Harder Task Needs More Experts: Dynamic Routing in MoE Models

Quzhe Huang[1*] , Zhenwei An[2*], Nan Zhuang[2*], Mingxu Tao[1],
Chen Zhang[1], Yang Jin[1], Kun Xu[3], Kun Xu[3], Liwei Chen[3],
🖂Songfang Huang[2], 🖂Yansong Feng[1]
[1]Peking University [2]Futurise AI [3]Kuaishou Technology
{huangquzhe,anzhenwei,zhuangn53}@pku.edu.cn
sfh@agibang.ai fengyansong@pku.edu.cn

Abstract

In this paper, we introduce a novel dynamic expert selection framework for Mixture of Experts (MoE) models, aiming to enhance computational efficiency and model performance by adjusting the number of activated experts based on input difficulty. Unlike existing MoE approaches that rely on fixed TopK Routing, which activates a predetermined number of experts regardless of the input's complexity, our method dynamically allocates experts based on the confidence level in expert selection for each input. This allows for more efficient utilization of computational resources, activating more experts for complex tasks requiring advanced reasoning and fewer for simpler tasks. Through extensive evaluations, our dynamic routing method demonstrates substantial improvements over Top2 Routing across various benchmarks, achieving an average improvement of 0.7% with less than 90% activated parameters. Further analysis shows our model dispatches more experts to tasks requiring complex reasoning skills, like BBH, confirming its ability to dynamically allocate computational resources in alignment with the input's complexity. Our findings also highlight a variation in the number of experts needed across different layers of the transformer model, offering insights into the potential for designing heterogeneous MoE frameworks. The code and models are available at https://github.com/ZhenweiAn/Dynamic_MoE.

1 Introduction

To effectively increase the model's parameter size, researchers have proposed the Mixture of Experts (MoE) framework (Shazeer et al., 2017; Lepikhin et al., 2021). By setting up multiple experts to enhance the model's overall capacity, MoE models selectively activate a subset of parameters for use, thereby achieving more efficient parameter utilization. With the same number of activated parameters, MoE models substantially outperform dense models in performance, achieving exceptional results in tasks such as QA and machine translation (Kim et al., 2021).

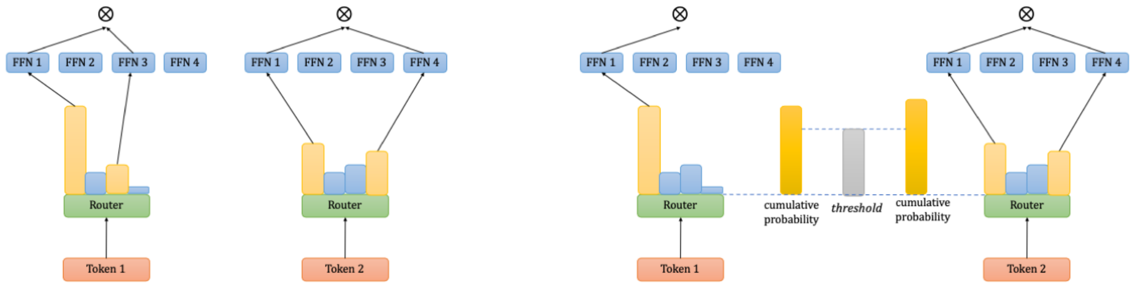
Most MoE frameworks adopt a routing mechanism that dispatches a fixed number of experts for every input (Fedus et al., 2022; Du et al., 2022). The most famous method is TopK Routing (Shazeer et al., 2017), which initially calculates the probability of each expert being suited to the current input and then activates the TopK suitable experts. Empirically, previous works (Lepikhin et al., 2021) activate two experts per token, as activating more experts offers limited improvements in model performance but substantially increases training overhead. Most of the subsequent studies (Zoph et al., 2022; Lewis et al., 2021) can be seen as variants of TopK Routing, where different constraints are introduced to ensure that the number of tokens processed by different experts is as balanced as possible. Almost all these efforts activate a fixed number of experts.

The TopK Routing, though it achieves good performance on downstream tasks, overlooks the different difficulties of inputs. Compared with simpler input, the more challenging input, e.g., tasks that require complex reasoning or logic inference, might need more parameters to solve. Dispatching experts equally across inputs could lead to computational waste on simpler tasks and insufficient computational resources for more difficult ones.

To fully leverage the potential of MoE models, we propose a dynamic routing mechanism that adjusts the number of required experts based on the confidence level in the expert selection. When the model deems the currently selected experts as insufficient, it activates more experts. Specifically, we first compute a probability distribution for selecting experts. If the highest probability for an expert exceeds a predefined threshold p, indicating high

[*] Equal Contribution. 🖂 Songfang Huang and Yansong Feng are the corresponding authors.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12883–12895
August 11-16, 2024 ©2024 Association for Computational Linguistics



(a) TopK Routing

(b) Dynamic Routing

Figure 1: Comparison between TopK routing mechanism and Dynamic Routing mechanism. (a) Each token selects fixed K=2 experts with TopK Routing probabilities. (b) In Dynamic Routing mechanism, each token selects experts with higher routing probabilities until the cumulative probability exceeds the threshold.

confidence, we activate only that one expert. Otherwise, we progressively add additional experts until the cumulative probability of the selected experts exceeds the threshold p. This approach allows for a dynamic selection of experts, with the number of experts adjusted based on the input's complexity.

