanced assignments that will result in a high computational bottleneck in the MoE layer and thus limit the computational efficiency:

$$\mathcal{L}_{bal} = \alpha \sum_{i=1}^{N} \left((|\mathcal{A}_i| - \overline{n}) \sum_{t\in\mathcal{A}_i} \sigma\left(s_{t,i}\right) \right), \tag{8}$$

where α is a hyper-parameter, $\mathcal{A}_i$ denotes the set of tokens assigned to expert i, and $\overline{n}$ denotes the average number of tokens per expert. Intuitively, if an expert is overloaded, the balance loss will urge its assignment scores to be smaller. Otherwise, if an expert is unoccupied, the balance loss will increase its assignment scores to capture more tokens.

Distilled Router As the routing strategy is being learned, we synchronously distill it into a lightweight router decoupled from the backbone model to mimic the original routing strategy. Let X be the input sequence and $\hat{E}$ be the distilled expert centroids, we use word embeddings $\mathrm{D}(\cdot)$ to extract the routing features. We use the cross-entropy loss as the distillation loss $\mathcal{L}_{dis}$:

$$\hat{\mathbf{h}}_t^{l-1} = \mathrm{D}(X_t), \quad \hat{s}_{t,i} = \hat{E}_i^\top \hat{\mathbf{h}}_t^{l-1}, \tag{9}$$

$$\mathcal{L}_{dis} = -\sum_{t=1}^{T} \log \frac{\exp\left(\hat{s}_{t,a_t}\right)}{\sum_{i=1}^{N} \exp\left(\hat{s}_{t,i}\right)}, \tag{10}$$

where $\hat{\mathbf{h}}_t^{l-1}$ is the distilled routing feature of token t, $\hat{s}_{t,i}$ is the distilled assignment score between token t and expert i, and a_t is the expert index that token t is actually sent to. In practice, $\mathrm{D}(\cdot)$

Methods	Assignment Algorithm	Gating Function	Balance Loss
Switch Transformer	Greedy	softmax	Yes
BASE Layer	Auction (Bertsekas, 1992)	sigmoid	No
Hash Layer	Fixed Hashing	$\{0, 1\}$	No
StableMoE			
Training Stage 1	Greedy	sigmoid	Yes
Training Stage 2	Fixed Routing	sigmoid	No

Table 1: Comparison of three core elements among StableMoE and existing MoE-based Transformers.

can also be other feature extractors such as CNNs or Transformers (we investigate other variants of distilled routers in Section 4.4.3), but the word embedding is the fastest one and achieves the best performance. At the end of training stage 1, we freeze all parameters for the distilled router (i.e., $\mathrm{D}(\cdot)$ and $\hat{E}$) to prepare a stable routing strategy for training stage 2 and the inference stage.

Training Objective In training stage 1, the training loss consists of the task loss, the balance loss, and the distillation loss:

$$\mathcal{L}_{S1} = \mathcal{L}_{task} + \mathcal{L}_{bal} + \mathcal{L}_{dis}. \tag{11}$$

3.2 Training Stage 2: Learn with Stable Routing Strategy

Given frozen $\mathrm{D}(\cdot)$ and $\hat{E}$, in training stage 2, we directly use them for a stable routing strategy. Keeping other processes the same as in training stage 1, we calculate the output of the MoE layer as follows:

$$\hat{\mathbf{h}}_t^{l-1} = \mathrm{D}(X_t), \quad \hat{s}_{t,i} = \hat{E}_i^\top \hat{\mathbf{h}}_t^{l-1}, \tag{12}$$

$$\hat{a}_t = \arg\max_i (\hat{s}_{t,i}), \tag{13}$$

$$\mathbf{h}_t^l = \sigma\left(s_{t,\hat{a}_t}\right) \mathrm{FFN}_{\hat{a}_t}\left(\mathbf{h}_t^{l-1}\right) + \mathbf{h}_t^{l-1}. \tag{14}$$

Notice that the sigmoid gate $\sigma(\cdot)$ still uses original assignment score $s_{t,\hat{a}_t}$ as input, so the gate signal can also be learned in training stage 2. Since the routing strategy has been fixed in training stage 2, we no longer need the balance loss and distillation loss. Therefore, the training loss for training stage 2 contains only the task loss:

$$\mathcal{L}_{S2} = \mathcal{L}_{task}. \tag{15}$$

3.3 Inference

During inference, we also use the frozen distilled router for routing. The fixed routing strategy, which is consistent with training stage 2, makes information learned in MoE layers be utilized more thoroughly and thus leads to better performance.

