MoE*fication*: Conditional Computation of Transformer Models for Efficient Inference

Anonymous ACL submission

Abstract

Transformer-based pre-trained language models achieve superior performance on most NLP tasks due to large parameter capacity, but also lead to huge computation cost. Fortunately, we observe that most inputs only activate a tiny ratio of neurons of large Transformer-based pre-trained models during inference. Hence, we propose to convert a model into its mixture-of-experts (MoE) version with the same parameters, namely MoE*fication*, which accelerates large-model inference by conditional computation based on the sparse activation phenomenon. Specifically, MoEfication consists of two phases: (1) splitting the parameters of feed-forward neural networks (FFNs) into multiple parts as experts, and (2) building expert routers to decide which experts will be used for each input. Experimental results show that MoEfication can save 80% computation cost of FFNs while maintaining over 95% original performance for different models, including models with different sizes (up to 3 billion parameters) and distilled models, on various downstream tasks. Moreover, we find that the MoEfied model achieves better performance than the MoE model pre-trained from scratch with the same model size. We will release all the code and models of this paper.

1 Introduction

Recent years have witnessed an exponential increase in the size of Transformer-based pre-trained language models (PLMs) (Han et al., 2021). From BERT (Devlin et al., 2019) in 2018 to GPT-3 (Brown et al., 2021) in 2020, the number of parameters has already increased by nearly 600 times. Moreover, the exploration of larger models is continuing. The increasing model size significantly improves the model performance on a variety of downstream NLP tasks (Raffel et al., 2020; He et al., 2021b), but also comes with huge computation cost, which limits the potential applications of large-scale PLMs. Hence, it is essential to explore novel techniques to make PLMs more efficient.

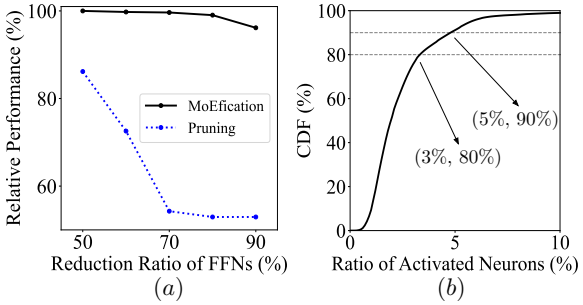


Figure 1: Results of a fine-tuned T5-Large (Raffel et al., 2020) on SST-2. (a) Relative performance compared to the original performance with different reduction ratios of FFNs. Large pruning ratios significantly degrade the performance. (b) Cumulative distribution function (CDF) of the ratio of activated neurons for each input. SST-2's training set is used as inputs. 90% inputs only activate less than 5% neurons of FFNs.

The computation of Transformer mainly consists of two parts: attention networks and feed-forward networks (FFNs). Much effort has been made to reduce the cost of attention networks (Beltagy et al., 2020; Kitaev et al., 2020; Tay et al., 2020) while little has been made for FFNs. Previous work on the acceleration of FFNs usually uses general pruning algorithms and ignores the characteristics of FFNs (Li et al., 2020; Xu et al., 2021). Hence, large pruning ratios will lead to poor results as shown in Figure 1. In this work, we explore to further accelerate FFNs beyond model pruning.

Fortunately, according to our observation on FFNs in Transformer models, we find a phenomenon of **sparse activation**, i.e., only a tiny fraction of neurons are activated for a single input. As shown in Figure 1, when we perform inference on a fine-tuned T5-Large model with 700-million parameters, 90% inputs only activate less than 5% neurons[1]. Hence, we can omit the computation of

[1]T5 uses ReLU as the activation function. We treat the neurons having positive outputs as activated neurons.

inactive neurons to reduce the cost. Meanwhile, most neurons will be eventually activated by some inputs. As a result, model pruning is not applicable and will significantly degrade the performance. Instead of model pruning, we explore efficient FFNs based on conditional computation (Bengio, 2013), which selectively activates parts of the network according to input. This mechanism **naturally** exists in the FFNs of pre-trained Transformers.

Inspired by the sparse activation phenomenon, we propose to convert a large-scale PLM into its mixture-of-experts (MoE) version with the same parameters for efficient conditional computation in inference, namely MoE*fication*. Different from previous work on MoE Transformers that typically *breeds* models into multiple experts (Lepikhin et al., 2021; Fedus et al., 2021), MoEfication aims to *split* existing models into multiple experts while keeping the model size unchanged. We expect an MoEfied model will improve the model efficiency and maintain the performance of the original model by dynamically selecting experts.

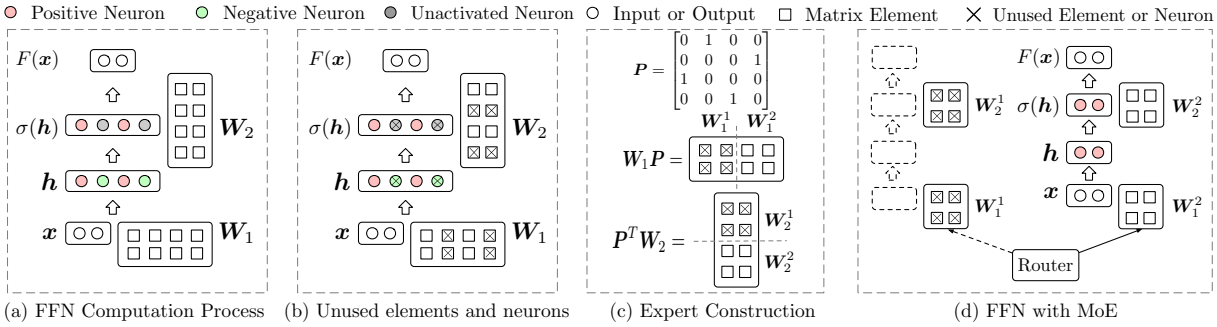
MoEfication consists of two phrases. (1) **Expert Construction**: Split a whole feed-forward layer into multiple experts. The goal is to group those neurons that are often activated simultaneously into the same expert network. To achieve this goal, we build a co-activation graph based on the activation results and divide this graph into several subgraphs as experts by graph partition. (2) **Expert Selection**: Select those experts that contain as many activated neurons as possible for each input to approximate to the original results. To reach this target, we first find the best selections based on the activation results and then use them to train shallow neural networks as expert routers.

