Exploring the Benefit of Activation Sparsity in Pre-training

Zhengyan Zhang [1] **Chaojun Xiao** [1] **Qiujieli Qin** [1] **Yankai Lin** [2] **Zhiyuan Zeng** [1] **Xu Han** [1] **Zhiyuan Liu** [1]
Ruobing Xie [3] **Maosong Sun** [1 4] **Jie Zhou** [3]

Abstract

Pre-trained Transformers inherently possess the characteristic of sparse activation, where only a small fraction of the neurons are activated for each token. While sparse activation has been explored through post-training methods, its potential in pre-training remains untapped. In this work, we first study how activation properties change during pre-training. Our examination reveals that Transformers exhibit sparse activation throughout the majority of the pre-training process while the activation correlation keeps evolving as training progresses. Leveraging this observation, we propose Switchable Sparse-Dense Learning (SSD). SSD adaptively switches between the Mixtures-of-Experts (MoE) based sparse training and the conventional dense training during the pre-training process, leveraging the efficiency of sparse training and avoiding the static activation correlation of sparse training. Compared to dense training, SSD achieves comparable performance with identical model size and reduces pre-training costs. Moreover, the models trained with SSD can be directly used as MoE models for sparse inference and achieve the same performance as dense models with up to 2× faster inference speed. Codes are available at https://github.com/thunlp/moefication.

1. Introduction

Recent studies have uncovered a notable characteristic of pre-trained Transformers: the sparse activation of neurons in their intermediate layers (Zhang et al., 2022b; Li et al., 2023; Liu et al., 2023; Dong et al., 2023; Mirzadeh et al., 2023). During inference, it has been observed that only a small fraction of the intermediate hidden states are activated, rendering a non-zero state, while the majority remain inactive. Sparse activation presents a promising direction for improving the efficiency of Transformer-based models.

Previous work has primarily focused on leveraging the sparse activation phenomenon to speed up the inference process through post-training methods (Liu et al., 2023; Alizadeh et al., 2023). For instance, with model parameters frozen, DejaVu (Liu et al., 2023) proposes to selectively engage a subset of neurons likely to activate during inference, thereby reducing both the parameter communication and model computation costs. However, the potential of utilizing sparse activation in pre-training remains largely unexplored.

Unlike the widely explored domain of post-training, where model parameters are fixed, the pre-training of Transformers is dynamic, requiring ongoing updates to model parameters. Therefore, a preliminary step is to investigate the activation of Transformers during pre-training. We conduct experiments on three representative text models, including GPT (Radford et al., 2019), BERT (Devlin et al., 2019), and T5 (Raffel et al., 2020), with different architectures and pre-training objectives.

Our findings reveal that these models become sparsely activated in the early stage of pre-training, subsequently stabilizing in this sparse state. It suggests that sparse activation is a pervasive phenomenon across pre-trained models, existing throughout the majority of the pre-training process. Meanwhile, although the activation sparsity becomes stable after a certain stage of pre-training, the activation pattern is still dynamic: the set of activated neurons for a certain input varies across different stages of pre-training. Consequently, the sparse training method for pre-training should be adaptive to the change in the activation patterns.

Based on these observations, we propose Switchable Sparse-Dense Learning (SSD), utilizing the phenomenon of sparse activation to accelerate the pre-training of Transformers and enhance the efficiency of inference. SSD contains two kinds of training phases: the original dense training, which facilitates the evolution of activation patterns, and the subsequent

[1]NLP Group, DCST, IAI, BNRIST, Tsinghua University [2]Gaoling School of Artificial Intelligence, Renmin University of China [3]Tencent [4]Jiangsu Collaborative Innovation Center for Language Ability, Xuzhou, China. Correspondence to: Zhengyan Zhang <zy-z19@mails.tsinghua.edu.cn>, Maosong Sun <sms@tsinghua.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

sparse training, which aims to efficiently optimize model parameters after the activation patterns have stabilized. SSD switches between these two phases throughout the pre-training process. Specifically, when the activation sparsity grows high and the activation patterns become stable, we switch to sparse training by converting the dense model to a Sparsely-activated Mixture-of-Experts (SMoE) model, thereby enabling an efficient approximation of the original dense model. Unlike traditional Transformers, SMoE models replace the feed-forward networks with SMoE layers, where each expert is a feed-forward network and the SMoE layer selectively engages a subset of experts, promoting computational efficiency. To ensure the model sustains its capability for dense computation and fully leverages the model capacity, we alternate between sparse and dense training multiple times, as opposed to a permanent shift to sparse training. This strategy aims to mitigate the risk of overfitting to the sparse computation paradigm shown in previous work (Zuo et al., 2022a). Besides, the final dense model is familiar with the sparse computation form, which is beneficial for the subsequent sparse inference.

In the experiments, we evaluate GPT, BERT, and T5 on language modeling and several representative downstream tasks, including sentence classification (Socher et al., 2013), natural language inference (Bowman et al., 2015; Williams et al., 2018), reading comprehension (Rajpurkar et al., 2016), and instruction-tuning (Honovich et al., 2022; Wang et al., 2022b). Compared to traditional dense training, SSD achieves comparable performance with the same model size and fewer pre-training costs, up to $1.44\times$ speedup in FLOPs. Besides, the models pre-trained with SSD can be used as an SMoE model for inference without any additional training, and reduces the inference time of feed-forward networks by up to $2\times$ while maintaining the performance as good as densely pre-trained models. Moreover, by flexibly adjusting the number of selected experts during inference, our method achieves the best trade-off between performance and efficiency compared to other baseline methods, which is impossible for the models pre-trained with SMoE.

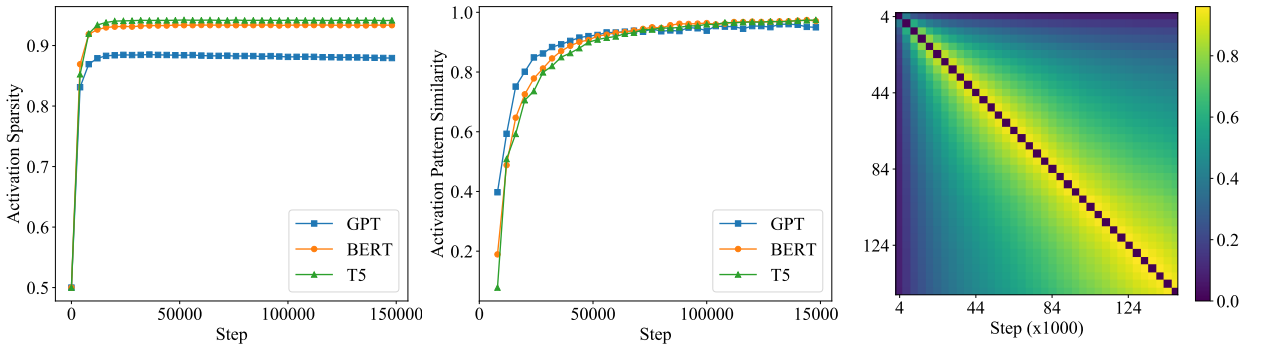
2. Related Work

Activation Sparsity of Transformers. While non-linear activation functions are prevalent in neural networks, the activation of neurons is typically dense, for instance, 44% of zeros in convolutional neural networks (Albericio et al., 2016). Contrastingly, pre-trained Transformers exhibit sparse activation, with over 90% of zeros in T5 (Zhang et al., 2022b), and similar phenomena are observed in other pre-trained models spanning both language and vision domains (Liu et al., 2023; Li et al., 2023). This sparsity has stimulated researchers' interest in utilizing it to accelerate inference. There are two main approaches: neuron-based acceleration (Liu et al., 2023) which dynamically selects subsets of neurons likely to be activated for computation, and SMoE-based acceleration (Zhang et al., 2022b), which first groups the neurons into experts and then computes in an SMoE manner by selecting the experts likely to contain most activated neurons. The former is a fine-grained approach suitable for single-instance inference, while the latter is a coarse-grained strategy well-suited for batch inference. In this work, we align with the SMoE-based approach, given that pre-training is commonly conducted in batch mode.

