Pruning Language Models into a Mixture of Low-Rank Experts

Roberto Garcia
robgarct@stanford.edu

Daniel Sorvisto
sorvisto@stanford.edu

Andri Vidarsson
andriv@stanford.edu

1 Introduction and Related Work

It is computationally costly to train and do inference on large language models (LLMs). Naively, designing, and building smaller models (in number of parameters) that are faster requires training them from scratch, which is also costly. As a consequence, methods to reduce the size of already trained large models have been proposed by the research community. Examples of this are techniques like quantization [1], knowledge distillation [2] or pruning [3].

Pruning methods attempt to remove redundant or non-significant weights from the model after the training stage, aiming to reduce inference computation cost while preserving most of the capabilities of the model. A seminal paper in neural network pruning is *Optimal Brain Damage* [4], where parameters are pruned by their "importance", i.e. the difference in the loss obtained by setting that weight to 0. Unstructured pruning methods, where individual weights are removed, are able to retain much of the capabilities, but result in sparse matrices that do not achieve great efficiency gains on conventional hardware. Structured methods, where sets of weights such as a rows or columns are removed, result in smaller dense matrices but do not achieve as good results as unstructured methods in LLMs. The first language model pruning methods are very recent; prominent examples include LLM-Pruner [3] and LoRAPrune [5]. LLM-Pruner managed to maintain 94.97% of the model performance while pruning 20% of the parameters, using a structured pruning approach, where the pruned structures are chosen by a similar importance approximation as [4]. LoRAPrune introduces a more efficient method of estimating importance using low-rank adaptation (LoRA) [6].

Mixture of Experts (MoE) [7] is an old but re-emerged ensemble method, that in the context of LLMs, is used to create models that perform inference more efficiently, compared to conventional models of similar size. A sparse MoE model has feedforward layers consisting of multiple blocks (i.e. experts), of which a router selects a subset based on e.g. token, sequence, or sentence. [8] was the first paper to demonstrate the potential of large sparse MoE models, and this approach was first applied to transformers in [9]. Modern MoE LLMs have shown very impressive results, such as Mixtral 8x7B outperforming Llama 2 70B [10]. There is extensive literature on both reducing model size by pruning, and using MoE models to speed up inference [11], but to the best of our knowledge, no one has tried pruning a conventional language model into a more efficient MoE model. Thus our work focuses on taking a conventional pre-trained language model, pruning its weights to reduce computational cost on inference, and using efficient low-rank experts to compensate for the pruned-away weights, selected from an ensemble of experts using a Softmax router module [9].

2 Methods

Our method involves pruning the hidden nodes of the multi-layer perceptron (MLP) layers of our model, and compensating for the pruned-away parameters by both adding a low-rank weight matrix on top of the remaining weights, similar to LLM-Pruner [3], and by approximating the removed nodes with equally sized sparsely activated low-rank experts. We build up this method step-by-step, noting the gained improvements of each step all the way from the baseline. This section presents the methods used in the importance calculation, initial feasibility experiments, experts, and expert routing in more detail.

2.1 Pruning by Importance

The importance of a weight is defined as the increase in the loss function caused by setting that weight to 0 [4]. In practice, this is estimated by a Taylor expansion up to the second degree, as described in equation 1. In our structured pruning approach, we compare the importances of each hidden node in a MLP, which is defined to be the sum of the importances of its inbound and outbound weights, i.e. i:th row of $W_1^{original} \in \mathbb{R}^{h \times d_{model}}$ and i:th column of $W_2^{original} \in \mathbb{R}^{d_{model} \times h}$ corresponding to the inbound and outbound weights of the i:th node, respectively. d_{model} is the dimensionality or the hidden size of the model and h is the number of hidden nodes in each MLP. We could then prune the least important $k\%$ of the hidden nodes, either *uniformly* where every layer gets pruned by $k\%$, or *holistically* where importances of hidden nodes are compared across layers (where the first and last layers get pruned less than the central ones).

$$\text{Importance}(w_i) = |\Delta L(D)| = |L_{w_i}(D) - L_{w_i=0}(D)| \approx |\frac{\partial L(D)}{\partial w_i} w_i - \frac{1}{2} w_i^2 H_{i,i}| \tag{1}$$

2.2 Clustering to Evaluate Feasibility of MoE

Inspired by the observation in Mixtral 8x7B [10] where the same tokens generally get routed through the same expert, we perform clustering on hidden-node importances per token. For every MLP layer in the LLM and for every token, we create a h-dimensional vector consisting of the importance of each hidden node. This gives us a dataset for every MLP layer, consisting of tokens and their corresponding importances for every hidden node in that layer. Finally, for every layer, we compare the R^2 obtained after doing k-means clustering (to cluster tokens) on the original versus a permuted version of the dataset, using $R^2 = 1 - \frac{\text{average distance to closest cluster}}{\text{variance}}$. Notably, this permutation test allows us to compute a p-value that can helps us to initially determine the feasibility of our approach since. If clusters are statistically significant, we believe that that an expert can be learned from each cluster or by other more sophisticated methods.