In the experiments, we validate the effectiveness of MoEfication on two typical kinds of downstream tasks, including GLUE and QA benchmarks (Wang et al., 2019; Rajpurkar et al., 2016; Lai et al., 2017), using T5 with different sizes (Raffel et al., 2020). Experimental results show that MoEfication can save 80% computation cost of FFNs while maintaining over 95% original performance for both conventional models (the number of parameters varies from 60 millions to 3 billions) and distilled models. Besides, we find that the MoEfied model achieves better performance than the MoE model pre-trained from scratch with the same model size. Then, we study the routing patterns of MoEfied models and hope these findings can help future work on the design and training of MoE models.

2 Related Work

Model Acceleration for PLMs. Model acceleration aims to reduce the time and space complexity of PLMs for faster inference and deployment on resource-constrained devices. There are several techniques for model acceleration, including knowledge distillation (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), model pruning (Voita et al., 2019; Michel et al., 2019; Zhang et al., 2021), model quantization (Zafrir et al., 2019; Zhang et al., 2020), and dynamic inference (Xin et al., 2020; Li et al., 2021). Among these techniques, model pruning and dynamic inference explore to omit unnecessary computation for acceleration, which is similar to the target of MoEfication. Different from model pruning, which omits redundant parameters, MoEfication keeps the original model size and dynamically selects parts of parameters at a time. For dynamic inference, previous work focuses on how to dynamically drop layers to accelerate inference (Huang et al., 2018; Wu et al., 2020; Li et al., 2021). In this manner, the output of each layer is expected to be able to predict labels, and hence it will introduce additional training objectives and prediction strategies. In contrast, MoEfication simplifies models in a finer granularity, and does not change the process of training and inference. In summary, MoEfication can be regarded as a novel direction diagonal with the above-mentioned approaches.

Large-scale PLMs with MoE. Jacobs et al. (1991) propose mixture-of-experts to build a system composed of many separate networks, which learn to handle a subset of the training examples independently. When deep neural networks achieve great success (Hinton et al., 2012; Krizhevsky et al., 2012; Goodfellow et al., 2013), Bengio (2013) thinks the model size is a key factor and MoE is an important technique to scaling model computation and proposes the idea of "conditional computation". The first large-scale MoE language model is proposed by Shazeer et al. (2017), which adds an MoE layer between two LSTM layers and independently assigns tokens to combinations of experts. Recently, GShard (Lepikhin et al., 2021), Switch-Transformer (Fedus et al., 2021), BASELayer (Lewis et al., 2021), and HashLayer (Roller et al., 2021) study how to build large-scale Transformer-based models with MoE and optimal training strategies, which can fully utilize the



(a) FFN Computation Process (b) Unused elements and neurons (c) Expert Construction (d) FFN with MoE

Figure 2: An example of the sparse activation phenomenon and MoEfication. (a) shows the computation process of an FFN for a given input. (b) shows the unused elements and neurons for this input. (c) shows how to construct experts. (d) shows how the MoEfied model handles this input efficiently.

model capacity. Different from them, we utilize the naturally-existing sparse activation phenomenon to convert a model into its MoE version for better efficiency during inference.

3 Method

In this section, we will introduce the general idea of MoEfication and divide it into two phases: expert construction and expert selection.

3.1 Overall Framework

MoEfication aims to utilize the sparse activation phenomenon in the FFNs of Transformers to reduce the computation cost.

We first formally describe the sparse activation phenomenon. The FFNs of Transformers are two-layer fully connected networks, which process an input representation $\boldsymbol{x} \in \mathbb{R}^{d_{model}}$ by

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1, \\ F(\boldsymbol{x}) &= \sigma(\boldsymbol{h})\boldsymbol{W}_2 + \boldsymbol{b}_2, \end{aligned} \tag{1}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ are the weight matrices, $\boldsymbol{b}_1 \in \mathbb{R}^{d_{ff}}$ and $\boldsymbol{b}_2 \in \mathbb{R}^{d_{model}}$ are the bias vectors, and $\sigma(\cdot)$ is a non-linear activation function, which prefers to retain positive values and discard negative ones. In this work, we study the activation function ReLU (Nair and Hinton, 2010), which is used by the original Transformer (Vaswani et al., 2017) and some widely-used Transformer-based PLMs (Sun et al., 2020; Raffel et al., 2020).

As shown in Figure 1, there are many inactive (zero) values in the intermediate output $\sigma(\boldsymbol{h})$. The computation of these values can be omitted for acceleration. Meanwhile, different inputs will activate different neurons. Hence, we explore to select the possibly-activated neurons of $\boldsymbol{h}$ before the FFN computation instead of model pruning.

We show an example in Figure 2. In this FFN, d_{model} is 2, d_{ff} is 4, and the bias vectors are omitted for simplification. For a given input representation $\boldsymbol{x}$, there are two positive values in $\boldsymbol{h}$. Hence, we only need to compute part of the FFN, i.e., a 2×2 submatrix of $\boldsymbol{W}_1$ and a 2×2 submatrix of $\boldsymbol{W}_2$, to obtain the same output $F(\boldsymbol{x})$. Correspondingly, we can MoEfy the original FFN to have an MoE layer with two experts and select the one on the right-hand side for this input $\boldsymbol{x}$.

