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-
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 propose 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 al-
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 Transformer, 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 \( U P \in \mathbb{R}^{im \times d_m} \) 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):

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

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 \( \text{len} \) 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., \( Q = [q_1, q_2, ..., q_{d_k}]^t \), \( K = [k_1, k_2, ..., k_{d_k}]^t \), 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):

\[
Q^t K = \sum_{i=1}^{d_k} q_i^t k_i. \tag{2}
\]

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 \( v_i^t o_i \). 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):

\[
\text{Out} = f_{c_2} F(f_{c_1} X). \tag{3}
\]

Here, \( f_{c_1} \in \mathbb{R}^{im \times d_m} \) and \( f_{c_2} \in \mathbb{R}^{d_m \times im} \). \( F \) 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):

\[
\text{Out} = \text{Down}(F(Gate \cdot X) \odot Up X). \tag{4}
\]

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: \( d_i g_i \),
and \( d_i u_i \), 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 sub-components \( d_i g_i \) and \( d_i u_i \) 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 \( \alpha \) and a row vector \( \beta \). We denote such coupled components as \( C = \alpha \beta \), where \( C \in \mathbb{R}^{dm \times dm} \). During the evaluation process, we evaluate the importance of the coupled component \( C \) by measuring the error in neural network predictions 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):

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

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):

\[
\begin{align*}
I_t^{(0)} &= x_1 I_{t-1}^{(0)} + (1 - x_1) I_{t-1}^{(0)}, \\
U_C^{(0)} &= |x_t - I_{t-1}^{(0)}|, \\
U_C^{(0)} &= x_2 U_C^{(t-1)} + (1 - x_2) U_C^{(t-1)}, \\
S_C &= \sum_t I_C^{(0)} U_C^{(0)}. \tag{7}
\end{align*}
\]

Here, \( t \) represents the current iteration of evaluation for the variable. \( I_C \) 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). \( U_C \) represents the upper bound confidence for \( I_C \) (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):

\[
I_1 \approx \sum_{c \in C} \left| \sum_{k \in C} \frac{\partial \mathcal{L}}{\partial c} c - \frac{1}{2} \left( \frac{\partial \mathcal{L}}{\partial c} c \right)^2 \right|. \tag{8}
\]

Here, \( C \) refers to the \( Q^T K \) and \( V^T O \) 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., \( Q = [q_1, q_2, \ldots, q_{dm}] \) and \( K = [k_1, k_2, \ldots, k_{dm}] \), 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
### 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.

<table>
<thead>
<tr>
<th>Remaining Ratio</th>
<th>Tune Method</th>
<th>WikiText2</th>
<th>PTB</th>
<th>BoolQ</th>
<th>PIQA</th>
<th>HellaSwag</th>
<th>Winogrande</th>
<th>ARC-c</th>
<th>ARC-e</th>
<th>OBQA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>100% w/</td>
<td>LLaMA-7B</td>
<td>12.62</td>
<td>22.14</td>
<td>73.18</td>
<td>78.35</td>
<td>72.99</td>
<td>67.01</td>
<td>67.45</td>
<td>41.38</td>
<td>42.40</td>
<td>63.25</td>
</tr>
<tr>
<td>20% w/</td>
<td>LP-Channel</td>
<td>74.63</td>
<td>153.75</td>
<td>62.75</td>
<td>62.73</td>
<td>61.40</td>
<td>51.07</td>
<td>41.38</td>
<td>27.90</td>
<td>30.40</td>
<td>45.38</td>
</tr>
<tr>
<td>20% w/o</td>
<td>LP-Block</td>
<td>19.24</td>
<td>34.09</td>
<td>62.54</td>
<td>75.41</td>
<td>65.99</td>
<td>60.30</td>
<td>61.57</td>
<td>36.09</td>
<td>39.20</td>
<td>57.39</td>
</tr>
<tr>
<td>24% w/</td>
<td>Ours</td>
<td>37.90</td>
<td>74.30</td>
<td>66.51</td>
<td>73.39</td>
<td>62.11</td>
<td>62.90</td>
<td>58.24</td>
<td>35.75</td>
<td>36.20</td>
<td>56.45</td>
</tr>
<tr>
<td>24% w/o</td>
<td>LP-Block</td>
<td>17.39</td>
<td>30.20</td>
<td>66.79</td>
<td>77.58</td>
<td>68.48</td>
<td>64.96</td>
<td>64.06</td>
<td>37.88</td>
<td>39.00</td>
<td>59.82</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>22.00</td>
<td>42.58</td>
<td>72.26</td>
<td>75.13</td>
<td>68.87</td>
<td>66.53</td>
<td>63.29</td>
<td>38.73</td>
<td>41.40</td>
<td>60.88</td>
</tr>
<tr>
<td>24% w/</td>
<td>Ours</td>
<td>34.55</td>
<td>72.14</td>
<td>63.36</td>
<td>69.96</td>
<td>55.92</td>
<td>60.37</td>
<td>53.19</td>
<td>33.70</td>
<td>35.40</td>
<td>53.12</td>
</tr>
<tr>
<td>24% w/o</td>
<td>Ours</td>
<td>25.01</td>
<td>46.79</td>
<td>68.47</td>
<td>73.88</td>
<td>65.88</td>
<td>63.53</td>
<td>59.63</td>
<td>35.58</td>
<td>38.00</td>
<td>57.85</td>
</tr>
</tbody>
</table>

