Structured Pruning for Large Language Models Using Coupled Components Elimination and Minor Fine-tuning

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated powerful capabilities in natural language processing, yet their vast number of parameters poses challenges for deployment and inference efficiency. Structured model pruning emerges as a viable approach to reduce model size and accelerate inference, without requiring specialized operators and libraries for deployment. However, structured pruning often severely weakens the model's capability. Despite repetitive fine-tuning can restore the capability to a certain extent, it impairs LLMs' utility as versatile problem solvers. To address this issue, we propose a novel structured pruning algorithm tailored for LLMs. It derives the importance of different components, namely rows and columns in parameter matrices, based on intermediate data dependencies. Then it removes coupled components across different layers simultaneously and preserves dependency relationships within remaining parameters, avoiding significant performance degradation. The pruned model requires only few epochs of fine-tuning to restore its performance, ensuring the model's ability to generalize. Empirical evaluations on LLaMA, Vicuna, and ChatGLM3 demonstrate our algorithm's efficacy, yielding 20% parameter reduction while retaining at least 94.4% of original performance metrics.

1 Introduction

Large language models (LLMs) have demonstrated powerful capabilities in solving a variety of general problems (OpenAI, 2023; Xue et al., 2020), particularly in language understanding and generating. However, the large number of parameters (Radford et al., 2018, 2019; Brown et al., 2020) in LLMs poses significant challenges for deployment and inference efficiency. Structured pruning (Wang et al., 2019; Xia et al., 2022; Zafrir et al., 2021) has been proved to be a viable approach to compress deep neural networks. It removes entire structural components of the neural network, without requiring specialized operators and libraries for executing the pruned model, so that it is convenient for deployment and acceleration.

Despite structured pruning algorithms have long been investigated (Lagunas et al., 2021; He et al., 2020; Kurtic et al., 2022), they face new challenges when tackling LLMs. Existing state-of-the-art pruning algorithms follow an iterative scheme (Han et al., 2015a; Louizos et al., 2017; Xia et al., 2022; Zafrir et al., 2021) for specific tasks. This scheme conducts iterative **evaluating, pruning and fine-tuning** on a large model for a single task, achieving low performance degradation. However, due to the repetitive fine-tuning on a single task, the pruned model has much less generalization ability on other tasks. This is a particularly serious issue for LLMs, since they are expected to be general-purpose models solving extensive problems. Simply extending the fine-tuning on more corpus and tasks to reserve the generalization ability is still challenging (Ma et al., 2023), because LLMs require huge volume of training corpus.

In this study, we propose a novel structured pruning algorithm tailored for LLMs. In contrast to existing iterative pruning works, our algorithm first conducts iterative **evaluating and pruning**, until the desired sparsity level is achieved. After completing all the iterations of evaluating and pruning, it then conducts one stage of **fine-tuning**, which involves few epochs of training on a small dataset. The intuition of our algorithm is to limit the fine-tuning operations as few as possible, so that the pruned model will not import too much bias towards specific tasks.

To ensure that the remaining parameters are consistently important and do not need repetitive fine-tuning to restore performance, we need to precisely evaluate the importance of structured components, namely rows and columns in parameter matrices. More concretely, our algorithm derives the im-

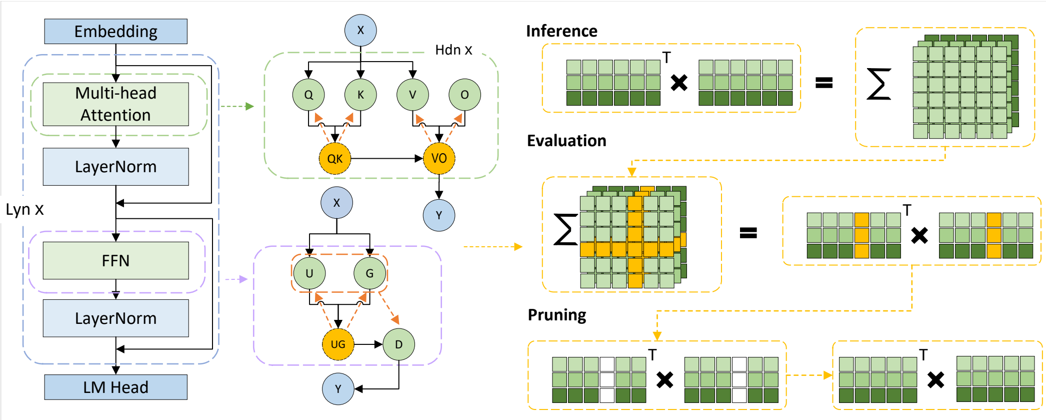

Figure 1: During the pruning process, we determine whether a component should be pruned according to the inference error caused by removing the component and its coupled components from intermediate results.