For MoEfication, we first split the FFN into several independent parts, namely expert construction, and then design a router to select suitable experts for each input, namely expert selection.

3.2 Expert Construction

In this subsection, we introduce how to split an FFN into several parts. The core idea is to group together the neurons that are often activated simultaneously. In this way, for each input, we can select a small number of experts to cover all its activated neurons. To achieve better parallel computation performance, we set the size of each expert to be the same. If the number of experts is k, the input and output dimension of experts is still d_{model} and their intermediate dimension is $d_e = \frac{d_{ff}}{k}$. Then, the parameters of i-th expert are denoted by

$$\boldsymbol{W}_1^i \in \mathbb{R}^{d_{model} \times d_e}, \boldsymbol{b}_1^i \in \mathbb{R}^{d_e}, \boldsymbol{W}_2^i \in \mathbb{R}^{d_e \times d_{model}}. \tag{2}$$

Given the result of splitting, we construct the corresponding permutation of intermediate neurons by $\begin{pmatrix} 1 & 2 & \dots & d_{ff} \\ f(1) & f(2) & \dots & f(d_{ff}) \end{pmatrix}$, where $f(n)$ is the mapping function from the original neuron index to the permuted neuron index. We compute $f(n)$ by

$$f(n) = (e(n) - 1)d_e + |\{m | m \le n, e(m) = e(n)\}|, \tag{3}$$

where $e(n)$ is the expert index of the n-th neuron, which varies from 1 to k, and $|\{m | m \le n, e(m) =$

$e(n)\}|$ is the index of the n-th neuron in the expert. Then, we use its permutation matrix $\boldsymbol{P} \in \mathbb{R}^{d_{ff} \times d_{ff}}$ to permute the rows or columns of parameters and have the following split:

$$\begin{aligned}[\boldsymbol{W}_1^1, \boldsymbol{W}_1^2, \ldots, \boldsymbol{W}_1^k] &= \boldsymbol{W}_1\boldsymbol{P}, \\ \boldsymbol{b}_1^1 \oplus \boldsymbol{b}_1^2 \oplus \ldots \oplus \boldsymbol{b}_1^k &= \boldsymbol{b}_1\boldsymbol{P}, \\ [(\boldsymbol{W}_2^1)^T, (\boldsymbol{W}_2^2)^T, \ldots, (\boldsymbol{W}_2^k)^T] &= (\boldsymbol{P}^T\boldsymbol{W}_2)^T,\end{aligned} \tag{4}$$

where $\oplus$ represents the vertical concatenation. Note that the permutation will not influence the output representation:

$$\begin{aligned}\sigma(\boldsymbol{h})\boldsymbol{W}_2 + \boldsymbol{b}_2 &= \sigma(\boldsymbol{h})\boldsymbol{P}\boldsymbol{P}^T\boldsymbol{W}_2 + \boldsymbol{b}_2, \\ &= \sigma(\boldsymbol{h}\boldsymbol{P})\boldsymbol{P}^T\boldsymbol{W}_2 + \boldsymbol{b}_2, \\ &= \sigma(\boldsymbol{x}\underline{\boldsymbol{W}_1\boldsymbol{P}} + \underline{\boldsymbol{b}_1\boldsymbol{P}})\underline{\boldsymbol{P}^T\boldsymbol{W}_2} + \boldsymbol{b}_2.\end{aligned} \tag{5}$$

In this work, we propose two methods to split an FFN into k parts.

Parameter Clustering Split. To take the parameter information into consideration, we treat the columns of $\boldsymbol{W}_1$ as a collection of vectors with d_{model} dimension. Based on the intuition that the neurons with similar vectors will be activated simultaneously, we apply balanced K-Means (Malinen and Fränti, 2014) to the vector collection to obtain k clusters to construct the mapping function.

Co-Activation Graph Split. To directly use the information of co-activation, we construct a co-activation graph by counting co-activations of PLMs for the samples of the training set. Each neuron will be represented by a node in the graph, and the edge weight between two nodes are their co-activation values. The co-activation value is computed by

$$\text{co-activation}(n, m) = \sum_{\boldsymbol{x}} \boldsymbol{h}_n^{(\boldsymbol{x})} \boldsymbol{h}_m^{(\boldsymbol{x})} \mathbb{1}_{\boldsymbol{h}_n^{(\boldsymbol{x})}>0, \boldsymbol{h}_m^{(\boldsymbol{x})}>0}, \tag{6}$$

where $\boldsymbol{h}_n^{(\boldsymbol{x})}$, $\boldsymbol{h}_m^{(\boldsymbol{x})}$ are the n-th and the m-th neurons of $\boldsymbol{h}$ for the input $\boldsymbol{x}$ and $\mathbb{1}_{\boldsymbol{h}_n^{(\boldsymbol{x})}>0, \boldsymbol{h}_m^{(\boldsymbol{x})}>0}$ indicates $\boldsymbol{h}_n^{(\boldsymbol{x})}$ and $\boldsymbol{h}_m^{(\boldsymbol{x})}$ are activated simultaneously. Then, we apply graph partitioning algorithms (Karypis and Kumar, 1998) to the co-activation graph to obtain the split, where the internal connections for each group will be strong. It means that the neurons splitted into the same group are often activated simultaneously for the training samples.