Sparse-Activated Mixture-of-Experts. SMoE is a representative method to improve training efficiency of Transformer-based large language models (Hazimeh et al., 2021; Gao et al., 2022; Zuo et al., 2022b; Lee-Thorp & Ainslie, 2022; Gururangan et al., 2022; Jang et al., 2023; Liu et al., 2022; Chen et al., 2023b; Muqeeth et al., 2023) and targets both feed-forward networks (FFNs) (Lewis et al., 2021; Roller et al., 2021) and attention networks (Zhang et al., 2022a). Based on the SMoE technique, we can train Transformers that are dozens of times larger without significantly increasing computational overhead (Artetxe et al., 2022; Riquelme et al., 2021). However, when we evaluate the models on an equivalent parameter basis, the performance of models pre-trained with the SMoE technique frequently lags behind that of their dense counterparts (Chen et al., 2023a).[1] The performance discrepancy of certain models could potentially be attributed to a phenomenon known as representation collapse (Chi et al., 2022), where multiple experts redundantly encode similar information, leading to inefficient parameter utilization. Furthermore, the mandatory selection of all experts during inference typically does not confer any notable benefits (Zuo et al., 2022a). To alleviate this issue, we combine SMoE with dense training, aiming to attain the model performance matching purely dense training while concurrently curtailing training costs. Similar to our work, Pan et al. (2024) propose to add constraints during dense training to induce SMoE-like behavior thereby improving the inference efficiency of the model but it may introduce additional training overhead.

Pre-training Acceleration Methods. In addition to SMoE, there are other methods to accelerate pre-training, including modifying the training objectives (Clark et al., 2020), inheriting the parameters from previous models (Chen et al., 2022; Qin et al., 2022; Gong et al., 2019), searching for appropriate hyperparameters (Izsak et al., 2021), and changing the model architecture (Yang et al., 2022; Zhang & He, 2020). These methods are orthogonal to our work and can be combined with our method to further improve efficiency.

[1] Although Mixtral 8×7B (Jiang et al., 2024) achieves better performance than LLaMA-70B (Touvron et al., 2023b), which is a larger dense model, the main reason may be that the pre-training corpus of Mixtral is better than that of LLaMA-70B. The direct comparison between Mixtral 8×7B and LLaMA-70B is not fair.



(a) Activation Sparsity w.r.t Step (b) Consecutive Act. Pattern Similarity w.r.t Step (c) Pair-wise Act. Pattern Similarity (GPT)

Figure 1. Activation sparsity and activation pattern change of three different models during pre-training.

3. Preliminary Study

We conduct a preliminary study on the evolution of the activation properties throughout the pre-training and focus on two aspects: activation sparsity and activation pattern. Specifically, we train three different types of PLMs, i.e., GPT, T5, and BERT, on the Pile dataset (Gao et al., 2021a), and report the statistics of the first $150,000$ steps. More pre-training details are in Section 5.1.

(1) **Activation Sparsity Change of Transformers.** Activation sparsity is defined as the fraction of zeros in the intermediate hidden states of FFNs, which is the basis of sparse computation. We save the model checkpoints every $4,000$ steps and calculate the activation sparsity of each checkpoint on the validation corpus. We plot the activation sparsity of the models during training in Figure 1(a). From this figure, we can see that the activation sparsity is around 0.5 at the beginning due to the symmetry of the initialization and quickly increases to about 0.9 after $20,000$ steps. After that, the sparsity is stable and fluctuates around 0.9. This observation is consistent with the previous work (Mirzadeh et al., 2023) on auto-regressive models. Here we extend the observation to other types of architectures and pre-training tasks, including BERT and T5.

(2) **Activation Pattern Change of Transformers.** Activation pattern refers to the activation correlation among neurons. While activation sparsity stabilizes after a certain stage in pre-training, the activation pattern remains dynamic due to the ongoing updates in model parameters throughout the training process. For instance, a pair of neurons activated together for a certain input at the onset of training may not exhibit the same behavior toward the end of training. This dynamic nature of activation patterns poses a challenge to existing sparse acceleration approaches (Liu et al., 2023), which are primarily designed for inference and may fall short in accommodating models with significantly fluctuating activation patterns.

Here, we study the activation pattern change of Transformers by analyzing the co-activation neuron groups. Utilizing MoEfication (Zhang et al., 2022b), we categorize the neurons activated together into the same group. By comparing the neuron groups of two checkpoints, we could measure the similarity of the activation patterns. This grouping process essentially serves as a neuron clustering exercise, and we use the Adjusted Rand Index (ARI) (Rand, 1971) to measure the similarity between the two clustering results. The ARI ranges from -0.5 to 1, where 1 means the two clustering results are identical and 0 means the two clustering results are random. We report the activation pattern similarity of consecutive checkpoints in Figure 1(b) and the activation pattern similarity of arbitrary checkpoints in Figure 1(c).

From these figures, we observe that the activation pattern similarity of consecutive checkpoints begins at a low point early in training, escalating to approximately 0.9 after $50,000$ steps. However, even as the activation pattern evolution decelerates during mid to late training stages, checkpoints separated by large step intervals continue to exhibit low activation pattern similarity. Due to the limited computational resources, we only study the models with about 100 million parameters. It would be interesting to investigate how the activation pattern scales with the model size in future work.

In summary, the observation of the high activation sparsity and the slow activation pattern change in the middle and late stages of training provides us with the possibility to incorporate sparse computation into dense training. After both the activation sparsity and pattern stabilize, we can apply existing sparse acceleration methods to pre-training. Note that the stabilization of the activation pattern unfolds at a slower pace compared to that of activation sparsity ($50,000$ steps vs. $20,000$ steps). Consequently, we choose to employ the metric of activation pattern similarity to pinpoint the transition juncture from dense to sparse training.

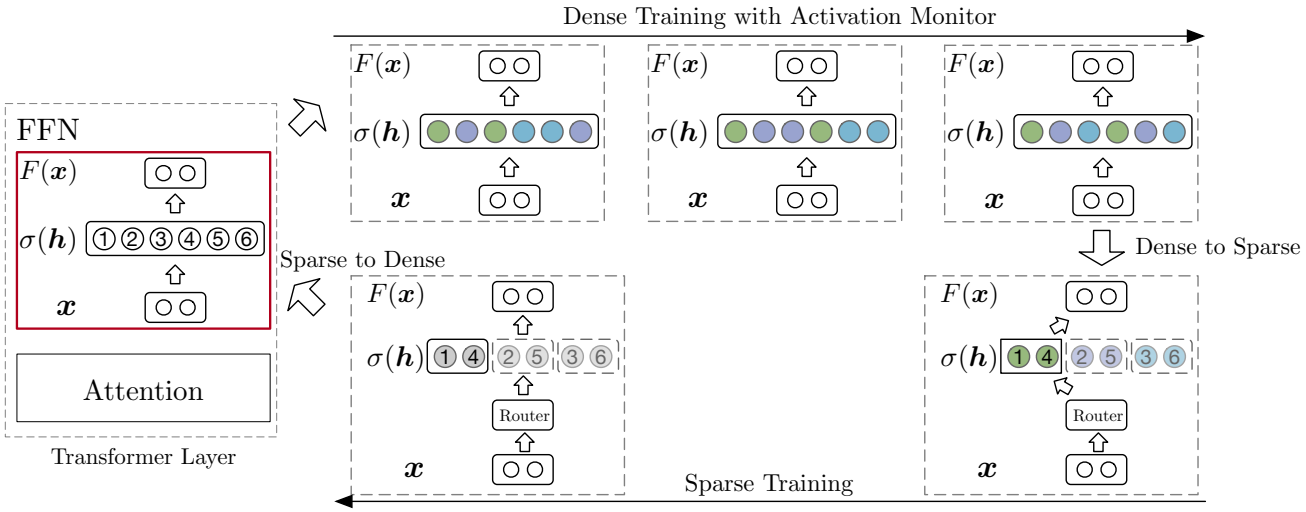


Figure 2. Illustration of SSD. During dense training, we monitor the activation pattern change for each checkpoint and transform the model into an SMoE model when the activation pattern becomes stable. During sparse training, we only compute and update the parameters of selected experts for better efficiency.

4. Method

In this section, we first describe the overall framework of SSD and then introduce its two main components: the mechanism to transition between sparse and dense, and the criteria to determine the opportune moment for such transitions.

4.1. Overall Framework

In this work, we focus on accelerating the feed-forward networks within Transformers (Vaswani et al., 2017b), which typically take more than 60% of the total computation (Wang et al., 2022a). The acceleration is achieved by switching between sparse and dense modes during the pre-training phase, as shown in Figure 2. Under sparse computation, the model is transformed into an SMoE model, incurring less computational costs compared to its original form. The sparse activation phenomenon enables the SMoE model to emulate the original model, thus achieving a balance between efficiency and effectiveness. Conversely, during dense computation, all model parameters are computed and optimized to achieve better performance. The final model reverts to a dense configuration to fully utilize the model capacity. Moreover, the final model also is familiar with the sparse computation, which can be directly used for efficient sparse inference without any additional training.