2.3 Experts

An expert is a submodule whose purpose is to be specialized to specific data or subtasks. The goal of a MoE ensemble model is to learn a diverse set of experts and a *router* module that selects a subset of the experts for each input, thus saving computation on inference while maintaining model performance and expressivity. In our case, we want to compensate the removal of the p least important hidden nodes of a MLP layer with a MoE module in parallel of the remaining original nodes, taking a step further from the simple single-LoRA retraining seen in prevous approaches like LLM-Pruner [3] and LoRAPrune [5]. We propose two expert architectures, *LoRA-on-top* and *LoRA-on-side*. To minimize training and inference computational costs, both methods utilize low-rank matrices $W' = AB^T$ of rank r with thin matrices A and B. An expert is therefore a specific pair of low-rank matrices W_1' and W_2' that get respectively applied to the first and second layers of the MLP using one of two methods.

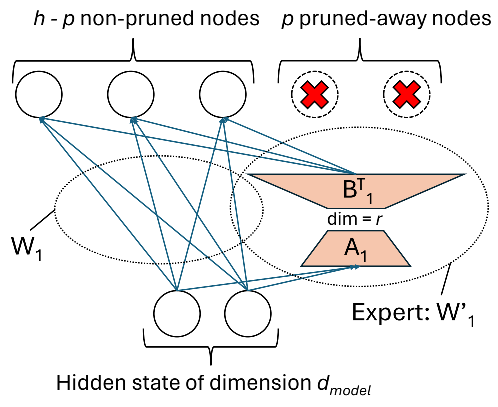
LoRA-on-top experts: The LoRA-on-top architecture consists of the application of classical LoRA where each expert is defined to be a pair of low-rank decompositions that, as shown in figure 1a, get directly added on top of the remaining original weights $W_1 \in \mathbb{R}^{(h-p)\times d_{model}}$ and $W_2 \in \mathbb{R}^{d_{model}\times(h-p)}$. Thus the expert matrices W_i' are of similar size as W_i, and the output of the MLP is therefore $z = (W_2 + W_2')\sigma((W_1 + W_1')x)$, with $\sigma(x)$ representing the activation function of the MLP.

LoRA-on-side experts: The LoRA-on-top architecture is a more novel method where $W_1' \in \mathbb{R}^{p\times d_{model}}$ and $W_2' \in \mathbb{R}^{d_{model}\times p}$ represent a low-rank approximation of the pruned-away nodes. The expert works not on top, but in parallel of the non-pruned weights, as shown in figure 1b, thus the output is $z = W_2\sigma(W_1x) + W_2'\sigma(W_1'x)$. A valuable advantage of this method is that the training of W_1' and W_2' can be initialized by the pruned-away weights, using one of two methods. It is easy to see that in the high-rank limit, LoRA-on-side perfectly represents the original model, hinting that its performance might scale with increased expert rank better than LoRA-on-top.

Importance initialization starts from matrices directly constructed from the pruned-away weights, with everything except the weights connected to the r most important nodes set to 0, that is, $A_1 = B_2 = [\, I_r|0 \,]^T \in \mathbb{R}^{p\times r}$, $B_1 = [w_1^{(h-p+1)T}, \ldots, w_1^{(h-p+r)T}] \in \mathbb{R}^{d_{model}\times r}$, and $A_2 = [w_2^{(h-p+1)}, \ldots, w_2^{(h-p+r)}] \in \mathbb{R}^{d_{model}\times r}$, where $w_1^{(i)}$ and $w_2^{(i)}$ represent the i:th most important row and column in $W_1^{original}$ and $W_2^{original}$, respectively.