Evaluation across multiple common benchmarks has revealed that our method substantially outperforms MoE models based on TopK Routing. Compared with Top2 Routing, our dynamic routing achieves an average improvement of 0.7% with less than 90% activated parameters. Further analysis has shown that our dynamic routing mechanism activates more experts in tasks requiring complex reasoning like BBH (Suzgun et al., 2023), while using fewer experts in relatively easier tasks such as Hellaswag (Zellers et al., 2019), confirming that our method indeed dynamically allocates experts based on the difficulty of the input. Token-level analysis indicates that tokens with ambiguous semantics are more challenging for the model, typically activating more experts. Another interesting finding is that the number of experts needed varies across different layers of the transformer. Lower layers require more experts for combination, while the top layer needs only one. This may relate to the *over-thinking* phenomenon (Kaya et al., 2019), which is widely observed in deep neural networks.

Our contributions can be summarized as follows:

1. We proposed a dynamic routing strategy that can adjust the number of activated experts based on the input difficulty.

2. We empirically validate that our proposed method is efficient in both training and inference, outperforming Top2 Routing while activating fewer experts.

3. We observe that for MoE models, the number of experts needed to be activated varies across different layers. This finding could help design heterogeneous MoE frameworks.

2 Method

In this section, we first briefly introduce the MoE model with TopK Routing strategy, which activates a fixed number of experts for each token. As TopK Routing ignores the varying difficulty of different inputs and the different requirements for experts at different layers, we propose a dynamic routing mechanism that adjusts the number of activated experts according to the complexity of inputs. To avoid activating too many parameters through the dynamic routing mechanism, we also introduce a dynamic loss to encourage the model to activate only the necessary experts.

2.1 TopK Routing MoE

In a Transformer model, the MoE layer is applied independently per token and replaces the feed-forward (FFN) layer of the transformer block (Lepikhin et al., 2021). For an MoE layer with N experts, $E = \{e_1, e_2, .., e_N\}$, an input $\mathbf{x} \in \mathbb{R}^d$ will be sent to the experts, where d is the hidden dimension. The output of the MoE layer is the weighted average of the experts':

$$MoE(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x}) * e_i(\mathbf{x}) \tag{1}$$

where $g_*(\mathbf{x})$ is computed by a routing network that determines the contribution of each expert to the final output. In consideration of computing efficiency, a token is dispatched to limited experts. Thus for most experts, the corresponding $g_*(\mathbf{x})$ is zero, which means that the token is not dispatched to that expert.

To obtain $g_*(\mathbf{x})$, we first compute the probability $\mathbf{P}$ of selecting each expert for input $\mathbf{x}$:

$$\mathbf{P} = Softmax(\mathbf{W_r} \cdot \mathbf{x}^T) \tag{2}$$

where $\mathbf{W_r} \in \mathbb{R}^{N \times d}$ is a learnable parameter. $\mathbf{P}$ is a vector of size N and P_i represents the probability of selecting the i^{th} expert e_i to process $\mathbf{x}$.

TopK Routing selects the k experts, whose probabilities are the highest k in $\mathbf{P}$. Then the probabilities of the selected experts are normalized and the weights of the remaining experts are set to zero, indicating they are not activated. The corresponding calculation of $g_*(\mathbf{x})$ is as follows:

$$g_i(\mathbf{x}) = \begin{cases} \frac{P_i}{\sum_{j \in TopK(\mathbf{P})} P_j}, & i \in TopK(\mathbf{P}) \\ 0, & i \notin TopK(\mathbf{P}) \end{cases} \tag{3}$$

where $TopK(\mathbf{P})$ returns the indices of the highest k elements in $\mathbf{P}$.

TopK Routing is initially proposed by (Shazeer et al., 2017), and subsequently, numerous studies have built upon it. The following works (Lepikhin et al., 2021; Zuo et al., 2022) introduce constraints aimed at ensuring a more balanced workload among the experts during training. The core of these works remains to select the most suitable experts for each token under specific constraints, based on the probability distribution $\mathbf{P}$ calculated in Equation 2. And the number of experts dispatched for each token is fixed across all these studies. Empirically, the value of k is set to 2, serving as a trade-off between training costs and model capabilities.

2.2 Dynamic Routing MoE

Although the TopK Routing strategy has shown promising performance, its assumption that an equal number of experts should be dispatched for each token overlooks the variability in difficulty across different inputs. Moreover, as a fixed number of experts are activated at every layer of the transformer, this method neglects the differences in representations across layers, potentially requiring a different number of experts for different layers.

To address these issues and make use of model parameters more efficiently, we propose a dynamic routing strategy based on model confidence. Unlike the TopK Routing, which selects a fixed number of experts, our method allows the model to assess whether the currently selected experts are sufficient. If not, it continues to activate more experts.