In dense computation, the FFNs are computed by

$$\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_o\sigma(\boldsymbol{W}_i\boldsymbol{x} + \boldsymbol{b}_i) + \boldsymbol{b}_o, \tag{1}$$

where $\boldsymbol{W}_i \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, $\boldsymbol{W}_o \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $\boldsymbol{b}_i \in \mathbb{R}^{d_{\text{ff}}}$, $\boldsymbol{b}_o \in \mathbb{R}^{d_{\text{model}}}$, σ is the activation function, and d_{ff} and d_{model} are the dimensions of the intermediate layer and the input/output, respectively. For simplicity, we omit the bias term $\boldsymbol{b}_i$ and $\boldsymbol{b}_o$ in the following discussion.

In sparse computation, the FFNs are equally split into N experts and computed in an SMoE manner,

$$\mathrm{FFN}_{\text{SMoE}}(\boldsymbol{x}) = \sum_{n=1}^{N} \alpha_n \boldsymbol{W}_{o,n}\sigma(\boldsymbol{W}_{i,n}\boldsymbol{x}), \tag{2}$$

where $\boldsymbol{W}_{i,n} \in \mathbb{R}^{\frac{d_{\text{ff}}}{N} \times d_{\text{model}}}$ and $\boldsymbol{W}_{o,n} \in \mathbb{R}^{d_{\text{model}} \times \frac{d_{\text{ff}}}{N}}$ are the parameters of the n-th expert, and α_n is the importance score of the n-th expert. A gating network is used to score the importance of each expert for a given input $\boldsymbol{x}$ and the experts with top-K scores are selected to compute the output. The α_n of unselected experts are set to 0. To ensure the SMoE computation is equivalent to the dense computation when $K = N$, we set the α_n of selected experts to 1 through post-processing. The details of the post-processing are provided in Appendix A

4.2. Transition between Sparse and Dense

Dense-to-Sparse Conversion. When the activation sparsity is high and the activation pattern is stable, we could efficiently approximate the original forward computation with sparse computation (Zhang et al., 2022b; Liu et al., 2023). Specifically, our approach leverages SMoE-based acceleration (Zhang et al., 2022b) over neuron-based acceleration (Liu et al., 2023) because fine-grained neuron-based selection for each token is not feasible in processing numerous tokens in a batch during pre-training.

The conversion from dense to SMoE needs to meet two

requirements: (1) the conversion should be fast to avoid additional training costs; (2) the conversion should be smooth, ensuring the performance of the converted model remains closely aligned with the original model to avoid unstable training. With these requirements in mind, we propose a method for fast and smooth conversion.

Specifically, the conversion contains two steps. (1) Neuron Clustering. We group the neurons that are often activated together into the same expert so that the SMoE model can efficiently compute most of the activated neurons by engaging a small fraction of experts to emulate the original model. Inspired by Zhang et al. (2022b), we cluster the rows of $\boldsymbol{W}_i$, each of which represents a certain neuron, into N groups by balanced k-means clustering (Malinen & Fränti, 2014), assuming that the neurons having similar weights are more likely to be activated simultaneously. This operation bypasses the need for directly counting the co-activation of neurons on a real-world corpus. The counting operation is time-consuming because it requires a large number of additional forward computations and cannot be replaced by using the activation results during training due to the dynamic nature of the activation pattern. Based on the clustering result $\boldsymbol{s} \in \mathbb{R}^{d_{\text{ff}}}$, containing the corresponding expert index for each neuron, we split the weight matrices $\boldsymbol{W}_i, \boldsymbol{W}_o$ into N sub-matrices $\boldsymbol{W}_{i,n}, \boldsymbol{W}_{o,n}$, respectively. To make the conversion smoother, we propose to use the clustering results of the previous checkpoint as the initialization of clustering in the current checkpoint. Through pilot experiments, we find that this simple strategy often provides better results, i.e., the smaller within-cluster sum of squares (WCSS)[2], than random initialization. The computation of WCSS is provided in Appendix A. To avert local optima, especially in early training stages where clustering may swiftly evolve, we conduct clustering twice, one with random initialization and the other with the initialization from the previous checkpoint, and select the better one. Formally, the clustering results $\boldsymbol{s}_j$ of the j-th checkpoint are computed by

$$\boldsymbol{s}_j = \min_{\boldsymbol{s} \in \{f(\boldsymbol{W}_i), f(\boldsymbol{W}_i, \boldsymbol{s}_{j-1})\}} \text{WCSS}(\boldsymbol{W}_i, \boldsymbol{s}), \tag{3}$$

where $f(\boldsymbol{W}_i)$ and $f(\boldsymbol{W}_i, s_{j-1})$ are the clustering results with random initialization and the initialization from the previous checkpoint, respectively. (2) Expert Selection. We use the similarity between the input $\boldsymbol{x}$ and the cluster centers as the importance score to select the top-K experts. Formally, the importance score of the n-th expert is computed by

$$\alpha_n = \boldsymbol{x}^\top \boldsymbol{c}_n, \quad \boldsymbol{c}_n = \frac{N}{d_{\text{ff}}} \sum_{m=1}^{\frac{d_{\text{ff}}}{N}} \boldsymbol{W}_{i,n}^m, \tag{4}$$

where $\boldsymbol{W}_{i,n}^m$ is the m-th row of $\boldsymbol{W}_{i,n}$, and $\boldsymbol{c}_n$ is the cluster center of the n-th expert.

[2] A metric to measure the compactness of the clustering. The smaller WCSS means the better clustering results.

Sparse-to-Dense Conversion. The performance of SMoE models tends to lag behind their dense counterparts with equivalent parameters, primarily due to the representation collapse issue (Chi et al., 2022; Zuo et al., 2022a). To optimally leverage the model capacity and avoid the overfitting of the sparse computation form, we strategically revert to dense training multiple times during training. The transition to dense is smooth given that SMoE computation aligns with dense computation when $K = N$. We conduct this conversion by concatenating the weight matrices of all experts, thereby obtaining the dense weight matrices, and concurrently omitting the gating network. This transition facilitates full-parameter optimization, effectively mitigating the representation collapse issue caused by sparse training and enabling the evolution of activation patterns.

Discussion of Sparse Approximation. Although we can approximate the model with a parameter-equivalent SMoE model, the approximation only holds for forward propagation. If we skip the computation of some neurons during forward propagation, the gradients of these neurons will be zero during backward propagation. However, the gradients of these neurons are usually not zero in the original model, posing an inconsistency between the SMoE model and the original model during backward propagation. We argue that this inconsistency may not be a problem. Intuitively, the inactivated neurons do not have strong relationships with the input, and the gradients of these neurons are not important, as the idea of Hebbian learning (Seung, 2000) that focuses on the neurons that are activated by the input.

4.3. Transition Time Determination

(1) **Dense-to-Sparse Conversion.** Considering the dynamic nature of the model activation during pre-training, we conduct the conversion when the activation sparsity is high and the activation pattern is stable. Inspired by the observation in Section 3, we propose to monitor the activation pattern change to determine the transition time, where the activation pattern similarity reflects the changing rate of the activation pattern. Specifically, we set a threshold τ and switch to sparse training when the activation pattern similarity between two consecutive checkpoints is larger than τ. (2) **Sparse-to-Dense Conversion.** To have a controllable speed ratio, we propose to maintain a constant ratio of sparse training steps to all training steps r. For example, if $r = 0.5$, we will train the model with 50% of the training data in the sparse training phases and the remaining 50% in the dense training phases. Specifically, we set the steps of sparse training to $T = \frac{r}{1-r}$ times the steps of the last dense training. We give a detailed example in Figure 6. And, to ensure the final model can be used densely, we adopt dense training at the end of the training. Hence, the pre-training process consists of several dense-sparse cycles and one final dense training. We will pre-define the ratio of the final

dense training steps as l, and the adjusted T is calculated as $T = \frac{r}{1-r-l}$.

5. Experiments

5.1. Experimental Setup

Here we briefly introduce the setups of our experiments. Please refer to Appendix A for more details.