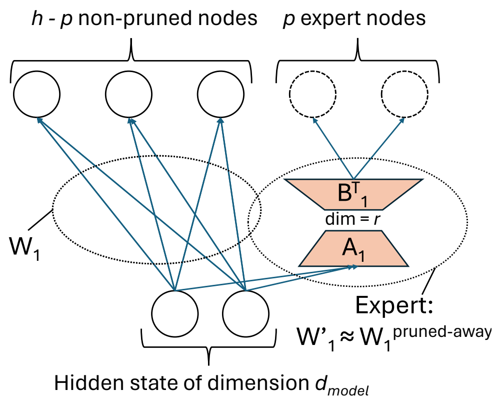
SVD initialization first sets W_1' and W_2' exactly equal to the pruned-away weights and turns them into matrices of rank r by setting their $p - r$ smallest singular values to 0. Matrices A and B are then given directly by the pruned singular value decomposition (SVD).

LoRA-on-top + LoRA-on-side: We further combine LoRA-on-top with LoRA-on-side. In this approach, we retrain the non-pruned weights using LoRA matrices W_1'' and W_2'' on top as well as adding LoRA-on-side experts W_1' and W_2', making the output of the final model $z = (W_2 + W_2'')\sigma((W_1 + W_1'')x) + W_2'\sigma(W_1'x)$, further improving the expert's expressivity with minimal additional cost.



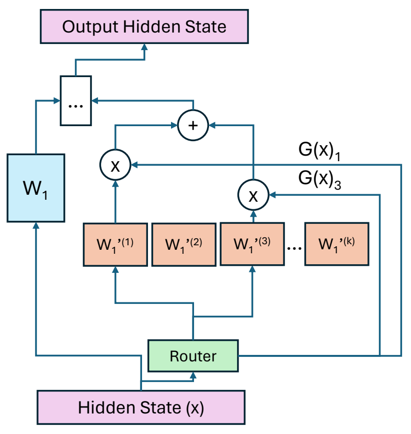
h - *p* non-pruned nodes
p pruned-away nodes
B^T_1
dim = *r*
A_1
W_1
Expert: W'_1
Hidden state of dimension d_{model}

(a) LoRA-on-top architecture



h - *p* non-pruned nodes
p expert nodes
B^T_1
dim = *r*
A_1
W_1
Expert:
$W'_1 \approx W_1^{\text{pruned-away}}$
Hidden state of dimension d_{model}

(b) LoRA-on-side architecture



(c) Routing mechanism

Figure 1: Illustration of the two expert architectures, as well as the expert routing, in the first layer of the MLP.

2.3.1 Expert Routing

The role of the router is to select a subset of the ensemble of k experts at every MLP layer, based on the input embedding, through which such an embedding will be routed as shown in figure 1c.

Cluster Routing: Cluster routing is our basic routing method, which routes embeddings to a single expert based on a clustering prediction for the corresponding input token. In the setup stage, we ran weighted k-means clustering on sampled importance vectors, where the weights are determined by taking the $\log$ of the occurrence frequency of the tokens in the training dataset. This method was chosen to address the skewed distribution of token appearances, with the top 1% of tokens by frequency representing over half of all token instances.

Softmax Routing: Softmax (SM) routing is our most sophisticated routing and it's influenced by [8]. It involves a $k \times d_{model}$ matrix, R, that projects each hidden state into weights for each of the experts. All except the top-n weights are then set to $-\infty$, after which the Softmax function is applied to produce the final routing weights, $G(x) = \text{Softmax}(\text{Top-n}(Rx))$ for the top-n experts. Matrices W_1' and W_2' are thus determined by $W_i' = \sum_{j \in \text{Top-n}} G(x)_j W_i'^{(j)}$, where *Top-n* are the indices for the top-n experts, $G(x)_j$ is the routing weight for the j:th expert, and $W_i'^{(j)}$, $i \in (1, 2)$, is the matrix given by the j:th expert.

For Softmax routing, we also propose an initialization method, which we refer to as *hidden-states centroid initialization*. This method works by setting the initial rows of R as the weighted average of the embeddings at each layer that correspond to the predicted input token clusters obtained for Cluster routing. With this method we try to bias Softmax routing to act similar to Cluster routing initially. We observed that the cosine similarity between hidden-state embedding and the rows of R was generally high for the row belonging to the corresponding cluster, resulting in that the expert corresponding to the cluster received a high weight.