Algorithm 1 Expert Selection in Dynamic Routing

```
Input:
    The probability of selecting each expert, $\mathbf{P}$;
    The threshold of confidence, $p$;
    Expert Set, $\{e_1, ..., e_N\}$;
Output:
    The activated expert set, $S$;
sorted_indices I = sort(P, descending_order)
cumulative_probability = 0
for i in I do
    cumulative_probability += $P_i$
    $S = S \cup \{e_i\}$
    if cumulative_probability > $p$ then
        Break
    end if
end for
return $S$
```

Specifically, we regard that $\mathbf{P}$ in Equation 2 reflects the confidence level of input $\mathbf{x}$ in selecting different experts. In other words, P_i represents how confident the model is that the i^{th} expert can adequately handle input $\mathbf{x}$. If the highest probability in $\mathbf{P}$ is sufficiently large, then we may only need to use the corresponding expert. However, if the highest probability is not large enough, we need to add more experts to increase the reliability of processing $\mathbf{x}$.

Algorithm 1 shows how to select activated experts in Dynamic Routing. we first sort the elements in $\mathbf{P}$ from highest to lowest, resulting in a sorted indices $\mathbf{I}$. Then we find the smallest set of experts S whose cumulative probability exceeds the threshold p, where p is the threshold that controls how confident the model should be when stopping adding more experts. p is a hyper-parameter whose range is from 0 to 1. The higher the p is, the more experts will be activated.

After obtaining the set of activated experts S, the calculation of $g_*(\mathbf{x})$ in Equation 1 is:

$$g_i(\mathbf{x}) = \begin{cases} P_i & e_i \in S \\ 0, & e_i \notin S \end{cases} \tag{4}$$

2.3 Loss

Dynamic Loss There is a risk associated with our dynamic routing mechanism: it could assign low confidence to all experts, thereby activating a larger number of experts to achieve better performance.

Suppose $\mathbf{P}$ is a uniform distribution and we set the hyper-parameter p to 0.5, then the model would activate up to half of the experts. This goes against the original intention of the MoE framework, which is to scale the model with great efficiency.

To prevent dynamic routing from using too many parameters to cheat and losing its ability to selectively choose experts, we introduce a constraint on $\mathbf{P}$. We expect the routing mechanism to select a small set of necessary experts, therefore, we aim to minimize the entropy of the distribution $\mathbf{P}$, ensuring that every token can focus on as less specific experts as possible. Our dynamic loss is designed to encourage the routing mechanism to select the minimal necessary set of experts, which is formalized as:

$$Loss_d = -\sum_{i=1}^{N} P_i * log(P_i) \tag{5}$$

Load Balance Loss MoE models typically require distributed training, where different experts are deployed across various computing nodes. To avoid scenarios where some nodes are fully utilized while others are underutilized, thereby impacting training efficiency, it is generally desirable for the number of tokens processed by different experts to be roughly the same. Furthermore, a previous study (Zuo et al., 2022) has shown that evenly activated experts in an MoE layer can lead to better performance. To achieve balanced loading among different experts, we have also incorporated a load-balance loss, $Loss_b$, which is widely used in previous works (Lepikhin et al., 2021; Fedus et al., 2022)

$$Loss_b = N * \sum_{i=1}^{N} f_i * Q_i \tag{6}$$

where f_i is the fraction of the tokens choosing expert e_i and Q_i is the fraction of the router probability allocated for expert e_i. For a sequence containing M tokens, f_i and Q_i are calculated as:

$$f_i = \frac{1}{M} \sum_{j=1}^{M} 1\{e_i \in S^j\} \tag{7}$$

$$Q_i = \frac{1}{M} \sum_{j=1}^{M} P_i^j \tag{8}$$

where S^j is the set of activated experts for token j, which is calculated by Equation **??**, and P^j is the probability of selecting each experts for token j, calculated by Equation 2.

Final Loss Our model is a generative model that uses the next token generation as the training objective. We denote this loss as $Loss_{lm}$. Our final loss is a combination of the language model loss, dynamic loss, and load-balance loss:

$$Loss = Loss_{lm} + \alpha Loss_b + \beta Loss_d \tag{9}$$

where α and β are hyper-parameters to adjust the contribution of the load balance loss and dynamic loss, respectively. In our experiment, we set α as 1e-2 and β is set as 1e-4.

3 Experiments

3.1 Settings

3.1.1 Training data

We use RedPajama(Computer, 2023) as our training data, which is a fully open-source implementation of the LLaMA (Touvron et al., 2023a) training dataset. RedPajama data consists of diverse sources including the Common Crawl (CC), C4, Github, Wikipedia, books, Arxiv, and StackExchange. In our main experiments, we randomly sample 100B tokens from RedPajam and use them to train all of our models.

3.1.2 Model Settings

The model architecture follows LLaMA(Touvron et al., 2023a). We use LLaMA2 (Touvron et al., 2023b) tokenizer whose vocabulary size is 32,000. The number of transformer layers is 24 and the hidden dimension is 1024. Each MoE layer has 16 experts. Under this configuration, the dense model has approximately 374M parameters. Each MoE model has 3.5B total parameters. Only 374M parameters are activated in MoE-Top1 and 581M parameters are activated in MoE-Top2. More detailed model and training settings are shown in Appendix A.