portance and uncertainty of different components based on intermediate data dependencies, as shown in Figure (1). According to the Transformer-based model architecture, we can identify the coupled components that have data dependency on pruned components. These coupled components across different layers can be removed simultaneously, and the dependency relationships within remaining parameters can be still preserved, avoiding significant performance degradation. Moreover, we employ LoRA (Hu et al., 2022) fine-tuning to restore model performance, and use LoRA gradients (Zhang et al., 2023) instead of full-scale fine-tuning gradients to reduce the computational overhead during pruning. The model pruned by our algorithm preserves the original architecture with smaller parameter matrices, thus it is compatible to any other Transformer-specific optimization techniques, e.g, FlashAttention (Dao et al., 2022; Dao, 2023). We have validated our algorithm on LLaMA (Touvron et al., 2023), Vicuna (Chiang et al., 2023), and ChatGLM3 (Zeng et al., 2022; Du et al., 2022), achieving about 20% parameter reduction while retaining at least 94.4% of original performance metrics.

Contribution. In this paper, (i) we proposes a new structured pruning algorithm for LLMs that uses minimal fine-tuning to recover model performance. The algorithm effectively reduces the number of parameters while maintaining model generalization. (ii) We propose a novel evaluation method that evaluates the impact of structured pruning on an LLM by evaluating coupled components instead of individual weights. (iii) We conduct our algorithm on representative LLMs, including LLaMA, Vicuna, and ChatGLM3. By reducing the parameter count by 20%, we maintain at least 94.4% of the model's performance while reducing MACs by 20%.

2 Related Work

2.1 Iterative Pruning

Iterative pruning is a type of algorithm that iteratively evaluates, prunes, and fine-tunes a neural network model. The process involves calculating scores for each weight in the model based on specific criteria, pruning weights with lower scores, and fine-tuning the pruned model on a dataset. PLATON (Zhang et al., 2022a) is a typical iterative pruning method for (Devlin et al., 2019) and ViT (Dosovitskiy et al., 2020). It considers the sensitivity and uncertainty of different model components during evaluation, improving the accuracy of the evaluation process. Although iterative pruning has been proved to be effective for task-specific models, it faces difficulty for general-purpose LLMs due to the repeated fine-tuning.

2.2 LoRA

LoRA is an efficient fine-tuning algorithm for LLMs. Due to the large size of the parameter matrices in LLMs, the computational cost of full fine-tuning is often prohibitively high. In LoRA fine-tuning, a data bypass is created for the target parameter W_0: $W = W_0 + BA$, where $W_0 \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{n \times r}$, $A \in \mathbb{R}^{r \times m}$, and $r \ll \min(n, m)$. Typically, the parameters in A are initialized with a random Gaussian distribution, and the parameters in B are set to 0. During the subsequent fine-tuning process, the parameters in W_0 are frozen, and only the parameters in A and B are fine-tuned. LLM-Prunner (Ma et al., 2023) is a structured pruning algorithm for LLMs. It combines efficient LoRA fine-tuning to recover the performance of the pruned model with fewer fine-tuning epochs. LoRAPrune (Zhang et al., 2023) is a non-structured pruning algorithm for LLMs. Due to the high cost of obtaining gradients in LLM, LoRAPrune leverages LoRA gradients instead of full fine-tuning gradients to reduce computational overhead.

3 Method

Our pruning consists of three steps. (i) Partitioning the model into kernels and features, and grouping the coupled components formed by kernels. (ii) Iteratively evaluating and pruning coupled components and features until the desired sparsity level is achieved. (iii) After all evaluating and pruning finish, a fine-tuning stage is conducted to restore the model performance.

3.1 Partition of Kernels and Features

In our algorithm, the pruning granularity is rows or columns in the parameter matrices. The functionality of a rows or a column varies in different parameter matrices. For example, in the Transformer architecture, each word in a sentence is transformed into a word vector with d_m features, the parameter matrix $V \in \mathbb{R}^{d_m \times d_k}$ of the Transformesr, each row encounters all the weights in the word vectors during computation. However, each column encounters only one weight in the word vector (Fang et al., 2023). Therefore, we divide them into kernels and features based on their functionalities in the inference computation. If a row (or column) receives all the features of the word vector, we refer to that row (or column) as a **kernel**. For example, each row in the $Q \in \mathbb{R}^{d_k \times d_m}$ of a single head, as well as each column in $O \in \mathbb{R}^{d_m \times d_k}$. If a row (or column) receives a specific feature of the word vector, we refer to it as a **feature**. For example, each row in O, or each column in $Up \in \mathbb{R}^{im \times d_m}$ in LLaMA's intermediate layers.