3 Experiments / Results / Discussion

All our experiments are performed on Phi-1.5 [12] and use 50,000 samples from the TinyTextbooks [13] dataset for post training when applicable. Our first experiments statistically tested the feasibility of our approach. Then, we experimented with multiple variations of experts, routers and baselines for both 20% and 50% pruning. We train these variations leveraging the HuggingFace library. Concretely, we used a batch size of 60 samples, each of at most 1024 tokens in length, and trained using the AdamW optimizer with a learning rate of 10^{-4} and default settings. Our initial hyper-parameters were obtained from the LLM-Pruner code and we slightly tweaked them to suit our computational resources and needs. Notably, these settings allowed us to achieve convergence on our training loss in at most 2 epochs using an A5000 Nvidia GPU. We also report the cross-entropy (CE) loss on the TinyTextbooks validation set, consisting of 2,000 samples, and the accuracy on 1000 random samples of HellaSwag [14], PIQA [15], BoolQ [16] and WinoGrande [17]. Our main results can be observed in Tables 2 and 3. Lastly, we provide a small evaluation of the impact of different datasets, where we compare TinyTextbooks to the Alpaca [18] dataset.

3.1 Feasibility of Approach

To evaluate the feasibility of our approach, we select 10,000 tokens and compute the importance of each hidden node in the 1st, 12th, and 24th layer for each token. We estimate the importance of each hidden node following the description in section 2.2.

For every token at each of the three layers, we create a 8192-dimensional vector consisting of the average importances of each hidden node. Then, we perform clustering on the vectors with $k = 8$ clusters. Finally, we statistically test the R^2 values of the clustering on the original versus permuted datasets, as described in 2.2.

On our test on layer 1, we get a p-value of 0.0049, strongly indicating that different clusters of tokens rely much more heavily on different hidden nodes. The existence of this structure indicated to us the feasibility of splitting the MLP of our model into different experts depending on the token. We also highlight the difference between tokens belonging to two different clusters, as shown in Table 1. Here, a clear difference can be seen. The first cluster contains mostly tokens relating to programming, whereas the second cluster holds a more diverse set of tokens.

Table 1: Comparison of example tokens from two different clusters at layer 2

Cluster 1	undefined, sql, localhost, void, interface, gpu, Range, db, suffix, Integer, buffer
Cluster 2	€, Money, takeover, Drakola, Sold, monop, Warning, panic, Ankara, Alabama

3.2 Baselines

To compute both of our baselines, we prune each of the 24 MLP layers by the ratio at hand in Phi 1.5 using uniform pruning. Further, we re-adjust the remaining weights in a post-training phase, where we train them using LoRA on a language modeling objective using a subset of the TinyTextbooks dataset. Concretely, our first baseline is magnitude pruning of the MLPs, where we prune weights in a structured manner based on the magnitude of the weights, similar to that done in [4]. Our second baseline is LLM-Pruner (adjusted to do uniform pruning for a fair comparison) applied to the MLPs . Their method is, as far as we know, the state-of-the-art pruning technique for LLMs.

Model	Router & Initialization	HellaSwag	PIQA	BoolQ	WinoGrande	Average Acc.	TinyTextbooks CE
Not-Pruned	N/A	48.0	77.1	73.6	72.3	67.8	3.170
Magnitude-Pruned + LoRA post-training	N/A	35.8	69.7	56.8	59.5	55.5	3.670
LLM-Pruner	N/A	36.4	72.6	64.6	57.5	57.7	3.384
MoE LoRA-on-top (rank 64)	Cluster Router	36.1	72.9	64.8	59.5	58.3	3.366
	SM Router	37.7	72.9	64.1	59.5	58.6	3.376
	SM Router (initialized)	37.4	72.7	64.3	57.8	58.1	3.340
MoE LoRA-on-side (rank 64)	Cluster Router	36.0	71.7	63.7	57.2	57.2	3.600
MoE LoRA-on-side + LoRA-on-top (rank 256)	Cluster Router (importance init.)	38.9	73.8	63.7	60.2	59.2	3.230
	Cluster Router (SVD init.)	**40.2**	**74.8**	**67.2**	**65.1**	**61.8**	**3.150**

Table 2: Performance of baselines and out approach on Phi-1.5 when pruning to 50%. Best of all is in bold. Best of all, not considering LoRA-on-side + LoRA-on-top experts, is underlined.