Baselines. We compare our method with the following baselines: (1) *Dense*: We compute and update all the parameters in the network, which is the default way of pre-training. (2) *SMoE*: We replace the FFNs in the *Dense* baseline with MoE layers and train the model in a sparsely-activated manner (Fedus et al., 2022). Specifically, the number of parameters in experts (not including router networks) is the same as the number of parameters in the FFNs. The final model is computed sparsely. (3) *Progressive Layer Dropping (PLD)* (Zhang & He, 2020): PLD randomly drops layers during training to reduce the costs. After pre-training, we use all the layers of the final model, i.e., the same as the *Dense* baseline. (4) *MoE-Dropout* (Chen et al., 2023a): At the beginning of pre-training, the model is an SMoE model. During the training, MoE-Dropout gradually increases the number of selected experts K to the number of experts N. The final model is also densely computed.

Datasets. (1) *Pre-training corpus*: We use the Pile dataset (Gao et al., 2021a) as the pre-training corpus. Due to limited computational resources, we use the first part of the Pile dataset with over 27GB of text data. (2) *Downstream tasks*: We consider two kinds of downstream tasks, i.e., natural language understanding and instruction tuning. More details are in Appendix A.

Training Details. (1) *Model architecture*: In this work, we evaluate our method on all three variants of Transformers, i.e., encoder-only BERT, decoder-only GPT, and encoder-decoder T5. For these models, we use the base version with 12 layers, 768 hidden size, and 12 attention heads for each encoder/decoder. Following Chen et al. (2023a), we set the intermediate size of FFNs to 6,144. For MoE-Dropout and SSD, we set the number of experts to 32 and the number of selected experts to 6. For SMoE, we set the number of experts to 3 and the number of selected experts to 2 to ensure similar computational costs to MoE-Dropout and SSD. (2) *Pre-training*: The training epoch is set to 10, which contains about 200,000 steps, and the warmup steps are set to 2,000. BERT and T5 adopt masked language modeling (MLM) as the pre-training task, and GPT adopts causal language modeling (CLM) as the pre-training task. (3) *Fine-tuning*: In this stage, we use two fine-tuning strategies for SMoE, i.e., sparse fine-tuning and dense fine-tuning, which are denoted as SMoE and SMoE (D), respectively. For the other models, we only use dense fine-tuning. We run each experiment 5 times and report the average of their best results on the development set. For the instruction tuning task, which contains zero-shot transfer, we use the best checkpoint of each run on the development set for evaluation.

5.2. Main Results

In this subsection, we study the training efficiency and inference efficiency of different methods.

Training Efficiency. We report the computational costs per batch and performance of different methods in Figure 3. For model performance, we evaluate perplexity on a held-out validation corpus. From this figure, we have three observations. (1) Although SMoE training can reduce the computational costs, the perplexity of SMoE is consistently higher than that of dense training. (2) PLD and MoE-Dropout can also reduce the cost while keeping the perplexity comparable to that of dense training. However, in some cases, the perplexity of PLD and MoE-Dropout is higher than that of dense training, such as GPT with PLD and T5 with MoE-dropout. (3) SSD has the same speedup as MoE-dropout (up to $1.44\times$) and achieves slightly lower or equal perplexity compared with dense training, i.e., the data points of SSD are placed at the bottom left corner of the figure. It indicates that SSD achieves the best trade-off between training costs and performance.

Inference Efficiency. We conduct experiments to investigate whether these dense models can be computed sparsely for efficient inference without additional training. We consider three methods, i.e., Dense, MoE-Dropout, and SSD, and vary the number of selected experts k from 1 to 32, which is the total number of experts. For the models pre-trained with Dense, we use MoEfication to transform them into SMoE models. For the models pre-trained with MoE-Dropout and SSD, we directly adopt their MoE structure used in pre-training. Note that we also try to use MoEfication to transform the models pre-trained with MoE-Dropout and SSD into SMoE models, but the performance is slightly worse than that of directly using their original MoE structure. Additionally, we also report the performance of the models pre-trained with SMoE. We also try to vary the number of selected experts k for the models pre-trained with SMoE, but the performance is consistently worse than that of using the original number of selected experts. Therefore, we do not report the results of SMoE with different k.

We report the perplexity on the validation set with different computational costs per batch in Figure 4(a). From this figure, we have three observations. (1) The performance of the models pre-trained with Dense is consistently worse than that of the models pre-trained with SMoE, MoE-Dropout, and SSD. Both MoE-Dropout and SSD adopt sparse training during pre-training, which makes them more suitable for sparse inference. It indicates that **computing sparsely dur-**

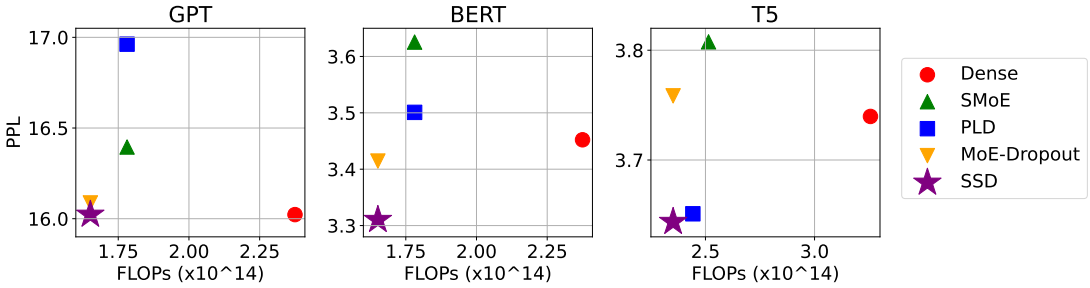
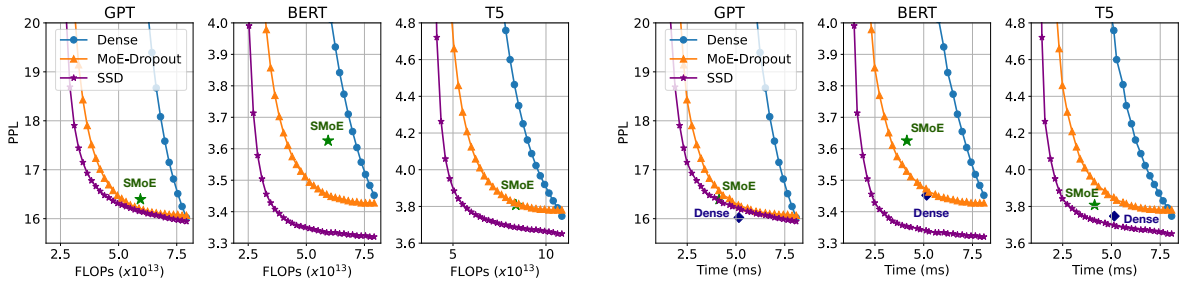


Figure 3. Computational costs (FLOPs) of pre-training and perplexity (PPL) on the validation set of different methods on three representative models. Smaller computational cost means better efficiency and smaller perplexity means better performance.



(a) Perplexity w.r.t computational costs.

(b) Perplexity w.r.t inference time.

Figure 4. Perplexity on the validation set with different computational costs and inference time by varying the number of selected experts. For comparison, we also scatter the results of SMoE with its default number of selected experts and dense models without any sparsity.

ing pre-training is necessary for sparse inference. (2) With the same computational cost, the performance of the models pre-trained with MoE-Dropout and SSD is overall better than that of the models pre-trained with SMoE. It shows the potential of building various SMoE models with different computational costs based on a single dense model instead of training multiple SMoE models with different numbers of experts from scratch. (3) The curve of SSD is always below that of MoE-Dropout, which indicates that SSD achieves a better trade-off between costs and performance than MoE-Dropout during inference.

We further investigate the inference time with different numbers of selected experts in Figure 4(b). Specifically, we report the inference time of a single MoE layer with different numbers of selected experts. The inference time is measured on a single NVIDIA RTX 3090 GPU, which is a popular GPU for LLM inference, with a batch size of 32 and a sequence length of 512. The MoE layer is implemented with the ScatterMoE library (Tan et al., 2024). From this figure, we observe that with the same inference time, the performance of the models pre-trained with SSD is still better than that of the models pre-trained with SMoE and MoE-Dropout. Besides, since the MoE implementation is different from the original FFN implementation, the inference time of dense models without any sparsity (the diamond points with dark blue color)is shorter than that of

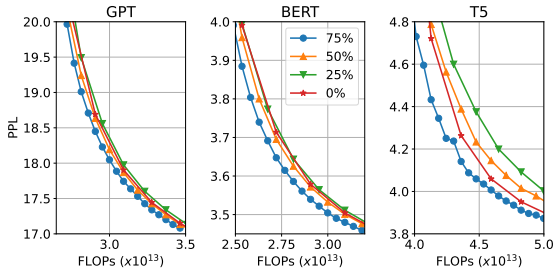


Figure 5. Perplexity on the validation set with different computational costs by truncating the experts with small importance scores.

the MoEfied dense models with large k. Despite this, the performance of SSD models is still better than that of dense models without any sparsity in the cases of BERT and T5 with the same inference time. Moreover, BERT trained with **SSD achieves the same performance as that of the dense model with less than half the inference time** (2.3ms vs. 5.1ms). It indicates that SSD can provide competitive performance and efficiency compared to original dense models and other sparse models during inference.