3.4 Comparison with Existing MoE Methods

We compare three core elements, including the assignment algorithm, the gating function, and the balance loss, among StableMoE and existing MoE-based Transformers. In Table 1, we summarize their differences.

Assignment algorithm Switch Transformer and the training stage 1 in StableMoE simply assign each token to the expert with the highest affinity. BASE Layer adopts the auction algorithm (Bertsekas, 1992) to find a global balanced assignment with the maximum affinity sum. Hash layer and the training stage 2 in StableMoE have token-level fixed routing strategies, which have good stability.

Gating function Hash Layer uses a hard gating function, which means an expert is either fully activated or not activated, no any intermediate state. Switch Layer, BASE Layer, and StableMoE have soft gating functions, which can judge the affinity between a token and its target expert and determine a proper ratio to use the expert. Soft gating mechanisms also urge models to learn a more cohesive token-to-expert assignment.

Balance loss BASE Layer and Hash Layer do not apply any balance losses. By contrast, Switch Transformer and the training stage 1 in StableMoE design balance losses to control the balance of the token-to-expert assignment.

In summary, combing two training stages, StableMoE has a stable, cohesive, and balanced routing strategy, while the other three MoE methods cannot meet them all simultaneously.

4 Experiments

4.1 Tasks and Datasets

Language Modeling Following (Lewis et al., 2021) and Roller et al. (2021), we use the combination of the corpora in RoBERTa (Liu et al., 2019) and the English subset of the CC100 (Conneau et al., 2020) corpus. The corpus contains

Size	Models	# Shared Params	# Expert Params	FLOPs	Valid PPL	Test PPL
Base	Standard Transformer	124M	N/A	146B	23.02	22.58
	Larger Transformer (deeper)	578M	N/A	610B	17.93	17.63
	Larger Transformer (wider)	578M	N/A	610B	18.31	18.01
	Switch Transformer	124M	454M	160B	19.79	19.20
	BASE Layer	124M	454M	160B	20.04	19.69
	Hash Layer	124M	454M	160B	19.63	19.25
	StableMoE	124M	454M	160B	**19.28**	**18.93**
Large	Standard Transformer	355M	N/A	414B	18.86	18.19
	Switch Transformer	355M	3.22B	465B	16.62	16.21
	BASE Layer	355M	3.22B	465B	16.36	15.75
	Hash Layer	355M	3.22B	465B	16.37	15.79
	StableMoE	355M	3.22B	465B	**16.22**	**15.59**

Table 2: Perplexity results of language modeling. We also report the training FLOPs, and the number of parameters for the shared backbone (# Shared Params) and the expert layers (# Expert Params). "N/A" denotes not applicable. StableMoE consistently outperforms other MoE methods under both the base and the large settings.

about 100B tokens, and we randomly sample 5M tokens for validation and 20M tokens for testing.

Multilingual Machine Translation We follow Wang et al. (2020) and Ma et al. (2020) to use a collection of parallel data in different languages from the WMT datasets.[1] The dataset contains 32.5 million parallel data for language pairs between English and other 9 languages, including French (Fr), Czech (Cs), German (De), Finnish (Fi), Latvian (Lv), Estonian (Et), Romanian (Ro), Hindi (Hi), and Turkish (Tr). In our experiments, we combine the original parallel data with 180 million back-translation data as described in (Ma et al., 2020) and call the augmented dataset WMT for short.

4.2 Experimental Setup

We conduct experiments based on fairseq[2]. All experiments are conducted on NVIDIA V100 GPUs with 32 GB memory.