Model	Router & Initialization	HellaSwag	PIQA	BoolQ	WinoGrande	Average Acc.	TinyTextbooks CE
Magnitude-Pruned + LoRA post-training	N/A	42.5	74.8	**66.4**	66.0	62.4	3.190
LLM-Pruner	N/A	42.1	75.8	65.0	68.6	62.9	3.10
MoE LoRA-on-top (rank 64)	Cluster Router	42.5	76.3	64.3	68.9	63.0	3.092
	SM Router	42.0	75.6	65.2	69.6	63.1	3.095
	SM Router (initialized)	42.0	76.4	64.9	68.4	62.9	3.085
MoE LoRA-on-side (rank 64)	Cluster Router	42.4	**76.6**	63.7	69.3	63.0	3.181
MoE LoRA-on-side + LoRA-on-top (rank 128)	Cluster Router (importance init.)	**43.6**	76.5	64.5	**70.4**	**63.8**	3.055
	Cluster Router (SVD init.)	43.4	75.7	63.3	68.7	62.8	**3.048**

Table 3: Performance of baselines and out approach on Phi-1.5 when pruning to 20%. Best of all is in bold.

3.3 Experts and Routers

We experiment with 3 different types of experts described in Section 2.3, namely: LoRA-on-top, LoRA-on-side and LoRA-on-top combined with LoRA-on-side. In addition, we try out the importance initialization and the SVD initialization for the LoRA-on-top. Results can be seen in Tables 2 and 3. Further, we experiment with 2 different types of routing described in Section 2.3.1, Cluster routing and Softmax routing. We also try out the hidden-states centroids initialization approach for the Softmax routing. Results can be seen in Tables 2 and 3.

We mainly discuss and refer to results obtained when pruning Phi-1.5 MLPs by 50% unless stated otherwise, as this is the most challenging task with the largest area of improvement in the LLM pruning literature. We note that all variations of our MoE approach have a higher average accuracy on the metrics than the magnitude pruning baseline. Further, most of them are better than the LLM-Pruner baseline, as observed in Table 2.

LoRA-on-top experts: We highlight the MoE with initialized Softmax routing. Such an approach performs on par with the LLM-Pruner baseline on HellaSwag, PIQA and BoolQ, however it has better accuracy at WinoGrande (59.5% vs 57.2%) and cross-entropy at TinyTextbooks (3.366 vs 3.384), as seen in Table 2. We observe similar results when pruning to 20%, ass seen in Table 3. We believe the reason other metrics perform on par, but this method improves on both cross-entropy and WinoGrande is because the TinyTextbooks dataset is more aligned with the WinoGrande task than the other metrics. WinoGrande seeks to test reasoning through sentence completion as opposed to the other metrics like BoolQ, which seek to test general knowledge.

LoRA-on-side experts: Results for LoRA-on-side by itself were not the expected. Notably, this method originally performed slightly worse on the overall accuracy than the LLM-Pruner baseline (57.2% vs 57.7%), as seen in 2. We believe this occurred since, while LoRA-on-side is capable of recovering the full network in the limit of full rank, the remaining weights on the pruned network are not the optimal for the new architecture. This suggested to us that using LoRA-on-top in conjunction with LoRA-on-side would constitute the most expressive low-rank expert.

LoRA-on-side + LoRA-on-top + SVD initialization: We found that combining LoRA-on-side with LoRA-on-top, in conjunction with SVD importance intialization, yielded the best results, as highlighted in Tables 2, 3. Notably, we underscore the results obtained when using the SVD initialization for 50% pruning. Such a method obtains both a better average accuracy (61.8% vs 57.7%) than LLM-Pruner, as well as a lower cross-entropy loss on TinyTextbooks (3.150 vs 3.384). In addition it also outperforms the other variants we propose on all metrics. We attribute the success of this approach to 2 phenomena. First, we believe that training the original weights of the pruned model leveraging LoRA-on-top allows them to adjust to the new architecture. Second, we believe initialization is a key component of improvement as the SVD initialization recovers the most important information of the weight matrix in a low-rank form.

Pruning to 20%: Results when pruning to 20% are illustrated in Table 3. We believe these results have a lower signal/noise ratio than those obtained when pruning to 50%. Generally speaking, we observed similar results to those obtained when using 50% pruning, but usually the observed result had a lower impact. We mainly attribute this to the fact that 20% pruning did not harm the model as significantly as 50% pruning did. This left a lower room for improvement during post-training.

Routing: As we can observe from Tables 2 and 3, using Softmax routing performs similarly to cluster routing. However, we observe that initializing the Softmax router using the hidden-states centroids produces a lower cross-entropy loss.