3.3 Expert Selection

In this subsection, we introduce how to create a router for expert selection. An MoEfied FFN processed an input $\boldsymbol{x}$ by

$$F_m(\boldsymbol{x}) = \sum_{i \in S} \sigma(\boldsymbol{x}\boldsymbol{W}_1^i + \boldsymbol{b}_1^i)\boldsymbol{W}_2^i + \boldsymbol{b}_2, \tag{7}$$

where S is the set of the selected experts. If all experts are selected, we have $F_m(\boldsymbol{x}) = F(\boldsymbol{x})$. Considering that $\sigma(\boldsymbol{x}\boldsymbol{W}_1^i + \boldsymbol{b}_1^i)\boldsymbol{W}_2^i$ equals to $\boldsymbol{0}$ for most experts, we try to select n experts, where $n < k$, minimize $||F_m(\boldsymbol{x}) - F(\boldsymbol{x})||_2$. The selection methods will assign a score s_i to each expert for the given input $\boldsymbol{x}$ and select the experts with the n highest scores by

$$S = \underset{A \subset \{1,2,\ldots,k\}, |A|=n}{\arg\max} \sum_{i \in A} s_i. \tag{8}$$

Groundtruth Selection for the intermediate output $\sigma(\boldsymbol{h})$. We can obtain the groundtruth selection, which minimizes $||\sum_{i \in S} \sigma(\boldsymbol{x}\boldsymbol{W}_1^i + \boldsymbol{b}_1^i) - \sigma(\boldsymbol{h})||_2$, by a greedy algorithm. We calculate the sum of positive values in each expert as s_i and select experts using Equation 8. This selection should approximate to the lower bound of $||F_m(\boldsymbol{x}) - F(\boldsymbol{x})||_2$. Correspondingly, its performance will approximate to the ideal performance of an MoEfied model. Meanwhile, it is intractable to directly optimize $||F_m(\boldsymbol{x}) - F(\boldsymbol{x})||_2$ because there are too many possible combinations of experts.

Similarity Selection. To utilize the parameter information, we average all columns of $\boldsymbol{W}_1^i$ and use it as the expert representation. Given an input $\boldsymbol{x}$, we calculate the cosine similarity between the expert representation and $\boldsymbol{x}$ as s_i.

MLP Selection. We train a multi-layer perceptron (MLP), which takes the $\boldsymbol{x}$ as input and predicts the sum of positive values in each expert. Then, we use the prediction as s_i. This method tries to approximate to the performance of groundtruth selection.

4 Experiment

4.1 Experimental Setups

Models and Hyperparameters We use four variants of T5 (Raffel et al., 2020), which are the 60-million-parameter T5-Small, the 200-million-parameter T5-Base, the 700-million-parameter T5-Large, and the 3-billion-parameter T5-XLarge. The non-linear activation function is ReLU (Nair and Hinton, 2010). We use Adam as the optimizer and a learning rate of 10^{-6} for fine-tuning on downstream tasks. The batch size is set to 64 and the number of epochs is set to 3.

Datasets. We use several natural language understanding datasets to evaluate our models. For text classification, we use GLUE benchmark (Wang et al., 2019), including MNLI-matched (Williams

et al., 2018), QNLI (Rajpurkar et al., 2016), QQP[2], RTE (Dagan et al., 2006), SST-2 (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), CoLA (Warstadt et al., 2019), and STS-B (Giampiccolo et al., 2007). For reading comprehension, we use SQuAD (Rajpurkar et al., 2016) and RACE (Lai et al., 2017), which are the representative datasets for span extraction and multi-choice QA, respectively. We report the results on their development sets. For MNLI, QNLI, QQP, RTE, SST-2, MRPC, RACE, we use accuracy as the metric. For CoLA, we use matthews correlation coefficient as the metric. For STS-B, we use pearson and spearman correlation as the metrics. For SQuAD, we use F1 score as the metric.

Expert Construction. For balanced K-Means, we use an open-source implementation[3]. Besides Parameter Clustering Split and Co-activation Graph Split, we also implement Random Split as a naive baseline, which uses an identity matrix as $\boldsymbol{P}$. We set the number of neurons in each expert to 32. Correspondingly, the number of experts varies from 64 to 512 for different T5 variants. With the same expert size, the relative computation cost of routing is the same as shown in Appendix.

Expert Selection. Besides Similarity Selection and MLP Selection, we also implement Random Selection, where we treat each expert as a collection of vectors with d_{model} dimension and randomly select one of them as the expert representation. For Random Selection and Similarity Selection, the computation complexity for routing is O(kd_{model}). For MLP Selection, we use a two-layer feed-forward network as the architecture. The input dimension is d_{model}, the intermediate dimension is k, and the output dimension is k. The non-linear activation function is $\tanh(\cdot)$. Its computation complexity is O($kd_{model} + k^2$). Compared to the computation complexity of FFNs of the original model, O($d_{model} \cdot d_{ff}$), the computation cost of routers is ignorable because k is much smaller than d_{ff}. For example, k is 128 and d_{ff} is 4096 for T5-Large. For the training of our MLP routers, we adopt cross-entropy as the training objective and use the Adam optimizer with the learning rate of 10^{-2}. The batch size is set to 512 and the number of epochs is set to 10. We sample nearly 500 thousand input representations from the training data and split them into the training and development sets with the ratio of 9 : 1. Note that we only use the activation information as supervision. The training time of each FFN is about several minutes on a single GPU.

[2] https://data.quora.com

[3] https://github.com/ndanielsen/Same-Size-K-Means

Model	SST-2	MNLI	RACE
Small	90.9	82.4	44.7
Small-Distill	91.9	82.6	50.6
Base	94.0	86.4	71.7
Large	96.2	89.5	81.3
XLarge	96.9	90.5	85.6

Table 1: Original Performance of different models on three downstream tasks. The model architecture is T5.

4.2 MoEfication with Different Models

In this subsection, we evaluate MoEfication on different PLMs. We consider two factors: the model size and whether the model is compressed. For the model size, we use five variants of T5 (Raffel et al., 2020), from T5-Small to T5-XLarge. For convenience, we directly use the scale names as the abbreviations. To investigate the influence of model compression, we compress T5-Large to T5-Small by classic knowledge distillation (Hinton et al., 2015). Specifically, the teacher model is a fine-tuned T5-Large and the student model is a pre-trained T5-Small. The distilled model is denoted by T5-Small-Distill. The expert construction and selection methods used here are Co-activation Graph Split and MLP Selection, which are proved to be the best combination in Section 4.4.