Language Modeling We adopt the tokenizer of GPT-2 (Radford et al., 2019), which uses byte-pair encoding (Sennrich et al., 2016) with a vocabulary size of 50,257. We set up two settings for StableMoE, a base one and a large one. For both settings, we insert one MoE layer after the middle Transformer block. We train the model for 60K steps in total (6K for training stage 1 and 54K for training stage 2). The dimension of the distilled routing features is 50, which brings 2.51M extra parameters for routing. The balance factor α is set to 0.3. We use Adam (Kingma and Ba, 2015) with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ as the optimizer. The rest of the hyper-parameters are summarized in Appendix A.

Multilingual Machine Translation Following (Ma et al., 2020), we use the SentencePiece (Kudo and Richardson, 2018) model to tokenize sentences. The vocabulary is learned from the training set and consists of 64,000 tokens. We insert two MoE layers, one after the third encoder block and one after the third decoder block. We train the model for 352K steps in total (30K for training stage 1 and 322K for training stage 2). The dimension of the distilled routing features is also set to 50. The balance factor α is set to 0.3. We use Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.98$ as the optimizer. The rest of the hyper-parameters are summarized in Appendix B.

4.3 Results

4.3.1 Language Modeling

We compare StableMoE with Switch Transformer, BASE Layer, Hash Layer, and the standard Transformer. All MoE models have the same number of shared parameters as the standard Transformer. Under the base setting, in addition, we compare two larger dense Transformers that add FFNs in a dense manner to achieve the same number of total parameters as MoE models. The deeper model stacks more FFNs, while the wider model uses FFNs with a larger hidden size. The floating point operations (FLOPs) per sequence are profiled by the torchprofile toolkit.

We show the main results of language modeling on the RoBERTa+cc100en corpus in Table 2. Under the base setting, StableMoE outperforms

[1] http://www.statmt.org

[2] https://github.com/facebookresearch/fairseq

Models	# Params	FLOPs	De	Ro	Fr	Cs	Et	Hi	Tr	Fi	Lv	Avg
Standard Transformer	77M	290B	39.8	36.0	32.5	29.1	27.2	24.5	23.6	21.8	20.3	28.31
Larger Transformer	90M	317B	40.6	36.9	33.7	29.8	27.8	25.4	24.6	22.2	20.9	29.10
Switch Transformer	480M	317B	42.3	37.1	33.8	31.0	28.6	26.0	24.3	23.0	21.2	29.70
BASE Layer	480M	317B	42.6	**37.8**	34.2	31.0	29.0	**26.9**	**25.1**	23.2	21.6	30.16
Hash Layer	480M	317B	42.7	37.0	34.6	31.3	28.7	26.5	23.9	23.1	21.7	29.94
StableMoE	480M	317B	**43.0**	37.4	**34.7**	**31.5**	**29.3**	26.8	24.7	**23.6**	**21.9**	**30.32**

Table 3: X→En test BLEU on WMT. We also report the total number of parameters, and training FLOPs. StableMoE outperforms other MoE-based Transformers across most languages.

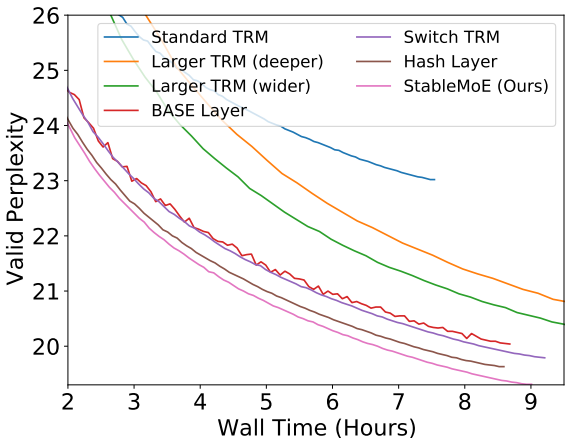

Figure 4: Convergence speed of different models. TRM is a shorthand for Transformer.