5.3. Dynamic Top-k

Based on SSD, we further investigate whether we can vary the number of selected experts for each token in an input

Table 1. Evaluation results (%) on natural language understanding tasks. MoE-D refers to MoE-Dropout. SMoE (D) refers to densely fine-tuned SMoE. Since the training costs of SMoE is smaller than other methods, we color the results of SMoE in gray.

	SST2 Acc.	SNLI Acc.	MNLI Acc.	QNLI Acc	QQP Acc.	SQuAD F1	Avg.
BERT-based Models							
Dense	90.0	88.4	80.1	87.2	89.7	78.9	83.2
SMoE	84.3	83.9	71.4	82.3	85.9	68.8	76.2
SMoE (D)	84.7	84.5	71.8	82.8	86.5	70.9	77.2
PLD	90.0	88.3	79.5	86.6	89.8	77.9	82.8
MoE-D	90.7	88.5	80.7	87.4	89.8	76.8	82.8
SSD	90.6	88.7	80.6	88.7	90.0	78.8	83.6
T5-based Models							
Dense	91.5	89.4	81.7	88.8	90.2	83.9	85.9
SMoE	86.0	87.5	77.9	84.2	88.3	78.5	81.6
SMoE (D)	86.2	87.8	78.6	84.7	88.6	78.8	82.0
PLD	92.1	89.5	82.8	89.9	90.4	85.1	86.7
MoE-D	91.8	89.4	82.3	89.5	90.4	83.6	86.0
SSD	92.5	89.8	82.5	89.5	90.5	85.0	86.6

Table 2. Evaluation results (%) on instruction tuning. MoE-D refers to MoE-Dropout. SMoE (D) refers to densely fine-tuned SMoE. "Dev" represents the development set and "NI" represents Super-NaturalInstructions. Since the training costs of SMoE are smaller than other methods, we color the results of SMoE in gray.

	Dev Rouge-L	NI Rouge-L	Avg.
GPT-based Models			
Dense	19.2	16.7	18.0
SMoE	17.6	15.9	16.8
SMoE (D)	18.3	17.9	18.1
PLD	19.1	17.5	18.3
MoE-D	18.8	18.2	18.5
SSD	19.4	18.0	18.7
T5-based Models			
Dense	19.7	19.1	19.4
SMoE	16.7	15.8	16.3
SMoE (D)	17.0	16.3	16.7
PLD	19.5	20.1	19.8
MoE-D	18.6	18.9	18.7
SSD	18.6	20.4	19.5

sequence. For example, some important tokens may need more experts to compute, and some unimportant tokens may need fewer experts to compute. Specifically, we first compute the top-k experts for each token in the input sequence as the expert candidates. Then, we truncate the experts with small importance scores based on a given ratio. We report the perplexity on the validation set with different truncation ratios in Figure 5. From this figure, we observe that truncating 75% of the experts can consistently achieve better performance than that of using a fixed number of experts for each token under the same computational cost. It opens up a new direction for future research to dynamically identify the number of experts for each token in an SMoE model.

5.4. Performance on Downstream Tasks

We report the model performance on natural language understanding tasks in Table 1, focusing on BERT and T5. Besides, We report the model performance on instruction tuning in Table 2, focusing on GPT and T5. The results with variance are reported in Appendix B. From these tables, we have three observations. (1) The perplexity is consistent with the overall performance on downstream tasks. For example, PLD and SSD achieve the lowest perplexity on T5, and they also achieve the top two overall performances on downstream tasks. The same phenomenon also appears on BERT and GPT. It indicates that the perplexity can be used as a good performance indicator on downstream tasks, which is also shown by Brown et al. (2021); Gordon et al. (2020). (2) Densely fine-tuning SMoE can achieve better performance than sparsely fine-tuning SMoE while still being worse than other methods. It indicates that pre-trained SMoE models cannot fully utilize the model capacity even with dense fine-tuning. (3) SSD achieves slightly better overall performance than dense training on all evaluation settings, the only one that achieves this result among the acceleration methods. It demonstrates the general applicability of SSD to different models and tasks.

5.5. Scalability

Due to the limited computational resources, we assess the scalability of SSD on large models by continual pre-training, which requires fewer training steps than pre-training from scratch. Specifically, we continue pre-training Persimmon-8B on a diverse Chinese corpus, containing encyclopedia, news, books, and web texts. Persimmon-8B (Elsen et al., 2023) is a sparse-activated large language model (LLM) and has competitive performance to LLaMA-7B (Touvron et al., 2023b) on several evaluation benchmarks. We compare SSD with dense training under the same training steps with nearly 4 billion tokens. The details of the training setup are shown in Appendix A. We evaluate the model performance on C-Eval (Huang et al., 2023), a widely-used Chinese benchmark and report the average accuracy of subtasks in Table 3. From this table, we observe that SSD also achieves comparable performance to dense training, which demonstrates the scalability of SSD to LLMs.

5.6. Speed Analysis

We report the average computation time of the original dense training and SMoE-based training in Table 4. We use four NVIDIA A800 GPU for training and adopt MegaBlocks (Gale et al., 2022) as the SMoE implementa-

Table 3. Performance (%) of Persimmon-8B on C-Eval. "Social" represents social science, "Human" represents humanities, and "Other" represents other subjects.

	STEM	Social	Human	Other	Avg.
Original	21.0	23.6	28.2	21.3	23.5
Dense	28.7	35.5	30.1	31.6	31.5
SSD	30.8	34.1	32.7	30.3	32.0

tion. From this table, we observe that SSD achieve better time speedup on the larger model, i.e., Persimmon-8B. This enhancement is attributed to the higher GPU utilization facilitated by the larger model, making the time speedup more obvious. It highlights the promising speedup potential of SSD on LLMs. However, a discrepancy is noted between theoretical speedup and actual time. A deeper analysis into time consumption across different operations reveals that the expert selection process incurs additional time, thereby presenting an avenue for future research to optimize the computation of SMoE. Although the time speedup of SSD is not fully matched with the theoretical speedup, the time reduction is indeed significant because the pre-training cost is huge and it would be increasingly valuable for large models.

Table 4. Speedup of SSD compared with dense training on the computational costs and times (hours).

	GPT		Persimmon-8B	
	Dense	SSD	Dense	SSD
FLOPs	237T	178T (1.44×)	2.97P	2.25P (1.32×)
Time	23.6h	21.5h (1.10×)	166h	134h (1.23×)

Here we also report the training time of other methods on GPT for comparison. PLD takes 20.3 hours, MoE-Dropout takes 35.9 hours, and SSD takes 21.5 hours. It show that our approach achieves competitive speedups compared with other methods. It worth noting that the time speedups of acceleration methods can be affected by their specific implementations, making it challenging to directly compare their time speedups, so that we mainly focus on the FLOP speedups in the main text.

5.7. Ablation Study

In this subsection, we conduct ablation studies on the transition time determination and transition method. (1) For the transition time determination, we compare the threshold-based method with the random method. In the random method, we will switch the computation mode at each monitoring step with a probability of 0.5. We conduct this experiment on GPT three times and report the perplexity in Table 5. From this table, we observe that the random method is consistently worse than the threshold-based method, which demonstrates the effectiveness of the threshold-based method. The reason is that the random method may switch at the beginning of pre-training when the activation pattern is unstable, or switch frequently in a certain period, leading to unstable training. (2) For the transition method, we choose one switch point of GPT pre-training as the original model and apply different transition methods to it, including random, clustering, and SSD. In the random method, we randomly split the neurons into groups. In the clustering method, we directly use the clustering method in Section 4.2 without the previous checkpoint initialization as MoEfication does. From Table 5, we observe that SSD achieves the smallest gap compared with the original model, which demonstrates the effectiveness of checkpoint initialization in SSD. This minimal gap facilitates a smoother transition process, potentially culminating in superior performance.

Table 5. Impact of transition time determination and transition method on the model perplexity.