We report the performance of these models on three datasets, SST-2, MNLI, and RACE, in Table 1. They are the representative datasets for single-sentence classification, sentence-pair classification, and reading compression, respectively. The original performance of PLMs grows as the model size grows, and knowledge distillation improves the performance of T5-small.

We first calculate the activation statistics of different models by inputting the training data of each dataset. The results are shown in Figure 3. From the figure, we have three observations. (1) The activations of these models are sparse. Different from the previous study on models trained with smaller datasets, where the activation ratios are range from 10% to 50% (Geva et al., 2021)[4], we find most inputs activate less than 10% of the neurons. (2) The activations of larger models are sparser than

[4] Since the activation ratios of a randomly-initialized model are around 50%, we guess these models do not make full use of their parameters.

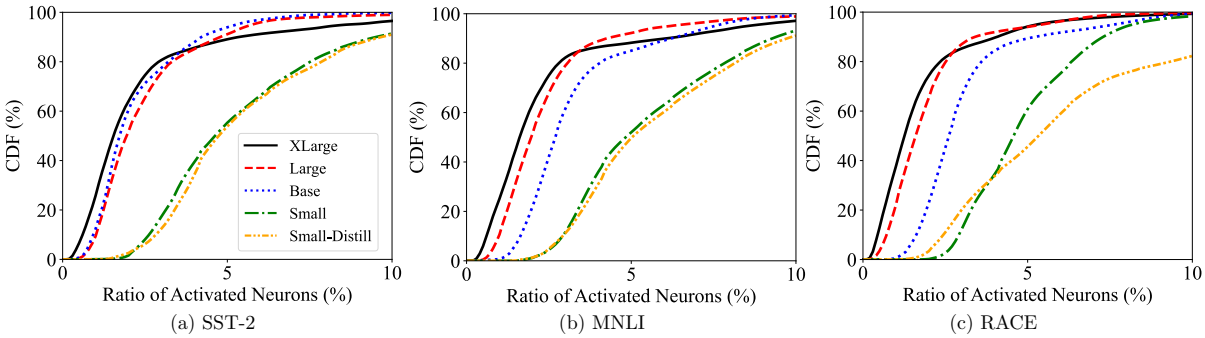


Figure 3: CDF of the ratio of activated neurons for each input with different models on three datasets.

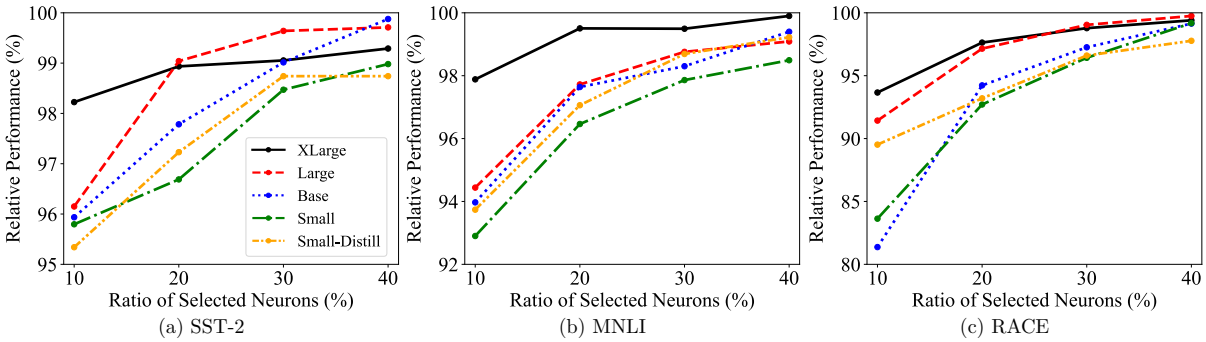


Figure 4: Relative performance of MoEfied models with different sizes on three datasets. Dynamically selecting 10% to 20% neurons can recover nearly 98% original performance for large models such as T5-XLarge.

those of smaller models. For example, 80% inputs only activate less than 3% neurons in T5-XLarge while 40% inputs activate more than 3% neurons in T5-Small. (3) The sparsity is less related to distillation than the model size. The CDF curve of T5-Small-Distill is close to that of T5-Small.

Then, we compare the performance of MoEfied models with different sizes and ratios of selected neurons and report the results in Figure 4. To measure the performance of MoEfication, we calculate the relative performance of the MoEfied model to the original model. From the figure, we have four observations. (1) MoEfication works well with all models on all three datasets. MoEfied models save 80% computation cost of FFNs while maintaining over 95% original performance. (2) The larger models can use fewer neurons to recover the original performance. For example, T5-XLarge achieves nearly 98% relative performance on SST-2 and MNLI with 10% neurons while T5-Small achieves the same results with 30% to 40% neurons. This result is consistent with the activation statistics, that is, larger models are sparser. We can expect that MoEfication can work better with super large models. (3) Difficult tasks require models to select more experts to maintain the performance. From Table 1, we can see that the accuracy of RACE is much lower than the other two tasks, and hence we think RACE is more difficult. Correspondingly, the relative performance with 10% neurons on RACE is also lower than those on the other tasks. (4) MoEfication works similarly on T5-Small and T5-Small-Distill, which indicates that MoEfication can work with knowledge distillation for more efficient inference.

4.3 Parameter Calibration

In practice, there is still a gap between the performance of MoEfied models and that of original models because selected experts cannot cover all positive neurons with a limited computation budget. Hence, the outputs of MoEfied models will be slightly different from those of original models. To calibrate MoEfied models, we further fine-tune the models on the training set, namely parameter calibration. Considering that current routers are based on the first layers of FFNs ($\boldsymbol{W}_1$ and $\boldsymbol{b}_1$), we only optimize the second layers of FFNs ($\boldsymbol{W}_2$ and $\boldsymbol{b}_2$) to ensure routers can also work well after fine-tuning.