3.1.3 Evaluation

We use opencompass[1] to evaluate our model. Specifically, we adopt a 3-shot evaluation for the BBH dataset and a 0-shot evaluation for the rest.

3.1.4 Experiment Models

We train several variants of our architecture from scratch using the above model settings.

[1] https://github.com/open-compass/OpenCompass/

	Dense(374M)	Dense(570M)	MoE-Top1	MoE-Top2	MoE-Dynamic
PIQA (Bisk et al., 2020)	64.3	65.9	67.3	**68.1**	**68.1**
Hellaswag (Zellers et al., 2019)	36.1	39.6	42.3	43.9	**44.3**
ARC-e (Bhakthavatsalam et al., 2021)	37.9	37.6	39.5	**40.4**	39.9
Commonsense QA (Talmor et al., 2019)	32.2	31.7	30.3	32.1	**33.6**
BBH (Suzgun et al., 2023)	22.3	22.1	23.0	23.3	**25.6**
Average	38.6	39.4	40.5	41.6	**42.3**

Table 1: Performance on downstream tasks. The best result for each task is emphasized in **bold**.

Dense We use dense models as our baseline. In dense models, each transformer layer is composed of a multi-head attention layer and a standard Feed Forward Network. We implement two Dense models: Dense(374M) and Dense(570M) by setting the hidden dimensions to 1024 and 1280, respectively.

MoE-Top1 / Top2 The MoE models with TopK Routing, where K = 1 and 2 respectively. Only language modeling loss, $Loss_{lm}$, and load-balance loss $Loss_b$ are used for training. The MoE-Top1 could be seen as a re-implementation of Switch Transformer (Fedus et al., 2022) and the MoE-Top2 is a re-implementation of Gshard(Lepikhin et al., 2021). The activated parameters of MoE-Top1 and MoE-Top2 are nearly the same as Dense(374M) and Dense(570M) respectively.

MoE-Dynamic MoE-Dynamic model uses our dynamic routing mechanism, activating a various number of experts depending on the input token representation. The threshold p in our routing mechanism is 0.4. During inference, MoE-Dynamic model activates no more than 2 experts, which means it uses fewer parameters than MoE-Top2.

3.2 Main Results

Table 1 shows the performance of different models on downstream tasks. Overall, the MoE models outperform the Dense models. Among all the MoE variants, our proposed Dynamic MoE demonstrates the best performance, achieving at least a 0.7% higher score on average compared to other models.

We first compare models with an equal number of activated parameters. It is observed that MoE-Top1 outperforms the Dense model with 374M parameters by an average of 1.9% score, and MoE-Top2 surpasses the Dense model with 570M parameters by 2.2% score. This indicates that, with the same number of activated parameters, MoE models substantially outshine their Dense counterparts.

When comparing models with the same architecture, we generally observe a positive correlation between model performance and the number of activated parameters. For Dense models, the model with 570M parameters outperforms the model with 374M parameters by 0.8% score on average. Similarly, among models using the MoE architecture with a fixed number of activated experts, MoE-Top2 reaches an average of 41.6% score and outperforms MoE-Top1 by 1.1% score. In fact, MoE-Top2 performs better than MoE-Top1 in all subtasks, demonstrating the rule of more parameters leading to better performance.

However, our proposed Dynamic Routing mechanism breaks this rule. As shown in Table 3, the average number of activated experts in the MoE-Dynamic during evaluation phases is less than two, meaning it activates fewer parameters than MoE-Top2. Yet, as shown in Table 1, compared to MoE-Top2, MoE-Dynamic achieves comparable or even better performance on nearly all the tasks and outperforms MoE-Top2 by 0.7% score on average. MoE-Dynamic obtains better performance, indicating that our dynamic routing mechanism can allocate the necessary experts for different inputs more reasonably and make use of parameters more efficiently.

4 Efficiency of Dynamic Routing

The greatest advantage of MoE models is their ability to efficiently scale to larger models. The TopK Routing mechanism controls the number of parameters used by the entire model by activating a fixed number of experts. In contrast, our proposed dynamic routing mechanism removes the limitation of a fixed number of experts. Naturally, there may be concerns that our method might assign too many experts to each token. Another possible concern is that our dynamic routing mechanism might cost too many resources and make the whole model inefficient. To address these concerns, we demonstrate the efficiency of the dynamic routing mechanism from both training and inference perspectives.

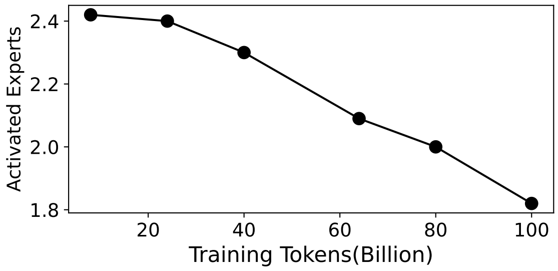


Figure 2: Average activated experts number across training procedure.