PPL	Random			SSD
	17.3	18.6	19.8	**16.0**

PPL	Original	Random	Clustering	SSD
	18.4	3533.3	20.9	**20.4**

6. Conclusion

In this work, we utilize the phenomenon of sparse activation to accelerate pre-training and inference of LLMs. Specifically, we propose Switchable Sparse-Dense Learning, which adaptively switches between sparse and dense training. Experimental results on three different model architectures and two kinds of downstream tasks show that our method can achieve comparable performance to dense training with less computational costs. Moreover, the models trained with SSD can be directly used as MoE models for inference and reduce the inference time of FFNs by up to $2\times$ while keeping the performance as good as dense models. We hope that our work can provide a new perspective for the acceleration based on sparse activation and inspire more research in this direction.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

Acknowledgement

This work is supported by the National Key R&D Program of China (No. 2022ZD0116312), Major Project of the National Social Science Foundation of China

(No. 228ZD298), Quan Cheng Laboratory (Grant No. QCLZD202301), National Natural Science Foundation of China (No. 623B1019), Institute Guo Qiang at Tsinghua University and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

The method is primarily designed by Zhengyan Zhang, and he conducted most of the experiments. Chaojun Xiao, Qiujieli Qin, Yankai Lin, Zhiyuan Zeng, Xu Han, Zhiyuan Liu and Ruobing Xie actively participated in experimental design, result analysis, and paper writing. Maosong Sun led the project and provided guidance. Jie Zhou provided valuable advice to this research.

References

Albericio, J., Judd, P., Hetherington, T. H., Aamodt, T. M., Jerger, N. D. E., and Moshovos, A. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Proceedings of ISCA*, pp. 1–13, 2016.

Alizadeh, K., Mirzadeh, I., Belenko, D., Khatamifard, K., Cho, M., Mundo, C. C. D., Rastegari, M., and Farajtabar, M. LLM in a flash: Efficient large language model inference with limited memory. *arXiv preprint arXiv:2312.11514*, 2023.

Artetxe, M., Bhosale, S., Goyal, N., Mihaylov, T., Ott, M., Shleifer, S., Lin, X. V., Du, J., Iyer, S., Pasunuru, R., Anantharaman, G., Li, X., Chen, S., Akin, H., Baines, M., Martin, L., Zhou, X., Koura, P. S., O'Horo, B., Wang, J., Zettlemoyer, L., Diab, M., Kozareva, Z., and Stoyanov, V. Efficient large scale language modeling with mixtures of experts. In *Proceedings of EMNLP*, pp. 11699–11732, 2022.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*, pp. 632–642, 2015.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are Few-Shot learners. In *Proceedings of NeurIPS*, pp. 1877–1901, 2021.

Chen, C., Yin, Y., Shang, L., Jiang, X., Qin, Y., Wang, F., Wang, Z., Chen, X., Liu, Z., and Liu, Q. bert2bert: Towards reusable pretrained language models. In *Proceedings of ACL*, pp. 2134–2148, 2022.

Chen, T., Zhang, Z., JAISWAL, A. K., Liu, S., and Wang, Z. Sparse moe as the new dropout: Scaling dense and self-slimmable transformers. In *Proceedings of ICLR*, 2023a.

Chen, W., Zhou, Y., Du, N., Huang, Y., Laudon, J., Chen, Z., and Cui, C. Lifelong language pretraining with distribution-specialized experts. In *Proceedings of ICML*, 2023b.

Chi, Z., Dong, L., Huang, S., Dai, D., Ma, S., Patra, B., Singhal, S., Bajaj, P., Song, X., Mao, X., Huang, H., and Wei, F. On the representation collapse of sparse mixture of experts. In *Proceedings of NeurIPS*, 2022.

Clark, K., Luong, M., Le, Q. V., and Manning, C. D. ELECTRA: pre-training text encoders as discriminators rather than generators. In *Proceedings of ICLR*, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL*, pp. 4171–4186, 2019.

Dong, H., Chen, B., and Chi, Y. Towards structured sparsity in transformers for efficient inference. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*, 2023.

Elsen, E., Odena, A., Nye, M., Taşırlar, S., Dao, T., Hawthorne, C., Moparthi, D., and Somani, A. Releasing Persimmon-8B, 2023.

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

Gale, T., Narayanan, D., Young, C., and Zaharia, M. Megablocks: Efficient sparse training with mixture-of-experts. In *Proceedings of ICML*, 2022.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. The pile: An 800gb dataset of diverse text for language modeling. *arxiv preprint arXiv:2101.00027*, 2021a.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., and Zou, A. A framework for few-shot language model evaluation, September 2021b.

Gao, Z.-F., Liu, P., Zhao, W. X., Lu, Z.-Y., and Wen, J.-R. Parameter-efficient mixture-of-experts architecture for pre-trained language models. In *Proceedings of COLING*, pp. 3263–3273, 2022.

Gong, L., He, D., Li, Z., Qin, T., Wang, L., and Liu, T. Efficient training of BERT by progressively stacking. In *Proceedings of ICML*, pp. 2337–2346, 2019.

Gordon, M., Duh, K., and Andrews, N. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Proceedings of RL4NLP*, pp. 143–155, 2020.

Gururangan, S., Lewis, M., Holtzman, A., Smith, N. A., and Zettlemoyer, L. DEMix layers: Disentangling domains for modular language modeling. In *Proceedings of NAACL-HLT*, pp. 5557–5576, 2022.

Hazimeh, H., Zhao, Z., Chowdhery, A., Sathiamoorthy, M., Chen, Y., Mazumder, R., Hong, L., and Chi, E. H. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. In *Proceedings of NeurIPS*, pp. 29335–29347, 2021.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint 1606.08415*, 2016.

Honovich, O., Scialom, T., Levy, O., and Schick, T. Unnatural instructions: Tuning language models with (almost) no human labor. *arxiv preprint arXiv:2212.09689*, 2022.

Huang, Y., Bai, Y., Zhu, Z., Zhang, J., Zhang, J., Su, T., Liu, J., Lv, C., Zhang, Y., Lei, J., Fu, Y., Sun, M., and He, J. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *arXiv preprint arXiv:2305.08322*, 2023.

Hwang, C., Cui, W., Xiong, Y., Yang, Z., Liu, Z., Hu, H., Wang, Z., Salas, R., Jose, J., Ram, P., Chau, J., Cheng, P., Yang, F., Yang, M., and Xiong, Y. Tutel: Adaptive mixture-of-experts at scale. *arXiv preprint arXiv:2206.03382*, 2022.

Izsak, P., Berchansky, M., and Levy, O. How to train BERT with an academic budget. In *Proceedings of EMNLP*, pp. 10644–10652, 2021.

Jang, J., Kim, S., Ye, S., Kim, D., Logeswaran, L., Lee, M., Lee, K., and Seo, M. Exploring the benefits of training expert language models over instruction tuning. In *Proceedings of ICML*, 2023.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de Las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

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

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015.

Lee-Thorp, J. and Ainslie, J. Sparse mixers: Combining MoE and mixing to build a more efficient BERT. In *Findings of EMNLP*, pp. 58–75, 2022.

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. BASE layers: Simplifying training of large, sparse models. In *Proceedings of ICM*, volume 139, pp. 6265–6274, 2021.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S. J., Ye, K., Chern, F., Yu, F., Guo, R., and Kumar, S. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *Proceedings of ICLR*, 2023.

Lin, C.-Y. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, July 2004.

Liu, R., Kim, Y. J., Muzio, A., and Hassan, H. Gating dropout: Communication-efficient regularization for sparsely activated transformers. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of ICML, 2022.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Ré, C., and Chen, B. Deja vu: Contextual sparsity for efficient llms at inference time. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Preceedings of ICML*, volume 202, pp. 22137–22176, 2023.

Malinen, M. I. and Fränti, P. Balanced k-means for clustering. In *Proceedings of SSSPR*, volume 8621, pp. 32–41, 2014.

Mirzadeh, I., Alizadeh, K., Mehta, S., Mundo, C. C. D., Tuzel, O., Samei, G., Rastegari, M., and Farajtabar, M. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*, 2023.

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

Pan, B., Shen, Y., Liu, H., Mishra, M., Zhang, G., Oliva, A., Raffel, C., and Panda, R. Dense training, sparse inference: Rethinking training of mixture-of-experts language models. *arXiv preprint arXiv:2404.05567*, 2024.

Qin, Y., Lin, Y., Yi, J., Zhang, J., Han, X., Zhang, Z., Su, Y., Liu, Z., Li, P., Sun, M., and Zhou, J. Knowledge inheritance for pre-trained language models. In *Proceedings of NAACL-HLT*, pp. 3921–3937, 2022.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified Text-to-Text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of ICML*, 2022.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pp. 2383–2392. Association for Computational Linguistics, 2016.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Pinto, A. S., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. In *Proceedings of NeurIPS*, 2021.