Data: We noticed that the distribution of the data we post-train our model on impacts the knowledge the pruned network recovers. First, we were hinted this by observing that our models had a much bigger impact on WinoGrande than the other datasets as seen in Tables 2 and 3. Upon inspection, we observed that the WinoGrande dataset is more similar to the TinyTextbooks dataset than the others. To confirm this, we tested the differences in performance when using the Alpaca dataset and the TinyTextbooks dataset with the LLM-Pruner baseline (which does a simple LoRA post-training on the pruned network). There, we observed that Alpaca outperformed TinyTextbooks in HellaSwag and performed on par on the other metrics, as seen in Table 4. This hints us that it's possible to tailor a better dataset encompassing multiple tasks that is capable of better recovering the knowledge of the pruned LLMs across a varied set of tasks.

Dataset	HellaSwag	PIQA	BoolQ	WinoGrande
TinyTextbooks	38.1	71.3	63.5	59.1
Alpaca	<u>39.4</u>	71.6	63.3	59.3

Table 4: Performance of the adjusted LLM-Pruner on Phi-1.5 when pruning to 50% using Alpaca and TinyTextbooks for post-training. Note that the evaluation samples used here were a different shuffle than the ones used in previous results.

4 Conclusion / Future Work

Our experiments demonstrate that retraining a pruned model with a set of low-rank experts is a feasible method that, when tuned properly, can outperform standard low rank retraining methods with limited extra cost during inference. We observed that, even with a simple deterministic routing method, our model is able to outperform models that leverage a single LoRA adapter at each layer. Our proposed expert architecture, LoRA-on-the side in conjunction with LoRA-on-top and SVD initialization, shows great promise, as experts retain greater information and the pruned layers are able to adapt to the pruned architecture. We also note that post-training datasets have a non-trivial impact on the re-learned capabilities of the pruned model.

Future work lies in applying more sophisticated and novel pruning procedures, routers, experts and expert initializations as well as curating more effective post-training datasets. In addition, we seek to extend our method to prune other layers in the LLM besides the MLPs.

5 Contributions

Roberto mainly worked on the experiments and their code, as well as the code for the experts. Daniel mainly worked on the pruning methods and SVD experiments. Andri mainly worked on the router code, experiments, and baseline pruners.

References

[1] H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King, “Binarybert: Pushing the limit of bert quantization,” 2021.

[2] H. Pan, C. Wang, M. Qiu, Y. Zhang, Y. Li, and J. Huang, “Meta-kd: A meta knowledge distillation framework for language model compression across domains,” 2022.

[3] X. Ma, G. Fang, and X. Wang, “Llm-pruner: On the structural pruning of large language models,” 2023.

[4] Y. LeCun, J. Denker, and S. Solla, “Optimal brain damage,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[5] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, and B. Zhuang, “Loraprune: Pruning meets low-rank parameter-efficient fine-tuning,” 2023.

[6] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” 2021.

[7] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural Computation*, vol. 3, no. 1, pp. 79–87, 1991.

[8] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, “Outrageously large neural networks: The sparsely-gated mixture-of-experts layer,” 2017.

[9] W. Fedus, B. Zoph, and N. Shazeer, “Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity,” 2022.

[10] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, “Mixtral of experts,” 2024.

[11] W. Fedus, J. Dean, and B. Zoph, “A review of sparse expert models in deep learning,” 2022.

[12] Y. Li, S. Bubeck, R. Eldan, A. D. Giorno, S. Gunasekar, and Y. T. Lee, “Textbooks are all you need ii: phi-1.5 technical report,” 2023.

[13] Nam Pham, “tiny-textbooks (revision 14de7ba),” 2023. [Online]. Available: https://huggingface.co/datasets/nampdn-ai/tiny-textbooks

[14] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, “Hellaswag: Can a machine really finish your sentence?” *CoRR*, vol. abs/1905.07830, 2019. [Online]. Available: http://arxiv.org/abs/1905.07830

[15] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, “PIQA: reasoning about physical commonsense in natural language,” *CoRR*, vol. abs/1911.11641, 2019. [Online]. Available: http://arxiv.org/abs/1911.11641

[16] C. Clark, K. Lee, M. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, “Boolq: Exploring the surprising difficulty of natural yes/no questions,” *CoRR*, vol. abs/1905.10044, 2019. [Online]. Available: http://arxiv.org/abs/1905.10044

[17] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, “WINOGRANDE: an adversarial winograd schema challenge at scale,” *CoRR*, vol. abs/1907.10641, 2019. [Online]. Available: http://arxiv.org/abs/1907.10641

[18] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, “Stanford alpaca: An instruction-following llama model,” https://github.com/tatsu-lab/stanford_alpaca, 2023.