4.1 Efficient Training

We sample 1000 pieces of text from different sources within Redpajama and calculate the average number of experts activated per token at different stages of training. Figure 2 shows the change in the average number of activated experts throughout the training process. As shown in the figure, we can observe that the number of experts activated per token decreases over time. In the early stages of training, dynamic routing assigns more experts to each token, but after 60B tokens, the average number of activated experts is already less than 2. Table 2 displays the number of experts activated by MoE-Dynamic at the end of the 100B training. It could be seen that across all data sources, the number of experts activated by MoE-Dynamic is less than 2.

Recently, the amount of tokens used in training for large language models far exceeds 100B, for instance, Pythia uses 300B tokens, and Llama2 uses 2T tokens. If we continue to train on an even larger scale corpus, the average number of parameters used throughout the training process is guaranteed to be lower than that of Top2-Routing.

Sources	Ratio	Activated Experts
CC	67%	1.82
C4	15%	1.84
Github	4.5%	1.88
Wiki	4.5%	1.78
Book	4.5%	1.73
Arxiv	2.5%	1.90
StackExchange	2%	1.79
Avg	100%	1.82

Table 2: Average activated experts in different parts of the training corpus.

4.2 Efficient Inference

To further explore whether our proposed method is efficient in inference, we calculate the average number of experts activated by the model across different downstream tasks. For every question, we use the template from the evaluation to concatenate the question with the gold answer into a complete input and truncate the tokens exceeding 2048 to fit our model's maximum input length. Table 3 shows the average number of experts activated per token across various downstream tasks. The result is averaged across all the layers of transformers and it is evaluated using the checkpoint trained on 100B tokens.

Sources	Activated Experts
PIQA	1.72
Winogrande	1.76
ARC-e	1.73
Commonsense QA	1.74
BBH	1.87
Avg	1.76

Table 3: Average activated experts in different downstream tasks.

From the table, we can observe that across all five downstream tasks, the number of activated experts is less than two. The model activates 1.76 experts on average, which is fewer than the fixed activation of two experts by the Top2 Routing method. During the training phase, our method and Top2 Routing are comparable in efficiency, but upon completion of training, our inference efficiency substantially outperforms Top2 Routing. Given that models are mostly trained once and deployment costs are the biggest burden nowadays, the advantages of our method over static MoE routing mechanisms like Top2 become even more apparent.

4.3 Influence of Routing Module is limited

Compared with TopK Routing, our proposed method will introduce extra computation during routing. However, the computation cost of the routing module is limited compared with the whole transformer and our proposed method is more efficient than Top2 Routing in reality.

Table 4 shows the throughput of MoE-Dynamic and MoE-Top2, both of which are trained with 100B tokens. As shown in the table, MoE-Dynamic is about 5% higher in throughput than MoE-Top2 during both training and inference, effectively validating that our Dynamic Routing is indeed more efficient.

	MoE-Dynamic	MoE-Top2
Training(K token/s)	98.5	93.9
Inference(Sample/s)	0.19	0.20

Table 4: Throughput of MoE models trained with 100B tokens. Training throughput is tested using 8 * A800 on the training dataset. Inference throughput is tested using a single A800 on BBH dataset.

5 What is Challenging Input?

The motivation for designing dynamic routing is to enable the model to adjust the number of allocated experts based on the difficulty of the input. In this section, we will explore what kinds of inputs are considered challenging for the model from various perspectives.

5.1 Tasks Requiring Reasoning

From Table 3, we could observe that solving the BBH task requires activating an average of 1.87 experts, more than the number needed for other tasks. BBH, which stands for BIG-Bench Hard, is a suite of 23 challenging BIG-Bench tasks. These tasks demand capabilities such as multi-hop reasoning, causal inference, logical deduction, and so on, making them substantially more difficult than normal NLP tasks (Suzgun et al., 2022). Our model's use of more experts on BBH tasks implies that our method can dynamically monitor task difficulty and apply more parameters to tackle more challenging tasks. Interestingly, as shown in Table 1, MoE-Dynamic, compared to MoE-Top2, sees the most improvement on BBH tasks. While the average improvement across all tasks is less than 1.0%, the improvement on BBH is more than 2.0%, which is more than double that of other tasks. This further illustrates that dynamically adjusting the number of activated experts is beneficial for solving downstream tasks, especially more challenging ones.

Task	Activated Experts
Tracking shuffled objects	
—- Object Number K = **3**	1.959
—- Object Number K = **5**	1.963
—- Object Number K = **7**	1.970
Logical deduction	
—- Object Number K = **3**	1.943
—- Object Number K = **5**	1.947
—- Object Number K = **7**	1.953

Table 5: Average activated experts in the same task with different elements.

5.2 Tasks with More Elements

In BBH, there are two sets of tasks, each containing three sub-tasks with the same goal but varying elements. We evaluate whether the number of experts activated for different sub-tasks is proportional to the number or elements.

The first task is Tracking shuffled objects: Given the initial positions of a set of K objects and a series of transformations (namely, pairwise swaps) applied to them, determine the final positions of the objects. The sub-tasks vary in the number of objects K each contains. The second task is Logical deduction: Deduce the order of a sequence of K objects based on the clues and information about their spatial relationships and placements. The sub-tasks also vary in the number of objects K each contains.