We evaluate this method on several downstream natural language understanding tasks with T5-Large. The ratio of selected neurons is set to 20%,

	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	RACE	SQuAD 1.1	Avg.
Original	89.5	94.4	91.7	87.1	96.2	88.0	59.4	91.2/90.9	81.3	93.2	87.2
MoEfied	87.5	93.2	90.2	86.4	95.4	87.5	55.5	90.6/90.3	79.0	92.2	85.7 (-1.5)
+GT	89.1	94.1	91.4	86.4	96.3	88.3	58.8	90.9/90.8	80.8	93.2	86.9 (-0.3)
+Calib	88.7	93.6	91.3	87.5	96.2	89.3	59.4	91.0/90.6	79.9	92.3	86.9 (-0.3)

Table 2: Results of T5-Large on GLUE benchmark and two QA datasets. The last row reports the differences between the original model and MoE+Calib. MoEfied models with parameter calibration achieve comparable performance to original models.

Construction	Selection	SST-2	MNLI	RACE
-	-	96.2	89.5	81.3
Random	Groundtruth	95.9	87.3	80.0
	Random	65.9	36.3	29.2
	Similarity	90.3	75.9	56.7
	MLP	94.1	84.1	75.0
Parameter Clustering	Groundtruth	95.5	88.8	80.9
	Random	70.6	36.4	41.8
	Similarity	86.7	66.3	63.6
	MLP	**95.9**	**87.5**	78.7
Co-Activation Graph	Groundtruth	96.3	89.1	80.8
	Random	85.3	68.5	54.7
	Similarity	92.2	81.4	71.0
	MLP	95.4	**87.5**	**79.0**

Table 3: Comparisons of different combinations of expert construction and selection methods using T5-Large. The first row is the original performance. The best results in each group are underlined and the best results on each dataset are in **boldface**.

Model	MLM Loss
MoE Pre-training	3.09
Standard Pre-training	2.88 (-0.21)
+MoEfication	3.02 (-0.07)
+GT	2.95 (-0.14)

Table 4: Comparisons of MoE models pre-trained from scratch and modified by MoEfication. We report the MLM loss on the validation set. Standard pre-training with MoEfication is better than pre-training a MoE model from scratch.

which is sufficient for T5-Large as show in Figure 3. We use a small learning rate of 10^{-7} for calibration. The other hyper-parameters remain the same as fine-tuning. The results are shown in Table 2. MoEfied refers to the combination of Co-activation Graph Split and MLP Selection. MoEfied+GT refers to the combination of Co-activation Graph Split and Groundtruth Selection. MoEfied+Calib is the calibrated version of MoEfied.

We observe that MoEfication introduces small performance loss (about 1.5% on average) with an 80% reduction of the computation cost in FFNs. Meanwhile, calibration can effectively deal with the issue of the precision errors brought by MoEfication. For example, MoEfied+Calib improves MoEfied by nearly 4% on CoLA and achieves the same average performance as MoEfied+GT.

4.4 Comparisons of MoEfication Strategies

To find the most effective MoEfication strategy, we evaluate different combinations of expert construction and selection methods. We use T5-Large and also set the ratio of selected neurons to 20%. The results are shown in Table 3. From the table, we have two observations:

(1) For expert construction, Co-activation Graph Split is the best method according to the overall performance. Compared to the other two methods, Co-activation Graph Split directly uses the co-activation information to group the neurons activating simultaneously into the same expert.

(2) For expert selection, the performance of Groundtruth Selection is close to that of the original model, which indicates that 20% parameters of FFNs are sufficient to achieve good performance on T5-Large. Meanwhile, MLP Selection is the best expert selection method and can work well with both Parameter Clustering Split and Co-activation Graph Split.

4.5 MoEfication vs. MoE pre-training

In this subsection, we compare the performance of two kinds of MoE models. The first one is pre-trained from scratch. The second one is transformed from a standard model by MoEfication. For fair comparisons, we pre-train one MoE model and one standard model with the same model size from scratch using WikiText-103 (Merity et al., 2017). The pre-training objective is masked language modeling (MLM). The model architecture is the same as T5-Small. For pre-training, we use the batch size of 4096, the learning rate of 0.01, the maximum sequence length of 512, and the Adam optimizer. The number of experts is set to 64 and the router

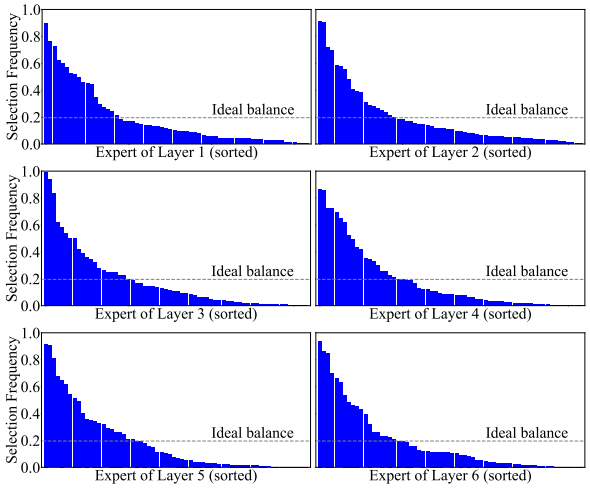


Figure 5: Selection Frequency of 64 experts in each encoder layer of MoEfied T5-Small. The frequency of ideal balance selection is 0.2 while the distribution is much unbalanced.

will select 32 of them for a single input.