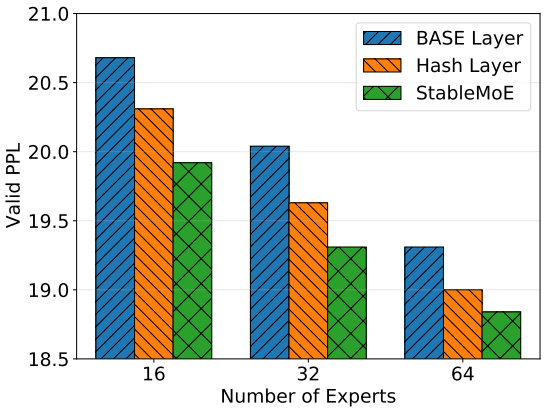

Figure 5: Comparison of MoE-based Transformers with different numbers of experts. Lower perplexity indicates better performance.

existing MoE methods on both the validation and the test sets by 0.3-0.8 perplexity. Compared with dense models, StableMoE achieves about 3.7 lower perplexity than the standard Transformer, and about 1.3 higher perplexity than the deeper larger model. Under the large setting, consistently, StableMoE outperforms the other MoE methods, and achieves about 2.6 lower perplexity than the standard Transformer.

We also compare the convergence speed of different models under the base setting. The results are plotted in Figure 4, which takes the validation perplexity as y-axis and the training wall time as x-axis. Although larger dense models achieve better validation perplexity at last, their training speed is quite slow. With regard to the convergence speed, MoE-based Transformers usually exceed dense models. Further, among the MoE methods, StableMoE has the fastest convergence speed.

4.3.2 Multilingual Machine Translation

We compare StableMoE with Switch Transformer, BASE Layer, Hash Layer, the standard Transformer, and a larger Transformer. All MoE-based models have the same number of shared parameters as the standard Transformer. Except the standard Transformer, the other models have the same FLOPs.

We translate other languages to English (X→En) and report the test BLEU on WMT in Table 3. StableMoE achieves the best average test BLEU among the compared MoE methods. Keeping the same FLOPs, StableMoE outperform the dense model by 1.22 test BLEU. With the MoE technique, we expand the number of parameters by 523% and the FLOPs just increase by 9.3%.

4.4 Analysis

4.4.1 Effects of Hyperparameters

On top of the base setting of language modeling, we investigate different settings for the MoE layers in StableMoE.

Number of Experts Figure 5 shows the results of BASE Layer, Hash Layer, and StableMoE with different numbers of experts. As the number of experts goes larger, the validation perplexity of each model tends to further descend. Consistently, StableMoE performs the best with different numbers of experts. In addition, it is worth

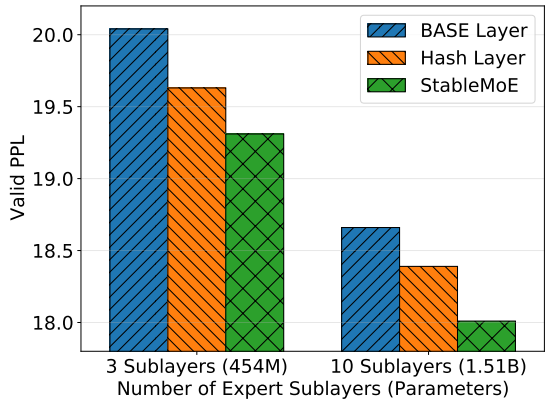

Figure 6: Comparison of MoE models with different numbers of expert sublayers (i.e., number of parameters). Lower perplexity indicates better performance.

Models	Valid PPL
StableMoE (stacked, top)	19.55
StableMoE (stacked, middle)	**19.28**
StableMoE (stacked, bottom)	22.82
StableMoE (scattered)	20.56

Table 4: Effects of the position of MoE layers. StableMoE (scattered) scatters 3 MoE sublayers uniformly into the standard Transformer, while the others stack 3 MoE sublayers together.

noting that StableMoE with 16 experts outperforms BASE Layer with 32 experts, and StableMoE with 32 experts achieves a similar perplexity to BASE Layer with 64 experts.