3.2 Evaluation of Importance

Evaluating coupled components. In the multi-head attention mechanism of Transformer, the computation of a single head can be represented by the following equation Eq. (1):

$$Attn = \text{Softmax}\left(\frac{X^t Q^t K X}{\sqrt{d_k}}\right) X^t V^t O^t, \qquad (1)$$

where $Q, K, V \in \mathbb{R}^{d_k \times d_m}$ represent the Query, Key, and Value of a single head in the multi-head attention mechanism, respectively, and $O \in \mathbb{R}^{d_m \times d_k}$ represents the projection matrix used to receive the output of that attention head. $X \in \mathbb{R}^{d_m \times \text{len}}$ represents the sequence of word vectors, where len is the length of the vector sequence. We can observe that Q and K are coupled together, and V and O are coupled together in the equation. The effective parameters in the multi-head attention mechanism are $Q^t K$ and $V^t O^t$. Hence, when evaluating the coupled components of the self-attention layer, we group Q, K for evaluation, and V, O for another evaluation. For the evaluation of coupled components, we take Q and K as an example. We consider Q and K as a sum of multiple kernels, i.e., $Q = [q_1^t, q_2^t, ..., q_{d_k}^t]^t$, $K = [k_1^t, k_2^t, ..., k_{d_k}^t]^t$, where $Q, K \in \mathbb{R}^{d_k \times d_m}$, and $q_i, k_i (i \in [1, d_k])$ are row vectors of dimension d_m. In this case, we expand $Q^t K$ in Eq.(2):

$$Q^t K = \sum_{i=1}^{d_k} q_i^t k_i. \qquad (2)$$

If we prune one q_i, we can observe that the corresponding k_i will no longer be effective in the inference process and should be pruned simultaneously. We have found the coupled component $q_i^t k_i$ generated by Q and K. The same applies to the grouping of $V^t O^t$, where the coupled components become $v_i^t o_i^t$. In the intermediate layers of the model, we can also find a similar relationship. In previous models such as BERT (Devlin et al., 2019), GPT-Neo (Black et al., 2022) and OPT (Zhang et al., 2022b), a two-layer structure was commonly used, which can be represented by the equation Eq.(3):

$$Out = fc_2 F(fc_1 X). \qquad (3)$$

Here, $fc_1 \in \mathbb{R}^{im \times d_m}$ and $fc_2 \in \mathbb{R}^{d_m \times im}$. F represents the activation function. The partitioning method at this stage is the same as the partitioning for $Q^t K$. In the LLaMA and ChatGLM3, a three-layer structure was used in the intermediate layers, which can be represented by the equation Eq.(4):

$$Out = Down(F(GateX) \odot UpX). \qquad (4)$$

Here, $Gate, Up \in \mathbb{R}^{im \times d_m}$, and $Down \in \mathbb{R}^{d_m \times im}$. In the LLaMA model, we cannot directly partition the kernels in the three parameter matrices through computation. However, we can observe that when any kernel in any of these three matrices is zero, the corresponding kernels in the remaining two matrices will no longer be effective. Therefore, we approximate the coupled component (d_i, g_i, u_i) as two sub-components: $d_i g_i^t$

and $d_i u_i^t$, where d_i, g_i, u_i correspond to the kernels in $Down, Gate, Up$, respectively. During the scoring process, we use the sum of scores of the subcomponents $d_i g_i^t$ and $d_i u_i^t$ to represent the score of the coupled component (d_i, g_i, u_i).

After grouping the kernels, these coupled components can be represented as the multiplication of a column vector α and a row vector β. We denote such coupled components as $C = \alpha\beta$, where $C \in \mathbb{R}^{d_m \times d_m}$. During the evaluation process, we evaluate the importance of the coupled component C by measuring the error in neural network prediction when removing this group of coupled components. This is defined as the importance I_C (Ma et al., 2023) and can be calculated as Eq.(5):

$$\begin{aligned} I_C &= \left| \sum_{c \in C} \mathcal{L}(c) - \mathcal{L}(c=0) \right| \\ &= \left| \sum_{c \in C} \frac{\partial \mathcal{L}}{\partial c} c - \frac{1}{2} \left(\frac{\partial^2 \mathcal{L}}{\partial c^2} c^2 \right) + \mathcal{O}(c^3) \right|. \end{aligned} \tag{5}$$

For the second-order error term $\left(\frac{\partial^2 \mathcal{L}}{\partial c^2} c^2\right)$, we approximate it as $\left(\frac{\partial \mathcal{L}}{\partial c} c\right)^2$ based on (Ma et al., 2023; Yang et al., 2023). Therefore, we have Eq.(6):

$$I_C \approx \left| \sum_{c \in C} \frac{\partial \mathcal{L}}{\partial c} c - \frac{1}{2} \left(\frac{\partial \mathcal{L}}{\partial c} c \right)^2 \right|. \tag{6}$$