The table below shows the number of parameters activated by MoE-Dynamic on the corresponding tasks. From the table, we can see that as the number of objects increases, the number of parameters activated by the model also gradually increases.

5.3 Tokens with Uncertain Meaning

	Examples	C-Words Ratio
Most Experts	tr, eq, mu, frac	10
Least Expers	to, that, and, show	51

Table 6: The first column shows examples of tokens requiring the most experts and least experts. The last column shows the complete word ratio in these two groups of tokens.

To further analyze what types of tokens are considered more challenging for a model, we examine the average number of experts activated for each token in the vocabulary across different contexts.

We sample 1 million tokens from the training dataset Redpajama, resulting in a new corpus of a total of 7 million tokens. In this corpus, we calculate the average number of experts activated for each token in the vocabulary. To minimize the effect of randomness, we only consider tokens that appear no less than 10,000 times in the new corpus.

Table 6 shows the number of complete words among the top 100 and bottom 100 tokens by the average number of experts activated, along with some examples.

Upon manually reviewing the 100 tokens that activate the most experts and the 100 tokens that activate the least, we observe an interesting phenomenon: Tokens with relatively definite meaning are considered easier by the model, activating fewer experts. In contrast, tokens with uncertain meanings are deemed more challenging and require more experts for processing.

Specifically, since our model's tokenizer is trained with Byte Pair Encoding (BPE), many tokens are not complete words but subwords. These subwords have vaguer meanings compared to complete words because they can combine with many other subwords to form words with different meanings. For example, the subword 'tr' can lead to the formation of hundreds of words with varied meanings, such as tree, triple, train, trick, trouble, and so on. Due to the multitude of possible meanings, different meanings may require different experts to process, making such subwords require a comprehensive understanding by more experts.

6 Bottom Layers Need More Experts

An intriguing observation from our study is that our model achieves superior performance while activating fewer parameters. As shown in Table 3, on all the tasks, our MoE-Dynamic activates an average of fewer than two experts. But it outperforms the MoE-Top2 in downstream tasks as shown in Table 1. This result is quite surprising, as performance on downstream tasks is typically correlated with the quantity of activated parameters.

This unexpected phenomenon might be attributed to our method's more proper allocation of the experts across different layers, employing more experts at the bottom layers and fewer at the top. This layer-wise dynamic allocation, as opposed to the fixed number of experts per layer, somewhat mitigates the common issue of overthinking in deep neural networks, thereby enhancing performance.

The overthinking refers to the situations where simpler representations of an input sample at an earlier layer, relative to the complex representations at the final layer, are adequate to make a correct prediction (Kaya et al., 2019). Previous works (Liu et al., 2020; Schwartz et al., 2020; Xin et al., 2021) have demonstrated that shallower representations can achieve comparable, if not better, performance across various tasks than deeper representations. This could be due to deeper representations overfitting specific distributions, lacking generalizability, and being more vulnerable to attacks (Hu et al., 2019; Zhou et al., 2020). It suggests that in some cases, acquiring a better shallow representation is more valuable than obtaining a more complex deep representation, which correlates to previous findings that removing top layers has a limited impact on the downstream tasks (Sajjad et al., 2023).

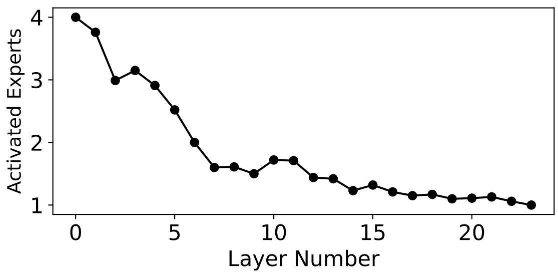


Figure 3: Activated experts in different layers

Compared with Top2 Routing, our Dynamic Routing activates more experts at the bottom layers to obtain better shallow representations and use the simpler network in the top layers to alleviate the overthinking issue. Figure 3 displays the number of experts activated per token at different layers[2]. From the figure, we observe a gradual decrease in the average number of experts activated per token with the increase in layer depth. The lowest layer activates the most experts, up to 4 experts per token, enabling better shallow representations through a wider network, which is beneficial for various downstream tasks. At the topmost layer, the number of activated experts per token is reduced to even one. This phenomenon can prevent the model from being too complex and preserve generality in the final representation.

It should be noted that we do not argue allocating most parameters to the bottom layers is the optimal strategy for a model. It is possible that one model could achieve better performance if more experts are activated in the intermediate layers. In our experiment, we observe the phenomenon that *bottom layers need more experts* and we think this is quite interesting and worthy of future exploration. Beyond the series of overthinking works that we mention above (Kaya et al., 2019; Liu et al., 2020), our finding also correlates with the conclusion that the upper layers of the model are tied to pre-training tasks (Zhang et al., 2020; Tao et al., 2024). Maybe, because the upper layer is specialized for the task of the next token prediction, the topmost layers only activate just one expert.

[2] The results are evaluated using a checkpoint trained on 100B tokens.