Number of Expert Parameters We compare MoE models with different numbers of expert parameters by setting different expert sublayers. Models with 3 and 10 expert sublayers have 454M and 1.51B expert parameters, respectively. From Figure 6, we observe that more expert parameters bring better performance, and StableMoE consistently performs the best under both settings.

Position of MoE Layers We investigate the effect of the inserting position of the MoE layer. By default, the MoE layer stacks 3 MoE sublayers and is inserted after the $\frac{L}{2}$-th Transformer block (middle). We also attempt to insert the MoE layer before the first Transformer block (bottom), and after the last Transformer block (top). In addition, we also investigate the effect if we scatter 3 MoE sublayers uniformly into the standard Transformer, i.e., after the $\frac{L}{4}$-th, $\frac{2L}{4}$-th, and $\frac{3L}{4}$-th blocks, respectively. As shown in Table 4, among the above four settings, inserting stacked MoE sublayers into the middle position allows StableMoE to achieve the best performance.

Models	Valid PPL
BASE Layer	20.04
+ Fixed Routing Strategy (Stage 2)	19.41 (0.63↓)
StableMoE with Only Stage 1	19.48
+ Fixed Routing Strategy (Stage 2)	19.28 (0.20↓)

Table 5: Effects of the fixed routing strategy.

Ratio Between Two Training Stages We investigate the balance point of the ratio between two training stages in StableMoE. Given a fixed number of total steps, allocating more steps to training stage 1 can help to learn and distill a better routing strategy. On the other hand, a larger ratio of training stage 2 means longer stable training. Under the base setting of language modeling, we attempt to allocate 6K, 15K, and 30K steps to training stage 1 and show the results in Table 6. We find that if we use word embeddings as the distilled router, allocating 6K steps (10% of the total steps) to training stage 1 is a good balance point. We speculate that the word embedding is simple enough to be learned fast, so longer stable training is more important to achieve better performance.

4.4.2 Effects of the Fixed Routing Strategy

Based on the base setting of language modeling, we design two experiments to investigate how much performance improvement the fixed routing strategy can bring. On the one hand, we equip BASE Layer with a stable routing strategy to address its routing fluctuation problem. Specifically, as in StableMoE, we use word embeddings to distill the routing strategy of BASE Layer in the first 6K training steps, and freeze the distilled router for stable routing in the remaining training. As shown in Table 5, the fixed routing strategy decreases the validation perplexity of BASE Layer by 0.63. On the other hand, we attempt to disable the training stage 2 in StableMoE and always train the model as in training stage 1. As a result, the validation perplexity of StableMoE becomes 0.20 higher than the full version that has a fixed routing strategy. These two cases support that the fixed routing strategy, which addresses the routing fluctuation problem, can bring better performance for MoE-based Transformers.

In addition, we visualize the fixed routing strategy of StableMoE in Appendix C for reference.

Distilled Routers	Stage 1 Steps	Valid PPL
Word Embedding	6K (10%)	**19.28**
Word Embedding	15K (25%)	19.34
Word Embedding	30K (50%)	19.41
CNN	15K (25%)	19.39
1-layer Transformer	15K (25%)	19.42
2-layer Transformer	15K (25%)	19.38
3-layer Transformer	15K (25%)	19.65

Table 6: Results of different ratios of two training stages and different variants of distilled routers.

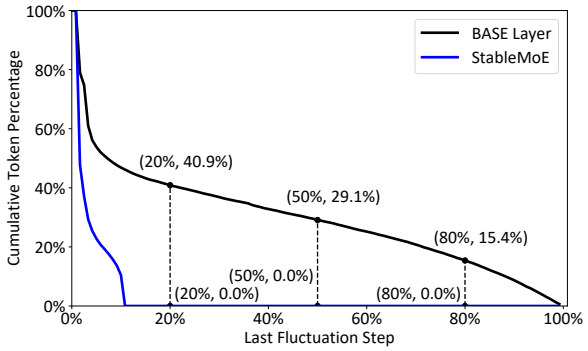

Figure 7: Cumulative token percentage about the last fluctuation step of tokens for BASE Layer and StableMoE. Notice that training stage 2 of StableMoE does not have routing fluctuation compared with BASE Layer.