Additionally, we refer to the evaluation method proposed by PLATON (Zhang et al., 2022a), which combines the sensitivity of the network to determine the final score for the coupled components. The scoring process is as Eq.(7):

$$\begin{aligned} \bar{I}_C^{(t)} &= x_1 \bar{I}_C^{(t-1)} + (1 - x_1) I_C^{(t)}, \\ U_C^{(t)} &= |I_C^{(t)} - \bar{I}_C^{(t)}|, \\ \bar{U}_C^{(t)} &= x_2 \bar{U}_C^{(t-1)} + (1 - x_2) U_C^{(t)}, \\ S_C &= \sum_t \bar{I}_C^{(t)} \bar{U}_C^{(t)}. \end{aligned} \tag{7}$$

Here, t represents the current iteration of evaluation for the variable. $\bar{I}_C$ represents the smoothed treatment of importance changes during fine-tuning (Molchanov et al., 2019; Liang et al., 2021) . U_C represents the uncertainty of current importance for the coupled component (Zhang et al., 2022a). $\bar{U}_C$ represents the upper bound confidence for $\bar{I}_C$ (Zhang et al., 2022a). Finally, S_C is the final score for the coupled component. The hyperparameters x_1 and x_2 are chosen as 0.5 in our experiments.

Evaluating Features. According to the description in the (Fang et al., 2023), in structured pruning, if we want to prune a feature at a specific position, we need to prune the corresponding features at that position in all parameter matrices of the model. Therefore, we only need to group all corresponding features at the same position in the model. When we remove a feature from the model, the resulting error can be approximated as Eq.(8):

$$I_f \approx \sum_C \left| \sum_{c \in C[:,f] \cup C[f,:]} \frac{\partial \mathcal{L}}{\partial c} c - \frac{1}{2} \left(\frac{\partial \mathcal{L}}{\partial c} c \right)^2 \right|. \tag{8}$$

Here, C refers to the $Q^t K$ and $V^t O^t$ for each attention head in each layer. Taking the grouping of $Q^t K$ as an example, we consider Q and K in the multi-head attention mechanism as the superposition of multiple features, i.e., $Q = [q_1, q_2, ..., q_{d_m}]$ and $K = [k_1, k_2, ..., k_{d_m}]$, where q_i and k_i are column vectors of dimension d_k. If we set all the values at position j to zero, it is equivalent to setting all the values in the j-th row and j-th column of the matrix $Q^t K$ to zero.

In the evaluation of features, we do not consider the impact of intermediate layers. The importance of features is mainly determined by the self-attention process of the model, while the role of intermediate layers is to superimpose multiple self-attention processes (de Wynter and Perry, 2020). In our experiments with BERT and ViT (Dosovitskiy et al., 2020), we find that evaluating features using only self-attention layers already achieves good results. Additionally, because the partitioning of intermediate layers in LLaMA does not strictly consider the computation process, it may also affect the accuracy of the evaluation.

We also incorporate the scoring process from the PLATON algorithm into the feature evaluation, as shown in Equation Eq.(7). In this case, the coupled components C are replaced by features f.

3.3 Pruning

In pruning self-attention layers, we adopt a simple uniform strategy to remove unimportant components. Our pruning strategy for self-attention layers is to remove the lowest-scoring self-attention head for each self-attention layer in each iteration. The score of a self-attention head is the sum of the scores of its constituent Q, K, V, and O kernels.

For the pruning of intermediate layers, we also adopt a uniform pruning strategy. In each iteration, a fixed number of kernels are pruned for all parameter matrices in these layers. We have observed that for most Transformer models, there is a constant

Remaining Ratio	tune	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio=100%	-	LLaMA-7B	12.62	22.14	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
Ratio=20%	w/o	LP-Channel	74.63	153.75	62.75	62.73	41.40	51.07	41.38	27.90	30.40	45.38
		LP-Block	19.24	34.09	62.54	75.41	65.99	60.30	61.57	36.69	39.20	57.39
		Ours	37.90	74.30	66.57	73.39	62.11	62.90	58.24	35.75	36.20	56.45
Ratio=20%	w/	LP-Channel	22.02	38.67	59.08	73.39	64.02	60.54	57.95	35.58	38.40	55.57
		LP-Block	17.39	30.20	66.79	77.58	68.48	64.96	64.06	37.88	39.00	59.82
		Ours	22.00	42.58	72.26	75.13	68.87	66.53	63.29	38.73	41.40	60.88
Ratio=24%	w/o	Ours	34.55	72.14	63.36	69.96	55.92	60.37	53.19	33.70	35.40	53.12
Ratio=24%	w/	Ours	25.01	46.79	68.47	73.88	65.88	63.53	59.63	35.58	38.00	57.85