Roller, S., Sukhbaatar, S., Szlam, A., and Weston, J. Hash layers for large sparse models. In *Proceedings of NeurIPS*, pp. 17555–17566, 2021.

Seung, H. S. Half a century of hebb. *Nature neuroscience*, 3(11):1166–1166, 2000.

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

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pp. 1631–1642, 2013.

Tan, S., Shen, Y., Panda, R., and Courville, A. C. Scattered mixture-of-experts implementation. *arXiv preprint arXiv:2301.11514*, 2024.

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

Touvron, H., Martin, L., Stone, K., Albert, P., et al. Llama 2: Open foundation and fine-tuned chat models. *arxiv preprint arXiv:2307.09288*, 2023b.

Vaswani, A., Shazeer, N., Parmar, N., and Uszkoreit, J. Attention is all you need. In *Proceedings of NeurIPS*, pp. 5998–6008, 2017a.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of NeurIPS*, pp. 5998–6008, 2017b.

Wang, X., Wen, K., Zhang, Z., Hou, L., Liu, Z., and Li, J. Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of EMNLP*, 2022a.

Wang, Y., Mishra, S., Alipoormolabashi, P., Kordi, Y., Mirzaei, A., Arunkumar, A., Ashok, A., Dhanasekaran, A. S., Naik, A., Stap, D., et al. Super-naturalinstructions:generalization via declarative instructions on 1600+ tasks. In *EMNLP*, 2022b.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, 2018.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *Proceedings of ICML*, pp. 10524–10533, 2020.

Yang, Q., Zhang, K., Lan, C., Yang, Z., Li, Z., Tan, W., Xiao, J., and Pu, S. Unified normalization for accelerating and stabilizing transformers. In *Proceedings of MM*, pp. 4445–4455, 2022.

Zhang, M. and He, Y. Accelerating training of transformer-based language models with progressive layer dropping. In *Proceedings of NeurIPS*, 2020.

Zhang, X., Shen, Y., Huang, Z., Zhou, J., Rong, W., and Xiong, Z. Mixture of attention heads: Selecting attention heads per token. In *Proceedings of EMNLP*, pp. 4150–4162, 2022a.

Zhang, Z., Lin, Y., Liu, Z., Li, P., Sun, M., and Zhou, J. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of ACL*, 2022b.

Zhang, Z., Zeng, Z., Lin, Y., Xiao, C., Wang, X., Han, X., Liu, Z., Xie, R., Sun, M., and Zhou, J. Emergent modularity in pre-trained transformers. In *Findings of ACL*, 2023.

Zhang, Z., Song, Y., Yu, G., Han, X., Lin, Y., Xiao, C., Song, C., Liu, Z., Mi, Z., and Sun, M. Relu^2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024.

Zuo, S., Liu, X., Jiao, J., Kim, Y. J., Hassan, H., Zhang, R., Gao, J., and Zhao, T. Taming sparsely activated transformer with stochastic experts. In *Proceedings of ICLR*, 2022a.

Zuo, S., Zhang, Q., Liang, C., He, P., Zhao, T., and Chen, W. MoEBERT: from BERT to mixture-of-experts via importance-guided adaptation. In *Proceedings of NAACL-HLT*, pp. 1610–1623, 2022b.

A. Other Experimental Details

Model Architecture. We use layer normalization before each attention and FFN layer (Xiong et al., 2020), which is beneficial for convergence (Izsak et al., 2021), and use ReLU as the activation function for easier MoE transformation than GeLU (Hendrycks & Gimpel, 2016; Zhang et al., 2022b).

Pre-training. We use part of the Pile dataset (Gao et al., 2021a) as the pre-training corpus. Based on observations from previous work (Zhang et al., 2023), the emergence of sparse activations appears to be more closely tied to the inherent training dynamics arising from the model architecture itself, rather than the type or scale of the data used. Therefore, we believe that using a subset of the Pile dataset for our pretraining experiments does not significantly impact the degree of activation sparsity observed. We use the Adam optimizer (Kingma & Ba, 2015) and Noam learning rate scheduler (Vaswani et al., 2017a) for pre-training. The batch size is set to 512 and the learning rate is set to 1 for BERT and T5 and 0.5 for GPT. The mask rate of MLM is set to 0.15. PLD increases the overall dropout rate from 0 to 0.25 quickly at the beginning of training and then keeps it at 0.25. For the MoE layers, we set the number of experts N to 32 for MoE-Dropout and SSD. MoE-Dropout linearly increases the number of selected experts K from 6 to 32 during the pre-training. For SSD, we set the threshold τ to 0.9 and monitor the activation pattern every 3,000 steps. In the sparse mode, we also select 6 experts for each layer. The ratio of the sparse mode r is set to 0.5. The ratio of the final dense training l is set to 0.1. For SMoE, we set the number of experts N to 3 and the number of selected experts K to 2 to ensure the computational cost is similar to that of other methods.

Persimmon-8B. For the continual pre-training of Persimmon-8B, we set the batch size to 2,048, the learning rate to 0.00003, and the max sequence length to 1024. The total pre-training steps are set to 128,000 and the gradient accumulation steps are set to 64. As a result, the total number of optimization steps is 2000. We use the same ratio of the sparse mode $r = 0.5$ as the experiments on base-scale models. Since the total optimization steps are smaller than training from scratch, we divide the SSD training into two stages: the first stage is 1000 steps of sparse training and the second stage is 1000 steps of dense training. We compare the SSD performance on Persimmon-8B with the dense training performance on Persimmon-8B with the same optimization steps. The total expert number N is set to 64 and the selected expert number K is set to 16, keeping the expert size the same as that of the base models. The other hyperparameters are the same as those of base models. When evaluating the performance of Persimmon-8B, we use LM Evaluation Harness (Gao et al., 2021b).

Downstream Tasks. First, we evaluate models on several natural language understanding tasks. For single-sentence classification, we use SST-2 (Socher et al., 2013), which is a sentiment analysis dataset. For sentence-pair classification, we use SNLI (Bowman et al., 2015), MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), and QQP[3], covering the tasks of natural language inference and paraphrase identification. For reading comprehension, we use SQuAD v1.1 (Rajpurkar et al., 2016), which is a widely used dataset for extractive question answering. Second, we evaluate models on instruction tuning. Specifically, we follow the setups of Honovich et al. (2022), where the model is trained on a model-generated instruction dataset and evaluated on several human-labeled instruction datasets. The training dataset is Unnatural Instructions (Honovich et al., 2022) and the development dataset contains 1,000 randomly sampled instances from the training set of Super-NaturalInstructions (Wang et al., 2022b), which is used to select the best checkpoint. The test dataset is the test set of Super-NaturalInstructions (Wang et al., 2022b). We conduct a grid search to find the best hyperparameters for each model, including the learning rate varied from 4e-4 to 2e-3, the batch size varied from 16 to 32, and the number of training epochs varied from 3 to 10.

SMoE Implementation. There are some frameworks optimized for SMoE, such as DeepSpeed-MoE (Rajbhandari et al., 2022), MegaBlocks (Gale et al., 2022), and Tutel (Hwang et al., 2022). Among them, we implement SMoE based on the MegaBlocks framework, which supports efficient dropless MoE layers.

Evaluation of Instruction Tuning. Following Honovich et al. (2022), we adopt greedy decoding to generate the responses to the instructions. Super-NaturalInstructions (Wang et al., 2022b) is evaluated by Rouge-L (Lin, 2004).

WCSS Computation. In the paper, we follow the standard formulation for calculating WCSS: $\sum_{k=1}^{K}\sum_{i\in S_k}\sum_{j=1}^{p}(x_{ij} - \bar{x}_{kj})^2$, where S_k is the set of data points assigned to cluster k, x_{ij} is the j-th feature value of data point i, and $\bar{x}_{kj}$ is the mean value of feature j across all points in cluster k. In our case of WCSS clustering, x_i is the i-th row of $\boldsymbol{W}_i$.

Post-processing of Expert Scores. In the paper, we set the α_n of selected experts to 1 through post-processing. Specifically, we compute $\alpha = 1 + \alpha - \alpha$.detach() in Pytorch to ensure the gradient backpropagation.

[3] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

Clustering. The clustering operation is performed on the GPU using the faiss-gpu library (Johnson et al., 2019). To ensure the balance of the number of neurons in each expert, we further apply the balanced assignment strategy (Lewis et al., 2021) to the results of k-means clustering. The time of each clustering is less than 1 minute and has little impact on the training time.