4.4.3 Variants of Distilled Routers

In Table 6, in addition to word embedding, we also investigate four variants of the distilled router including CNN and three Transformers with different numbers of layers. We allocate 15K steps to training stage 1 for all of them. From the table, we find that using word embedding achieves the best performance, while the 3-layer Transformer does not perform well. For the routing strategy distillation, the distilling signal from a 32-category classification objective may not be informative enough to learn a complex router. By contrast, it is more suitable for simpler routers. Therefore, we recommend using word embedding, which is simple and effective, as the distilled router in StableMoE.

4.4.4 Analysis of Routing Fluctuations

We compare the degree of routing fluctuations between StableMoE and BASE Layer to show our advantage with regard to the routing stability. During the 60K training steps, we examine the token-to-expert assignment for tokens in the validation set every 500 steps. For each token, we define the last fluctuation step as the last step where its target expert is different from the final step. We plot the cumulative token percentage about the last fluctuation step in Figure 7. For ease of reading, we annotate the x-axis as the percentage it accounts for all training steps. From the figure, we find that the routing fluctuation problem is notable for BASE Layer. By contrast, for StableMoE, there is no routing fluctuation in training stage 2 since we apply a fixed routing strategy.

5 Related Work

Jacobs et al. (1991); Jordan and Jacobs (1994) propose Mixture of Experts (MoE) to compute different examples with independent expert modules. Shazeer et al. (2017) introduce MoE to build large-scale language models based on LSTMs (Hochreiter and Schmidhuber, 1997). Recently, as Transformers become popular, many pieces of work design MoE-version FFNs to build MoE-based Transformers. GShard (Lepikhin et al., 2021), Switch Transformer (Fedus et al., 2021), and BASE Layer (Lewis et al., 2021) follow the learning-to-route paradigm and dynamically learn how to route each input token to experts. However, we point out that these learning-to-route methods face the routing fluctuation problem. Hash Layer (Roller et al., 2021) propose a non-parametric routing strategy, which uses a pre-designed token-level hash table to determine the token-to-expert assignment. The static routing strategy will not fluctuate, but the randomly determined hash table limits the upper bound of its performance. Our work includes the advantages of learning-to-route methods to learn a balanced and cohesive routing strategy, and further addresses the routing fluctuation problem through applying a frozen lightweight router that mimics the original routing strategy.

6 Conclusion

In this paper, we point out the routing fluctuation problem that exists in previous learning-to-route MoE methods. In order to address this problem, we propose StableMoE with two training stages. We first learn a balanced and cohesive routing strategy and synchronously distill it into a lightweight router decoupled from the backbone model. Then, we freeze the distilled router for a stable routing strategy in the remaining training. We validate StableMoE on language modeling and multilingual machine translation. The results show that StableMoE outperforms existing MoE methods in terms of both convergence speed and performance.

References

Hangbo Bao, Li Dong, Furu Wei, Wenhui Wang, Nan Yang, Xiaodong Liu, Yu Wang, Jianfeng Gao, Songhao Piao, Ming Zhou, and Hsiao-Wuen Hon. 2020. Unilmv2: Pseudo-masked language models for unified language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119 of *Proceedings of Machine Learning Research*, pages 642–652. PMLR.

Dimitri P. Bertsekas. 1992. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020*. OpenReview.net.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020*, pages 8440–8451. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pages 13042–13054.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computing*, 9(8):1735–1780.