Table 1: LLaMA pruning experiments. The evaluation metric for WikiText2 and PTB tests is perplexity, which is the smaller the better. The evaluation metric for other tasks is accuracy, which is higher the better. In the experiments, "w/o" indicates that the model did not undergo fine-tuning after the pruning process, and "w/" indicates that the model underwent fine-tuning after the pruning process.

ratio between the number of kernels im in each intermediate layer and the number of $head_{num} \times d_k$ in the self-attention layers (de Wynter and Perry, 2020). For example, this ratio is 4 for OPT models (Zhang et al., 2022b) and around 2.7 for LLaMA models. Therefore, in each iteration, we prune $r \times d_k$ kernels for each parameter matrix in the intermediate layers, where $r = im/(head_{num} \times d_k)$.

For features, we need to remove the features in the same positions of all parameter matrices of the model (Fang et al., 2023). We only need to score all features in each iteration and remove the lowest-scoring features. Since most parameter matrices in the self-attention layers of Transformer models are square matrices, for simplicity, we prune d_k features in each pruning operation, which ensures that the parameter matrices in the pruned self-attention layers are still square matrices.

Algorithm 1 LLMs Structure Pruning

Input: pre-trained model, number of iterations
Output: pruned model

def EvalandPruning (PreTrainModel)
  Partition and Eval kernels and features
  for i **in** $[0 : LayerNum)$
    Remove the head with the lowest score
    Remove the $r \times d_k$ kernels in FFN
  end # end for
  Remove d_k features in every weight matrix
  Change the model size
return PrunedModel # end def

Main( )
  $model \leftarrow initial\ model$
  for i **in** $[0 : iterations)$
    $model$:= EvalandPruning($model$)
  end # end for
  $FinalModel$:= Finetune($model$)
return $FinalModel$ # end Main

3.4 Overall Process

This section summaries the overall process of our pruning algorithm, as shown in Alg.(1). It begins by partitioning the parameters using the approach outlined in section 3.1. Subsequently, we employ an iterative evaluation and pruning strategy, where the parameters are evaluated using the methods described in section 3.2, and the model is pruned using the approach detailed in section 3.3. Once the evaluation and pruning process is completed, we proceed with fine-tuning to restore the model's performance.

4 Experiments

4.1 LLaMA and Vicuna Pruning Experiments

We conduct experiments on the LLaMA-7B and Vicuna-7B which have identical architectures. We test the performance of these models at sparsity levels of 20% and 24%. The evaluation tasks we used are WikiText2 (Merity et al., 2016), PTB (Marcus et al., 1993), BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-e, ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). The evaluation metrics for WikiText2 and PTB tests are perplexity, which is the smaller the better. The evaluation metric (Gao et al., 2023) for other tasks is accuracy, which is higher the better. We compare the results with the structurally pruned LLM-Pruner. The experimental results are shown in Tables 1 and 2. All experiments are conducted on two Nvidia A100 GPUs.

Experimental Details. In every evaluation iteration of LLaMA and Vicuna, we randomly take 10 sentences of length 64 from the C4 (Dodge et al., 2021) dataset to obtain gradient and magnitude information. Our algorithm uses LoRA gradients instead of actual gradients. Since the parameters in the LoRA matrix are randomly initialized, we first train the LoRA parameter matrix for 5 iterations with the 10 sentences after concatenating the LoRA parameter matrices. After the pre-processing of the LoRA parameter matrix, we collect the gradient and magnitude information generated by inputting these 10 sentences into the model for evaluation.

In every prunning iteration, one self-attention head is pruned for all self-attention layers, and 320 kernels were removed for gate-proj, up-proj, and down-proj in each layer. Additionally, 128 features (model's $d_k = 128$) were removed from all parameter matrices.

To obtain the models with sparsity levels of 20%, we initially performed 3 iterations of evaluation and pruning. After the completion of the third iteration of evaluation-pruning, we obtained the 20% sparse model without fine-tuning. We can further increase the sparsity to 24% in the same way, just by changing the number of evaluation-pruning iterations from 3 to 4. Then we fine-tune this model for 4 epochs on the Alpaca (Taori et al., 2023) to restore its performance.