7 Related Work

The Mixture of Experts (MoE) model is initially introduced by (Jacobs et al., 1991). Recent studies have demonstrated sparsely gated MoE models have substantial improvements in model capacity and efficiency, enabling superiors performance than dense models(Shazeer et al., 2017). Particularly MoE has shown great potential with the integration of transformer architectures (Zoph et al., 2022; Jiang et al., 2024).

In previous MoE architectures, a static number of experts are activated regardless of the varying complexity presented by input tokens. Most of MoE models activate Top1 or Top2 experts (Lepikhin et al., 2021; Fedus et al., 2022). There are many works (Roller et al., 2021) make improvements based on the Top2 Routing (Lepikhin et al., 2021). They make great efforts to distribute tokens among different experts more evenly during the training process and all of them activate static experts for each token. We choose Top2 Routing (Lepikhin et al., 2021) as the baseline because it's currently the most widely used method and the only one successfully applied in large language models (LLM) thus far. For instance, Mixtral-MoE (Jiang et al., 2024), DeepSeek-MoE (Dai et al., 2024) and GroK-MoE (X.AI, 2024) all use this structure.

There are some work that allocating different numbers of experts to different tokens. Expert-Choice MoE model selects TopK tokens for each expert(Zhou et al., 2022) and each token will be allocated with a different number of experts. However, on average every token activates exactly two experts, which is the same with Top2 Routing models and does not save computation. Different from the previous works, our dynamic routing mechanism can activate fewer parameters on average by dynamically allocating experts to different tokens, which makes our method more efficient.

Recent works point out that in language modeling not all tokens require the same calculation cost (Raposo et al., 2024) and tasks across different domains require varying numbers of experts(Lu et al., 2024). Our findings that *harder task needs more experts* are correlated with this line of work.

8 Conclusion

Our paper introduces a dynamic expert selection framework for Mixture of Experts (MoE) models, surpassing existing static TopK Routing by adjusting expert activation based on input complexity. Our approach not only improves computational efficiency but also model performance, evidenced by obvious gains over TopK Routing in our evaluations. Our findings reveal the framework's effectiveness at dynamically dispatching different numbers of experts, particularly for complex reasoning tasks, and suggest the potential for developing more challenging heterogeneous MoE models. In support of further research, we will open-source our models, contributing to advancements in the MoE domain.

Limitation

Due to resource constraints, the size of the model we trained is limited, with only about 600M activation parameters, and the entire MoE (Mixture of Experts) model being just over 3B in size. However, (Dai et al., 2024) has validated that within the MoE framework, conclusions drawn from smaller models can be generalized to larger models with more parameters. Hence, we believe our proposed dynamic routing method could also be effective in larger-scale models. Additionally, we have only trained on 100B tokens, which may not be sufficient for model training. Yet, given the same scale of training data, our method demonstrated superior performance, which also underscores the efficiency of our training process.

Acknowledgments

This work is supported in part by NSFC (62161160339) and Beijing Science and Technology Program (Z231100007423011). We thank the anonymous reviewers for their valuable suggestions. For any correspondence, please contact Yansong Feng.

References

Sumithra Bhakthavatsalam, Daniel Khashabi, Tushar Khot, Bhavana Dalvi Mishra, Kyle Richardson, Ashish Sabharwal, Carissa Schoenick, Oyvind Tafjord, and Peter Clark. 2021. Think you have solved direct-answer question answering? try arc-da, the direct-answer AI2 reasoning challenge. *CoRR*, abs/2102.03315.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI*

2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 7432–7439. AAAI Press.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *CoRR*, abs/2401.06066.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen S. Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23:120:1–120:39.

Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. 2019. Triple wins: Boosting accuracy, robustness and efficiency together by enabling input-adaptive inference. In *International Conference on Learning Representations*.

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

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR.

Young Jin Kim, Ammar Ahmad Awan, Alexandre Muzio, Andrés Felipe Cruz-Salinas, Liyang Lu, Amr Hendy, Samyam Rajbhandari, Yuxiong He, and Hany Hassan Awadalla. 2021. Scalable and efficient moe training for multitask multilingual models. *CoRR*, abs/2109.10465.

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, Virtual Event, Austria, May 3-7, 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, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044.

Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. 2024. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. *arXiv preprint arXiv:2402.14800*.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. 2021. Hash layers for large sparse models. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 17555–17566.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6640–6651.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

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, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4149–4158. Association for Computational Linguistics.

Mingxu Tao, Quzhe Huang, Kun Xu, Liwei Chen, Yansong Feng, and Dongyan Zhao. 2024. Probing multimodal large language models for global and local semantic representations. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*, pages 13050–13056. ELRA and ICCL.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv e-prints*, pages arXiv–2307.