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

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 66–71. Association for Computational Linguistics.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. Gshard: Scaling giant models with conditional computation and automatic sharding. In *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. BASE layers: Simplifying training of large, sparse models. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Shuming Ma, Jian Yang, Haoyang Huang, Zewen Chi, Li Dong, Dongdong Zhang, Hany Hassan Awadalla, Alexandre Muzio, Akiko Eriguchi, Saksham Singhal, Xia Song, Arul Menezes, and Furu Wei. 2020. XLM-T: scaling up multilingual machine translation with pretrained cross-lingual transformer encoders. *CoRR*, abs/2012.15547.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

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.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. 2021. Hash layers for large sparse models. *CoRR*, abs/2106.04426.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016*. The Association for Computer Linguistics.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 5998–6008.

Yiren Wang, ChengXiang Zhai, and Hany Hassan. 2020. Multi-task learning for multilingual neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*, pages 1022–1034. Association for Computational Linguistics.

Appendix

A Hyper-parameters for Language Modeling

The hyper-parameters of STABLEMOE under the base and the large settings for language modeling are summarized in Table 7.

Hyper-parameters	**Base**	**Large**
Number of Experts	32	64
Number of MoE Layers	1	1
Sublayers per Expert	3	6
Embedding & Hidden Size	768	1024
FFN Inner Hidden Size	3072	4096
Number of Attention Heads	12	16
Number of Transformer Blocks	12	24
Sequence Length	1024	1024
Batch Size	512K Tokens	512K Tokens
Optimizer	Adam	Adam
Maximum Learning Rate	6e-4	3e-4
Learning Rate Scheduler	Linear Decay	Linear Decay
Total Steps	60K	60K
Warm-up Steps	2K	2K
Gradient Clip Norm	0.1	0.1
Dropout	0	0

Table 7: Hyper-parameters of STABLEMOE under the base and the large settings for language modeling.

B Hyper-parameters for Multilingual Machine Translation

The hyper-parameters of STABLEMOE for multilingual machine translation are summarized in Table 8.

Number of Experts	32
Number of MoE Layers	2
Sublayers per Expert	3
Embedding & Hidden Size	512
FFN Inner Hidden Size	2048
Number of Attention Heads	8
Number of Transformer Encoder Blocks	6
Number of Transformer Decoder Blocks	6
Maximum Sequence Length	256
Maximum Batch Size	512K Tokens
Optimizer	Adam
Maximum Learning Rate	5e-4
Learning Rate Scheduler	InvSqrt
Total Steps	352K
Warm-up Steps	4K
Gradient Clip Norm	0.1
Dropout	0.1
Attention Dropout	0
Label Smoothing	0.1

Table 8: Hyper-parameters of STABLEMOE for multilingual machine translation.

Experts	Most Frequent Tokens	Descriptions
5	my, his, her, year, years, day, life, week, family, days	possessive case & time units
6	with, at, from, about, them, need, want, him, against, using	prepositions & objective case
11	that, ?, !, which,)., ."., That, "., .), !!, ?", !!!, :), Â, !", ?, !, !),	conjunctions & punctuations
12	one, what, some, any, two, many, $, use, 2, 1	numerals
13	information, support, experience, service, data, services, money, access, research	nouns about technologies
17	world, government, state, country, community, city, 2018, United, US, law	nouns about politics
22	right, business, high, free, important, public, big, top, hard, small	adjectives
27	time, work, home, place, care, water, area, health, job, car	nouns about the daily life
29	ing, a, ed, in, er, on, o, e, as, es, an, al, en, am, it, is, ie, os, le	suffixes
30	you, we, they, there, It, We, here, You, ve, 've	pronouns
31	and, or, by, when, after, through, before, while, And, until	conjunctions

Table 9: The most frequent tokens assigned to each expert in the validation set. We present several representative experts. Tokens assigned to the same expert usually share some common features.

C Visualization of the Fixed Routing Strategy of StableMoE

We visualize the fixed routing strategy of StableMoE in Table 9. On the validation set, for each expert, we demonstrate the most frequent tokens assigned to it along with a text that describes their common features. We find that tokens assigned to the same expert usually share some common features, e.g., Expert 22 captures adjectives and Expert 31 captures conjunctions. These cases show good cohesiveness of the token-to-expert assignment in StableMoE.