Experimental Analysis. In the LLaMA pruning experiments, we observe that our pruning algorithm performs well even at lower sparsity levels, even without fine-tuning. At sparsity levels of 20% and 24%, our algorithm surpasses LLM-Pruner's Channel mode at 20% sparsity. After pruning and fine-tuning, our algorithm achieves slightly higher perplexity in the WikiText2 and PTB tasks at a 20% sparsity level. Our algorithm outperforms LLM-Pruner's Channel and Block modes in average scores from BoolQ to OBQA, reaching 96% of the performance of the unpruned network. At a sparsity level of 24%, our algorithm, after fine-tuning, outperforms LLM-Pruner's Channel mode at 20% sparsity in average scores from BoolQ to OBQA, with an average score of 91% compared to the unpruned network.

In the Vicuna pruning experiments, our algorithm exhibits similar performance. At a sparsity level of 20%, our algorithm's perplexity performance in WikiText2 and PTB is comparable to LLM-Pruner's Block mode. Our algorithm outperforms LLM-Pruner's Block mode in average scores from BoolQ to OBQA, reaching 94% of the performance of the unpruned network. Additionally, at a sparsity level of 24%, our pruned network, after fine-tuning, shows no significant difference compared to LLM-Pruner's Block mode 20% sparsity model. The average score from BoolQ to OBQA only decreases by 0.17 points compared to LLM-Pruner, while achieving the performance of the original unpruned network 92%.

The inference performance and storage overhead of our pruned models are presented in Table 3. The evaluation is conducted following the methodology described in the (Ma et al., 2023). At sparsity levels of 20%, although our algorithm retains more remaining parameters, it doesn't exhibit a significant difference in memory consumption compared to LLM-Pruner. Our computational complexity falls between LLM-Pruner's Channel mode and Block mode. Therefore, our algorithm theoretically offers better acceleration performance than LLM-Pruner's Block mode.

4.2 ChatGLM3 Pruning Experiment

We conduct experiments on the ChatGLM3. We test the model on the datasets same to LLaMA and Vicuna to evaluate its performance at sparsity levels of 10% and 20%. We compare our pruning algorithm with random pruning and L2 (Han et al., 2015b; Li et al., 2016) weight pruning. All experiments are conducted on two Nvidia A100 GPUs.

Experimental Details. Differing from many Transformer-based models, like LlaMA, BERT, ViT, etc., ChatGLM3 has a unique structure in its self-attention layers. In ChatGLM3-6B, there are 32 Query heads and only 2 Key and Value heads in the multi-head self-attention mechanism. During inference, the model replicates the Key and Value heads 16 times to match the number of Query heads, and the subsequent computation follows the same process as other Transformer models. We make appropriate adjustments to our pruning algorithm to accommodate ChatGLM3's computation approach.

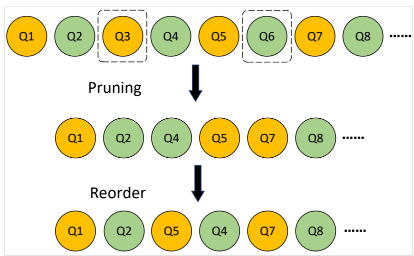

Figure 2: We reorder the remaining pruned Query heads. The processing of parameter matrix O follows the same approach.

We observe that in ChatGLM3, odd-numbered Query heads correspond to odd-numbered Key and Value heads, and the same applies to even-numbered heads. Therefore, our previous pruning strategy becomes removing the Query head with

Remaining Ratio	tune	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio=100%	-	Vicuna-7B	16.11	61.37	76.57	77.75	70.64	67.40	65.11	41.21	40.80	62.78
Ratio=20%	w/o	LP-Channel	71.75	198.88	51.77	63.93	42.558	55.17	43.94	29.27	33.40	45.72
		LP-Block	26.51	90.87	62.97	74.76	63.40	55.88	64.23	38.14	36.60	58.57
		Ours	28.50	92.56	69.69	73.77	58.72	61.79	62.92	35.06	35.40	56.76
Ratio=20%	w/	LP-Block	19.47	76.55	66.45	75.84	65.05	60.38	62.37	36.43	39.80	58.05
		Ours	22.89	73.23	70.73	74.48	66.29	63.22	65.19	36.00	38.80	59.24
Ratio=24%	w/o	Ours	34.30	113.18	67.43	70.56	53.34	58.87	58.37	31.99	34.00	53.50
Ratio=24%	w/	Ours	26.20	84.12	69.11	73.23	63.52	63.69	63.08	34.98	37.60	57.88

Table 2: The Vicuna pruning experiments.