Table 6. Evaluation results (%) on natural language understanding tasks. MoE-D refers to MoE-Dropout. SMoE (D) refers to densely fine-tuned SMoE. Since the training costs of SMoE are smaller than other methods, we color the results of SMoE in gray. The last column represents the average of all previous columns.

	SST2 Acc.	SNLI Acc.	MNLI-m Acc.	QNLI Acc	QQP Acc.	SQuAD EM	SQuAD F1	Avg.
BERT-based Models								
Dense	90.0±0.5	88.4±0.1	80.1±0.1	87.2±0.4	89.7±0.1	68.4±0.2	78.9±0.2	83.2
SMoE	84.3±0.2	83.9±0.2	71.4±0.7	82.3±0.2	85.9±0.7	56.8±1.0	68.8±0.7	76.2
SMoE (D)	84.7±1.0	84.5±0.1	71.8±0.2	82.8±0.1	86.5±0.4	59.0±1.0	70.9±0.8	77.2
PLD	90.0±0.4	88.3±0.2	79.5±0.3	86.6±0.4	89.8±0.1	67.3±0.3	77.9±0.3	82.8
MoE-D	90.7±0.3	88.5±0.2	80.7±0.2	87.4±0.3	89.8±0.1	65.9±0.4	76.8±0.7	82.8
SSD	90.6±0.4	88.7±0.1	80.6±0.4	88.7±0.4	90.0±0.1	68.1±0.3	78.8±0.1	83.6
T5-based Models								
Dense	91.5±0.5	89.4±0.2	81.7±0.1	88.8±0.2	90.2±0.1	75.5±0.4	83.9±0.3	85.9
SMoE	86.0±0.1	87.5±0.1	77.9±0.4	84.2±0.0	88.3±0.2	68.8±0.3	78.5±0.3	81.6
SMoE (D)	86.2±0.6	87.8±0.2	78.6±0.5	84.7±0.2	88.6±0.1	69.1±0.3	78.8±0.2	82.0
PLD	92.1±0.1	89.5±0.1	82.8±0.1	89.9±0.1	90.4±0.1	76.8±0.3	85.1±0.4	86.7
MoE-D	91.8±0.3	89.4±0.1	82.3±0.1	89.5±0.2	90.4±0.1	74.9±0.2	83.6±0.1	86.0
SSD	92.5±0.2	89.8±0.1	82.5±0.1	89.5±0.1	90.5±0.1	76.5±0.4	85.0±0.2	86.6

B. Results with Standard Deviation

We report the evaluation results with standard deviation on natural language understanding tasks in Table 6 and instruction tuning tasks in Table 7.

Table 7. Evaluation results (%) on instruction tuning. MoE-D refers to MoE-Dropout. SMoE (D) refers to densely fine-tuned SMoE. "Dev" represents the development set and "NI" represents Super-NaturalInstructions. Since the training costs of SMoE are smaller than other methods, we color the results of SMoE in gray. The last column represents the average result.

	Dev Rouge-L	NI Rouge-L	Avg.
GPT-based Models			
Dense	19.2±0.3	16.7±0.6	18.0
SMoE	17.6±0.4	15.9±0.8	16.8
SMoE (D)	18.3±0.5	17.9±0.9	18.1
PLD	19.1±0.7	17.5±0.4	18.3
MoE-D	18.8±0.5	18.2±1.7	18.5
SSD	19.4±0.4	18.0±0.9	18.7
T5-based Models			
Dense	19.7±0.3	19.1±0.8	19.4
SMoE	16.7±0.2	15.8±0.2	16.3
SMoE (D)	17.0±0.7	16.3±1.0	16.7
PLD	19.5±0.3	20.1±0.7	19.8
MoE-D	18.6±0.3	18.9±0.7	18.7
SSD	18.6±0.2	20.4±0.6	19.5

C. Case Study

To better understand the training process of SSD, we visualize the change of evaluation perplexity and the stages of sparse mode in Figure 6. From this figure, we observe that there are three different lengths of sparse stages: the first one is the

longest (22,500 steps), the second and third ones are the second longest (7,500 steps), and the rest are the shortest (3,750 steps). Here the proportion of the sparse mode to the dense mode is set to 1.25. The first dense training has 18,000 steps, followed by 22,500 (18000×1.25) sparse training steps. The second dense training has 6,000 steps, followed by 7,500 (6000×1.25) sparse training steps. It reveals the change of activation pattern during pre-training, i.e., the activation pattern changes dramatically at the beginning and then becomes stable. When the activation pattern is unstable, the length of the dense stage is long, and the corresponding sparse stage is also long, vice versa. Mixing dense training with sparse training will slow down the learning of the model, and the perplexity is higher than that of dense training in the middle stage. However, the perplexity of SSD can quickly catch up with that of dense training after the final dense training. Note that at the same step, the computational cost of SSD is much lower than that of dense training, which indicates that SSD can achieve a good trade-off between computational cost and model performance.

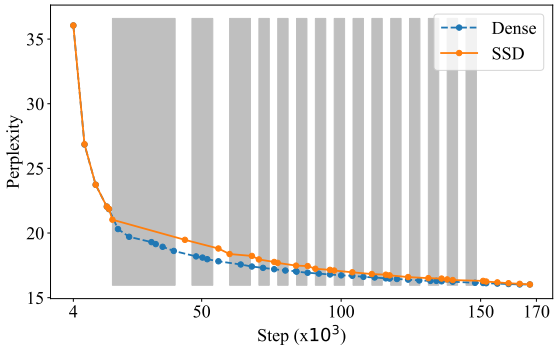


Figure 6. Change of evaluation perplexity during pre-training. The model architecture is GPT. The gray areas represent the stages of sparse mode for SSD.

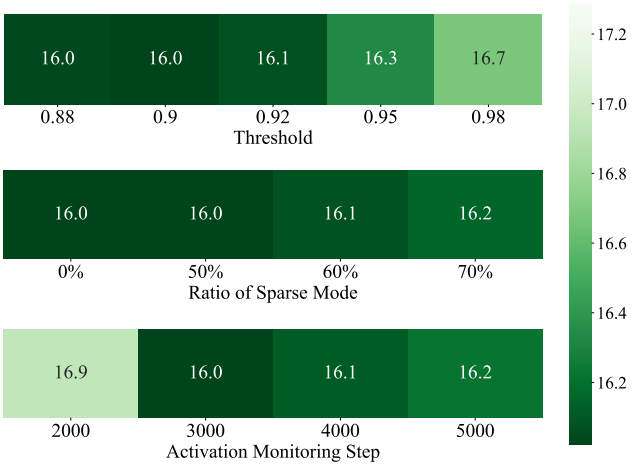


Figure 7. Effect of SSD hyperparameters on perplexity. The model architecture is GPT.

D. Hyperparameter Analysis

In this section, we evaluate the effect of the conversion threshold τ, the ratio of the sparse mode r, and the steps of monitoring the activation pattern on the performance of GPT with SSD.

The results are shown in Figure 7. From this figure, we have three observations. (1) The conversion threshold τ cannot be too large, e.g., 0.98, which will lead to a significant performance drop. The reason may be that a large τ will result in a long continuous period of sparse training, which may lead to the overfitting of the SMoE mode and affect the utilization of model capacity. (2) The ratio of the sparse mode r has little effect on the performance of SSD, which suggests that we can further

increase the ratio to achieve a higher speedup. For example, a sparse ratio of 70% can achieve a FLOPs speedup of $1.63\times$ with a comparable perplexity to dense training. (3) The steps between two activation pattern monitoring cannot be too small, which may lead to frequent switching and unstable training, e.g., 2000 steps. In the future, it is worth exploring the adaptive calculation steps to avoid this issue.

E. Discussion on GLU Models

While our work focuses on ReLU-based FFNs, it is worth discussing the applicability of our method to Gated Linear Units (GLU) (Shazeer, 2020) models, which are widely used in current LLMs (Touvron et al., 2023a). One potential approach would be to modify our clustering algorithm to account for the gating mechanism in GLUs. Instead of simply clustering the activation weight matrix, we could jointly cluster the activation and gate weight matrices, treating them as a unified representation of the feedforward unit's behavior. This would allow our method to identify sparse modes that capture the intrinsic computational patterns emergent from the interplay between activations and gates.

It is also worth noting that while GLUs introduce additional nonlinearities, they still maintain a level of sparsity in their output representations due to the gating mechanism. As shown in Zhang et al. (2024), LLaMA with SwiGLU also exhibits sparse activation phenomenon.