### 3.4 Overall Process
This section summarizes 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 pruning 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.](image)

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
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 Figure 2.

The model evaluation and fine-tuning process are the same as in the LLaMA and Vicuna pruning. The 10% sparse model undertook 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 Figure 3, 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-
Table 4: The pruning experiment for ChatGLM3-6B.

<table>
<thead>
<tr>
<th>Pruning Ratio</th>
<th>tunne Method</th>
<th>WikiText2</th>
<th>PTB</th>
<th>BoolQ</th>
<th>PIQA</th>
<th>HellaSwag</th>
<th>Winograd</th>
<th>ARC-e</th>
<th>ARC-c</th>
<th>ORQA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio=0%</td>
<td>ChatGLM3-6B</td>
<td>108.15</td>
<td>160.49</td>
<td>69.54</td>
<td>71.10</td>
<td>56.59</td>
<td>60.69</td>
<td>49.03</td>
<td>31.74</td>
<td>37.40</td>
<td>53.72</td>
</tr>
<tr>
<td>Ratio=10%</td>
<td>Random</td>
<td>338.39</td>
<td>247.57</td>
<td>55.31</td>
<td>66.48</td>
<td>43.77</td>
<td>55.16</td>
<td>47.10</td>
<td>28.41</td>
<td>38.00</td>
<td>47.74</td>
</tr>
<tr>
<td></td>
<td>L2</td>
<td>5758.39</td>
<td>50814.52</td>
<td>55.70</td>
<td>53.10</td>
<td>25.19</td>
<td>49.48</td>
<td>26.26</td>
<td>24.14</td>
<td>36.00</td>
<td>38.26</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>178.26</td>
<td>234.40</td>
<td>51.10</td>
<td>67.17</td>
<td>48.41</td>
<td>55.64</td>
<td>46.11</td>
<td>29.77</td>
<td>36.80</td>
<td>47.89</td>
</tr>
<tr>
<td>Ratio=10%</td>
<td>Ours</td>
<td>75.80</td>
<td>93.44</td>
<td>54.31</td>
<td>71.59</td>
<td>52.14</td>
<td>55.86</td>
<td>50.16</td>
<td>32.16</td>
<td>38.20</td>
<td>53.44</td>
</tr>
<tr>
<td>Ratio=20%</td>
<td>Random</td>
<td>967.15</td>
<td>775.58</td>
<td>50.00</td>
<td>50.25</td>
<td>37.46</td>
<td>42.35</td>
<td>34.64</td>
<td>23.46</td>
<td>35.20</td>
<td>40.50</td>
</tr>
<tr>
<td></td>
<td>L2</td>
<td>113621.15</td>
<td>110125.40</td>
<td>49.09</td>
<td>52.82</td>
<td>25.15</td>
<td>49.09</td>
<td>25.29</td>
<td>23.03</td>
<td>35.80</td>
<td>37.18</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>575.63</td>
<td>702.52</td>
<td>38.07</td>
<td>63.18</td>
<td>38.22</td>
<td>51.11</td>
<td>39.56</td>
<td>28.07</td>
<td>35.00</td>
<td>42.17</td>
</tr>
<tr>
<td>Ratio=20%</td>
<td>Ours</td>
<td>112.46</td>
<td>140.51</td>
<td>69.54</td>
<td>68.17</td>
<td>47.40</td>
<td>56.35</td>
<td>46.29</td>
<td>30.63</td>
<td>36.60</td>
<td>50.71</td>
</tr>
</tbody>
</table>

Table 5: Statistic for ChatGLM3.

<table>
<thead>
<tr>
<th>Method</th>
<th>Ratio</th>
<th>#Params</th>
<th>#MACs</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6.2B</td>
<td>352.5G</td>
<td>11944.8MB</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>10%</td>
<td>5.5B</td>
<td>337.4G</td>
<td>10542.7MiB</td>
</tr>
<tr>
<td>Ours</td>
<td>20%</td>
<td>4.8B</td>
<td>295.1G</td>
<td>9249.1MiB</td>
</tr>
</tbody>
</table>

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


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.