Method	Ratio	#Params	#MACs	Memory
-	-	6.7B	424.0G	12884.5MiB
LP-Channel	20%	5.4B	323.7G	10488.4MiB
LP-Block		5.4B	367.5G	10375.5MiB
Ours		5.5B	351.7G	10687.2MiB
Ours	24%	5.2B	328.7G	9998.0MiB

Table 3: Statistic for LLaMA and Vicuna.

the lowest score among all odd-numbered heads, the Query head with the lowest score among all even-numbered heads, and their corresponding parameter matrix O. The Key and Value heads remain unchanged. After pruning, as the order of Query heads may change from odd to even or vice versa, we rearrange the Query heads and the parameter matrix O according to their parity as Figure2.

The model evaluation and fine-tuning process are the same as in the LLaMA and Vicuna pruning. The 10% sparse model underwent one iteration of evaluation and pruning, while the 20% sparse model underwent two iterations of evaluation and pruning. After evaluation and pruning, all models are fine-tuned on the Alpaca dataset for 4 epochs.

For the random pruning and L2 weight pruning experiments, we also use the same grouping method. The only difference is that during the coupled components and feature evaluation, we don't consider the coupling relationship and only perform random pruning or evaluate based on the sum of L2 values of the kernels containing parameters.

Experimental Analysis. Our pruning algorithm achieves almost no decrease in average scores from BoolQ to OBQA at a sparsity level of 10%. At a sparsity level of 20%, our model retains 94% of the original model's performance. Furthermore, by comparing our algorithm with L2 weight pruning, we find that algorithms like L2 pruning, which are based on pruning based on the magnitude of model parameters, are almost ineffective in structured pruning tasks for LLMs. This evaluation method doesn't consider the dependencies between different coupled components, making it unsuitable for such coarse-grained structured pruning. Our algorithm, on the other hand, considers the coupling relationship between different components and the errors that may arise in the model's inference process after eliminating these components. Therefore, it performs better in structured pruning tasks for LLMs.

The inference performance and storage overhead of our pruned models are shown in Table 5. Our algorithm reduces MACs overhead by 30% at a sparsity level of 20%.

4.3 More Analysis

Global Pruning vs. Layer-wise Pruning. During coupled component elimination, we can employ layer-wise sorted pruning or global sorted pruning methods. However, during our initial experimentation with global ranking, we find that the global sorting approach was not effective. In our pruning experiments, we observe that most low-scoring coupled components are concentrated in the first two layers. However, removing these coupled components results in a significant performance degradation. Additionally, the pruning in LLM-Pruner excludes these layers, there is a need for prior knowledge (Ma et al., 2023) in determining the regions of the model that cannot be pruned. Therefore, we adopt a simpler strategy of uniform pruning (Sun et al., 2023) for every layer.

Kernel vs. Head. When pruning the self-attention layers, we have two options: removing the same number of kernels for each self-attention head or maintaining the same number of kernels per layer but removing one self-attention head in each layer. Based on our experiments with BERT and ViT in Figure3, the latter option performs better when the number of parameters keeps the same. This is because the distribution of importance in the model is not uniform, and low-importance kernels are often concentrated within the same self-attention head. We observe this phenomenon in LLaMA and Vicuna as well. Therefore, our prun-

Pruning Ratio	tune	Method	WikiText2↓	PTB↓	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Ratio=0%	-	ChatGLM3-6B	108.15	169.49	69.54	71.10	56.59	60.69	49.03	31.74	37.40	53.72
Ratio=10%	w/o	Random	338.39	247.57	55.31	66.48	43.77	55.16	47.10	28.41	38.00	47.74
		L2	57580.39	50814.52	53.70	53.10	25.19	49.48	26.26	24.14	36.00	38.26
		Ours	176.24	234.40	51.10	67.57	48.41	55.64	46.21	29.77	36.60	47.89
Ratio=10%	w/	Ours	75.80	95.44	74.31	71.59	52.14	55.56	50.16	32.16	38.20	53.44
Ratio=20%	w/o	Random	967.15	775.58	50.15	60.25	37.46	42.35	34.64	23.46	35.20	40.50
		L2	113621.15	110125.40	49.09	52.82	25.15	49.09	25.29	23.03	35.80	37.18
		Ours	575.63	702.52	38.07	63.16	38.22	53.11	39.56	28.07	35.00	42.17
Ratio=20%	w/	Ours	112.46	140.51	69.54	68.17	47.40	56.35	46.29	30.63	36.60	50.71

Table 4: The pruning experiment for ChatGLM3-6B.

Method	Ratio	#Params	#MACs	Memory
-	-	6.2B	382.5G	11944.8MiB
Ours	10%	5.5B	337.4G	10542.7MiB
Ours	20%	4.8B	295.1G	9249.1MiB

Table 5: Statistic for ChatGLM3.