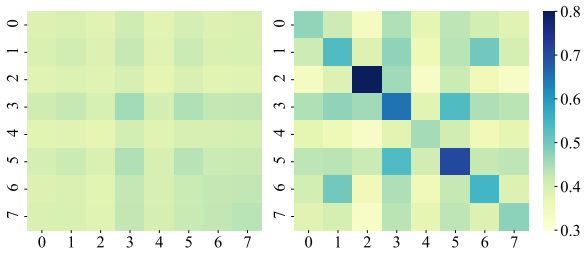
We report the MLM loss on the validation set in Table 4. From the table, we have two observations. (1) The loss of the standard pre-trained model is lower than that of the pre-trained MoE model. We guess that the optimization of MoE models is difficult than that of the standard models because of the restricted selection of MoE models. (2) MoEfied models achieve better performance than the pre-trained MoE model. It indicates that pre-training a standard model then conducting MoEfication can be a better option than pre-training an MoE model from scratch.

5 Analysis

In this section, we investigate the routing patterns of MoEfied models and validate whether they are consistent with those of MoE models trained from scratch.

First, we count the selection frequency of each expert. Previous work introduces training objectives to ensure balance selection to make full use of model parameters (Lepikhin et al., 2021; Fedus et al., 2021). We report the results of the MoEfied T5-Small with 20% experts on SST-2 in Figure 5. From the figure, we observe that the frequency distribution of expert selection is much unbalanced[5]. There are some commonly-used experts, whose frequencies are higher than 80%. Meanwhile, there

[5]Unbalanced selection will not influence the computation efficiency with current MoE implementations such as FastMoE (He et al., 2021a).



(a) The 8 most selected experts (b) The 8 least selected experts

Figure 6: Input similarities between experts in the last encoder layer of MoEfied T5-Small. For the most selected experts, both the self-similarities and inter-similarities are low. For the least selected experts, the self-similarities are much higher than inter-similarities.

are also some long-tail experts whose frequencies are lower than 10%.

Then, we calculate the self-similarities and inter-similarities of inputs between experts by sampling 10,000 inputs for each expert. We report the results of the last layer in Figure 6. For the most selected experts, which are selected by most inputs, the self-similarities are close to the inter-similarities. For the least selected experts, the self-similarities are much higher than the inter-similarities, which suggests that the inputs of each expert have obvious cluster structure.

From these results, we can conclude the routing patterns of MoEfied models: there are some general experts, which can work for most inputs, and some input-specific experts, which are seldom used and may work in specific domains or tasks. This observation may inspire future work on training MoE models from scratch.

6 Conclusion

In this work, we propose MoEfication, a new model acceleration technique, for large-scale Transformer models. MoEfication utilizes the sparse activation phenomenon in FFNs of Transformer to convert a normal model to its MoE version with the same parameters. Experimental results show that large MoEfied models can achieve comparable performance to the original models using only 10% to 20% computation cost of FFNs. By studying the routing patterns of MoEfied models, we find that there are general and input-specific experts, which may inspire future work on training MoE models. In the future, we plan to extend MoEfication to other Transformer models, such as BERT and GPT, and design better strategies for MoEfication. We hope MoEfication can benefit the real-world applications of large PLMs with better efficiency.

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The Long-Document transformer. *arXiv preprint 2004.05150*.

Yoshua Bengio. 2013. Deep learning of representations: Looking forward. In *Proceedings of SLSP*, pages 1–37.

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. 2021. Language models are Few-Shot learners. In *Proceedings of NeurIPS*, pages 1877–1901.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In *Machine learning challenges.*, pages 177–190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL*, pages 4171–4186.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of IWP*, pages 9–16.

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

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer Feed-Forward layers are Key-Value memories. *arXiv preprint 2012.1491*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of TEP*, pages 1–9.

Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. Maxout networks. In *Proceedings of ICML*, pages 1319–1327.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-Trained models: Past, present and future. *arXiv preprint 2106.07139*.

Jiaao He, Jiezhong Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. 2021a. FastMoE: A fast Mixture-of-Expert training system. *arXiv preprint 2103.13262*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021b. DeBERTa: decoding-enhanced bert with disentangled attention. In *Proceedings of ICLR*.

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

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint 1207.0580*.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. 2018. Multi-Scale dense networks for resource efficient image classification. In *Proceedings of ICLR*.

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.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of EMNLP*, pages 4163–4174.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proceedings of ICLR*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of NeurIPS*, pages 1106–1114.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of EMNLP*, pages 785–794.

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 *Proceedings of ICLR*.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. BASE layers: Simplifying training of large, sparse models. *arXiv preprint 2103.16716*.

Bingbing Li, Zhenglun Kong, Tianyun Zhang, Ji Li, Zhengang Li, Hang Liu, and Caiwen Ding. 2020. Efficient transformer-based large scale language representations using hardware-friendly block structured pruning. In *Findings of EMNLP*, pages 3187–3199.

Lei Li, Yankai Lin, Deli Chen, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021. CascadeBERT: Accelerating inference of pre-trained language models via calibrated complete models cascade. In *Findings of EMNLP*, pages 475–486.

Mikko I. Malinen and Pasi Fränti. 2014. Balanced k-means for clustering. In *Proceedings of SSSPR*, volume 8621, pages 32–41.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proceedings of ICLR*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proceedings of NeurIPS*, pages 14014–14024.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of ICML*, pages 807–814.

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. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pages 2383–2392.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason Weston. 2021. Hash layers for large sparse models. *arXiv preprint arXiv:2106.04426*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint 1910.01108*.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The Sparsely-Gated Mixture-of-Experts layer. In *Proceedings of ICLR*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pages 1631–1642.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of EMNLP*, pages 4323–4332.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact Task-Agnostic BERT for Resource-Limited devices. In *Proceedings of ACL*, pages 2158–2170.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, and Jakob Uszkoreit. 2017. Attention is all you need. In *Proceedings of NeurIPS*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing Multi-Head Self-Attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of ACL*, pages 5797–5808.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *TACL*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, pages 1112–1122.

Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, Yi Yang, and Shilei Wen. 2020. Dynamic inference: A new approach toward efficient video action recognition. *arXiv preprint 2002.03342*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of ACL*, pages 2246–2251.

Dongkuan Xu, Ian En-Hsu Yen, Jinxi Zhao, and Zhibin Xiao. 2021. Rethinking network pruning – under the pre-train and fine-tune paradigm. In *Proceedings of NAACL-HLT*, pages 2376–2382.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. *arXiv preprint 1910.06188*.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of EMNLP*, pages 509–521.

Zhengyan Zhang, Fanchao Qi, Zhiyuan Liu, Qun Liu, and Maosong Sun. 2021. Know what you don't need: Single-Shot Meta-Pruning for attention heads. *AI Open*, 2:36–42.

A Activation Statistics before Fine-tuning

We count the activation statistics of PLMs before fine-tuning on the pre-training data containing about $50,000$ input tokens. The results are shown in Figure 7. We observe that PLMs before fine-tuning also have the sparse activation phenomenon and fine-tuning brings little change.

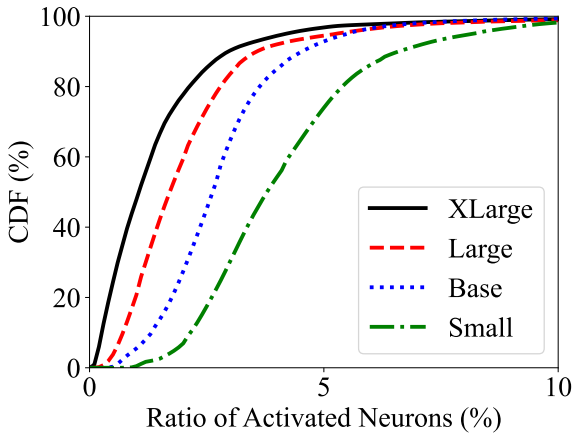


Figure 7: CDF of the ratios of activated neurons for each input with different models before fine-tuning.

Then, we compare the activations of pre-trained models and those of fine-tuned models. We use the average ratio of activated neurons as the index. The results are shown in Table 5. We observe that fine-tuning increases the average activation ratio for most models. The reason may be that different neurons start to learn the same task-specific patterns during fine-tuning. Interestingly, the increase on RACE is smaller than that on the other datasets. Since RACE is more difficult than the other datasets, there should be more task-specific patterns in RACE and less neurons learn the same patterns. Moreover, the pre-training task MLM requires more patterns than RACE so the ratios of MLM are lowest.

	Small	Base	Large	XLarge
MLM	4.18	2.85	2.17	1.52
SST-2	5.53	2.24	2.50	2.46
MNLI	5.59	3.25	2.44	2.45
RACE	4.94	3.08	1.98	1.79

Table 5: Average ratio of activated neurons for each input. MLM represents the pre-trained models with masked language modeling. SST-2, MNLI, RACE represent the fine-tuned models on each dataset.

B Results of Graph Partition

Co-activation Graph Split achieves good performance in expert construction. Here, we study whether the co-activation graph is suitable for partitioning. We report the results of graph partition of T5-Large on SST-2 in Figure 8. Smaller ratios of edgecuts, which straddle partitions, mean that more co-activation pairs are included in experts. We only report the results of encoder layers because all ratios of decoder layers are smaller than 0.001. From this figure, we can see that the overall ratio is small and these graphs are suitable for partitioning.

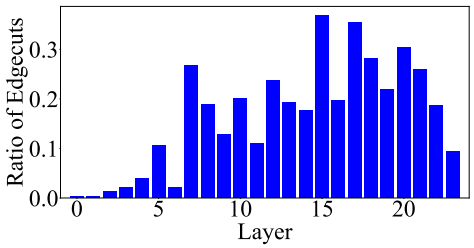


Figure 8: Ratio of edgecuts in different layers.

C Accuracy of MLP Selection

MLP selection trains MLPs to fit the groundtruth selection. In this part, we report the accuracy of MLPs in T5-Large fine-tuned on SST-2. The results are shown in Figure 9 and 10. The overall accuracy of the encoder is about 0.8 and the overall accuracy of the decoder is about 0.7.

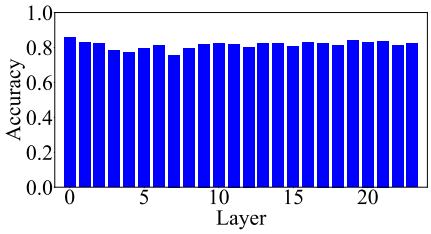


Figure 9: Accuracy of MLPs of encoder layers.

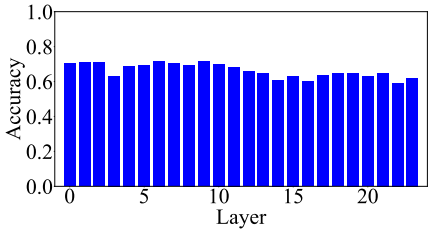


Figure 10: Accuracy of MLPs of decoder layers.

D Relative Cost of Routing

In this work, we set the number of neurons in each expert to 32. Then, the number of experts in each layer k is $\frac{d_{ff}}{32}$. In most Transformer models, $d_{ff} = 4d_{model}$. The computation complexity of Similarity Selection for each input is

$$O(kd_{model}) = O(\frac{d_{model}^2}{8}). \tag{9}$$

The computation complexity of FFNs for each input is

$$O(d_{model} \cdot d_{ff}) = O(4d_{model}^2). \tag{10}$$

Then, the relative cost of routing to that of FFNs is constant for different models. It is also similar to MLP Selection.