X.AI. 2024. Grok-1. https://github.com/xai-org/grok-1?tab=readme-ov-file.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. Berxit: Early exiting for bert with better fine-tuning and extension to regression. In *Proceedings of the 16th conference of the European chapter of the association for computational linguistics: Main Volume*, pages 91–104.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. In *International Conference on Learning Representations*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. 2022. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

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

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. 2022. Taming sparsely activated transformer with stochastic experts. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

	Dynamic		Top1	Top2	Top2-Inference-with-Dynamic			
Threshold p	0.4	0.5	-	-	0.1	0.2	0.3	0.4
Activated Experts	1.8	2.3	1.0	2.0	1.1	1.9	2.8	3.9
Avg Performance	42.3	42.8	40.5	41.6	40.0	40.3	42.1	42.8

Table 7: Experiment results about threshold p. The first two columns present results of Dynamic MoE models trained separately with $p = 0.4$ and $p = 0.5$. The last four columns reports results of applying dynamic routing mechanism as a post-hoc method to MoE-Top2.

A Detailed Training Setting

A.1 Model Setting

The model architecture follows LLaMA(Touvron et al., 2023a). We use Llama2 tokenizer whose vocabulary size is 32000. Unless specifically stated otherwise, we set the number of transformer layers to 24, and the hidden dimension to 1024. We employ the multi-head attention mechanism with a total of 16 attention heads, where each head has a dimension of 64. We use SwiGLU(Shazeer, 2020) in FFN layers. For initialization, all learnable parameters are randomly initialized with a standard deviation of 0.006. Each MoE layer has 16 experts, which have the same initialized parameters as a standard FFN. Under this configuration, each dense model has approximately 374M parameters. Each MoE model has 3.5B total parameters. Only 374 parameters are activated in MoE-Top1 and 581M parameters are activated in MoE-Top2.

A.2 Training Setting

We adopt the AdamW optimizer with first-moment decay $\beta_1 = 0.9$ and second-moment decay $\beta_2 = 0.95$. The weight decay is 0.1. The learning rate warms up from 0 to 3e-4 in the first 2000 steps and decays in the remaining steps using the cosine decay schedule to 3e-5. We set the context length to 2048 and adopt the batch size of 2048. We use Megatron-LM serving as the backbone of our training infrastructure. Our configuration includes a tensor parallelism of 1 and a pipeline parallelism of 2. During training, we use at most 128 NVIDIA A800 GPUs.

B Analysis about experiments details

B.1 Influences of threshold p

Threshold p is a hyper-parameter used to control the dynamic routing mechanism. In this section, we illustrate that Dynamic Routing method is not very sensitive to the choosing of p in both training and inference.

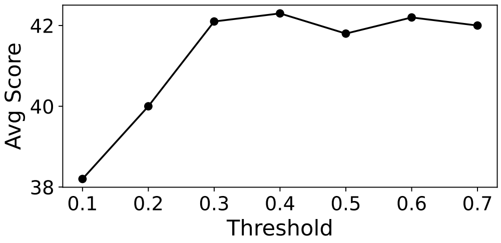


Figure 4: Average scores of MoE-Dynamic($p = 0.4$) with different threshold p in inference on downstream tasks

We train Dynamic MoE models based on threshold $p \in \{0.4, 0.5\}$. Experiment results in Table 7 show that both settings outperform MoE-Top2, which indicates that dynamic MoE is not very sensitive to the choosing of p during training. We are not able to try other p as training models from scratch with different values of p is very resource-intensive

We also evaluate the model with different p during inference. Figure 4 demonstrates the average performance on downstream tasks with different p of Dynamic-MoE trained with $p = 0.4$. The figure reveals that when p is too low, like 0.1 and 0.2, the model's performance on downstream tasks decreases dramatically due to activating too few experts. Conversely, once p surpasses a certain threshold, like 0.3, the model's performance stabilizes, and the influence of this parameter on downstream tasks will become minimal.

B.2 Post-hoc method for TopK?

Another interesting question is whether we can use a model trained with TopK routing but run the inference with Dynamic Routing, which is like a post-hoc pruning for efficiency. As most public MoE models are trained with TopK Routing, we hope to apply our Dynamic Routing in these models directly and improve their efficiency.

Table 7 shows the performance of a model trained with Top2 Routing and inference with Dynamic Routing. As shown in the table, we can control the number of activated experts by chang-

	Activated Experts	Avg Performance
MoE-Dynamic	1.8	42.8
w/o $Loss_d$	2.0	40.0

Table 8: Experiment results of Ablation study on Dynamic Loss. The first column presents average number of activated experts for each token. The second column reports models' average performance on downstream tasks in Table 1.

ing the threshold p during inference. We find that when using proper p, e.g. 0.3 and 0.4, inference with Dynamic Routing could outperform inference with Top2 Routing. But this is at the cost of activating more parameters. When using p=0.2 for inference, the model activates 1.9 experts on average, which is slightly fewer than MoE-Top2. At this time, the model's performance is inferior to the performance of MoE-Top2. It implies that direct inference with Dynamic Routing using MoE-Top2 model may not work.

B.3 Ablation On Dynamic Loss

We conducted an ablation study using the same experimental settings as MoE-Dynamic. As shown in Table 8, the removal of Dynamic Loss leads to a noticeable increase in the number of activated experts and a significant drop in performance on downstream tasks. This finding indicates that the proposed loss function enhances the model's efficiency and effectiveness. The Dynamic Loss likely achieves this by compelling the model to allocate the most suitable experts for each token.