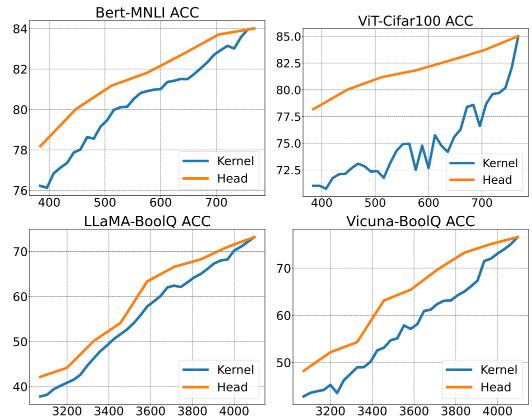

Figure 3: Pruning experiments on BERT, ViT, LLaMA and Vicuna, where the x-axis represents the parameter size of the self-attention layers and the y-axis represents the accuracy of the tasks.

ing strategy for self-attention layers is to remove the lowest-scoring head in each iteration.

Comparison to LLM-Prunner. Our algorithm shares similarities with LLM-Prunner's Channel mode in terms of pruning granularity. Our algorithm prunes features and removes one self-attention head per layer, reducing the size of parameter matrices and the number of self-attention computations, leading to a significant reduction in MACs. However, due to the negative impact from feature pruning, a more accurate evaluation is necessary. Our algorithm evaluates intermediate computation results during inference, offering a more accurate assessment of the impact of structured pruning on model inference performance, compared to LLM-Prunner's element-wise evaluation and summation.

LLM-Prunner's Block mode and our individual kernel-level pruning share similarities in terms of smaller pruning granularity. These operations have minimal impact on the model and enable more fine-grained optimization. However, LLM-Prunner's Block mode uses a global pruning strategy, excluding the first two layers and relying on prior knowledge. In contrast, our algorithm simplifies the process by evaluating multiple kernels as self-attention heads, eliminating the need for prior knowledge.

Furthermore, LLM-Prunner's Block mode alters the structure of certain layers in the model, thus it cannot adopt off-the-shelf libraries for convenient implementation and deployment. In contrast, our algorithm only modifies the size of parameter matrices and reduces the number of self-attention computations while preserving the model's structure. Therefore, our pruned model keeps compatible to existing deep learning programming frameworks, as well as all optimization techniques for Transformer-based models.

5 Conclusion

In this paper, we propose a structured pruning algorithm for LLMs. Our algorithm categorizes parameters into kernels and features based on their relationships between parameter matrices and word vectors in computations. We evaluated these components considering their coupling relationships and the computational characteristics of Transformer architecture. Experimental evaluations on LLaMA, Vicuna, and ChatGLM3 models demonstrated that our algorithm achieves compression to 20% of the original size with minor performance degradation. Our algorithm preserves the model structure, facilitating integration with other optimization techniques and practical deployment.

Limitations

Our algorithm employed a simple uniform pruning scheme across different layers of an LLM, which allows us to avoid acquiring prior knowledge and assumes equal importance for each layer in the model. However, most previous global pruning schemes imply an uneven distribution of importance across different layers of the model, which we did not further explore. In addition, we employed a more empirical approach for intermediate layer pruning, without further exploring the specific number of kernel pairs to be pruned in each layer. Our future work will focus on improving these aspects.

References

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. 2022. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359.

Adrian de Wynter and Daniel J Perry. 2020. Optimal subarchitecture extraction for bert. *arXiv preprint arXiv:2010.10499*.

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

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 320–335.

Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. 2023. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16091–16101.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. 2022. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Chen Liang, Simiao Zuo, Minshuo Chen, Haoming Jiang, Xiaodong Liu, Pengcheng He, Tuo Zhao, and Weizhu Chen. 2021. Super tickets in pre-trained language models: From model compression to improving generalization. *arXiv preprint arXiv:2105.12002*.

Christos Louizos, Max Welling, and Diederik P Kingma. 2017. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems*.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11264–11272.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2:13.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*.

Huanrui Yang, Hongxu Yin, Maying Shen, Pavlo Molchanov, Hai Li, and Jan Kautz. 2023. Global vision transformer pruning with hessian-aware saliency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18547–18557.

Ofir Zafrir, Ariel Larey, Guy Boudoukh, Haihao Shen, and Moshe Wasserblat. 2021. Prune once for all: Sparse pre-trained language models. *arXiv preprint arXiv:2111.05754*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, et al. 2022. Glm-130b: An open bilingual pre-trained model. *arXiv preprint arXiv:2210.02414*.

Mingyang Zhang, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, Bohan Zhuang, et al. 2023. Pruning meets low-rank parameter-efficient fine-tuning. *arXiv preprint arXiv:2305.18403*.

Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. 2022a. Platon: Pruning large transformer models with upper confidence bound of weight importance. In *International Conference on Machine Learning*, pages 26809–26823. PMLR.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022b. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.