Pruning before Fine-tuning: A Retraining-free Compression Framework for Pre-trained Language Models

Pingjie Wang[1,2], **Hongcheng Liu**[1], **Yu Wang**[1,2]*, **Yanfeng Wang**[1,2]
[1]Cooperative Medianet Innovation Center, Shanghai Jiao Tong University
[2]Shanghai Artificial Intelligence Laboratory†
{pingjiewang, hongcheng_liu, yuwangsjtu, wangyanfeng622}@sjtu.edu.cn

Abstract

Structured pruning is an effective technique for compressing pre-trained language models (PLMs), reducing model size and improving inference speed for efficient deployment. However, most of existing pruning algorithms require retraining, leading to additional computational overhead. While some retraining-free approaches have been proposed for classification tasks, they still require a fully fine-tuned model for the task, and may cause catastrophic performance degradation on generative tasks. To address these challenges, we propose P-pruning (pre-pruning), an innovative task-specific compression framework. P-pruning prunes redundant modules of PLMs before fine-tuning, reducing the costs associated with fine-tuning. We also introduce a pruning algorithm for this framework, which includes two techniques: (i) module clustering, which clusters the outputs of all heads and neurons based on the task input; and (ii) centroid selection, which identifies the most salient element in each cluster and prunes the others. We apply our method to BERT and GPT-2 and evaluate its effectiveness on GLUE, SQuAD, WikiText-2, WikiText-103, and PTB datasets. Experimental results demonstrate that our approach achieves higher performance in both classification and generative tasks, while also reducing the time required for fine-tuning. Our code is publically released at https://github.com/applewpj/P-pruning.

Keywords: pre-trained language models, retraining-free compression, structured pruning

1. Introduction

In recent years, pre-trained language models (PLMs) have attained significant achievement on various downstream tasks (Frankle and Carbin, 2018; Zhou et al., 2023). However, despite the substantial development, the deployment of PLMs leads to high parameter counts and significant computational overhead, which greatly limits the applications in practice. To tackle this challenge, a variety of task-specific compression approaches have emerged to reduce the model size and accelerate the inference process (Gupta and Agrawal, 2022). Structured pruning is one of the most effective methodologies among various compression approaches (Xia et al., 2022; Tao et al., 2023), especially for the principal counterpart of PLMs, transformers (Vaswani et al., 2017), which is becoming commonly used in many domains including computer vision (Touvron et al., 2021) and speech recognition (Baevski et al., 2020; Hsu et al., 2021).

While most of the existing works for task-specific model compression on pruning transformers of PLMs substantially reduce the model size and inference latency (Lagunas et al., 2021; Xia et al., 2022), these approaches are still restricted when employed in practice as they require retraining, which introduces extra computational overhead and additional engineering efforts such as hyperparameter search and training code rewriting.

For this reason, Kwon et al. (2022) and Nova et al. (2023) proposed a pruning framework for the fine-tuned model, which avoids expensive retraining and retains the comparable performance on classification tasks. However, although such works avoid additional retraining and therefore reduce the pruning time, there are still two problems. (i) Firstly, considering the time costs required for a pruned task-specific model is composed of fine-tuning and pruning, such methods still require a fully fine-tuned model to prune, which does not bring any reduction in time consumption for fine-tuning. (ii) Furthermore, they only conduct experiments for encoder-based models (e.g. BERT (Devlin et al., 2018)) on sequence classification and question-answering tasks, in which the results are easier to reconstruct compared to decoder-based models (e.g. GPT-2 (Radford et al., 2019)) and generative tasks. Our extended experiments also demonstrate that such works may exhibit poor performance on generative tasks.

To address the mentioned issues caused by traditional retraining-free pruning methods, this paper proposes P-pruning (Pre-pruning), a novel framework for compressing task-specific models. P-pruning prunes redundant modules before fine-tuning for the downstream task, reducing the overhead of fine-tuning, especially at high compression rates. Despite compression, P-pruning re-

* Corresponding author
† This work is conducted during an internship at Shanghai Artificial Intelligence Laboratory.

LREC-COLING 2024, pages 13279–13289
20-25 May, 2024.

Method	Re-training free	Efficient pruning	Efficient tuning	Generative tasks
DynaBERT (Hou et al., 2020)	✗	✗	✗	N/A
EBERT (Liu et al., 2021)	✗	✗	✗	N/A
Mask-Tuning (Kwon et al., 2022)	✓	✓	✗	✗
KCM (Nova et al., 2023)	✓	✓	✗	✗
Ours	✓	✓	✓	✓

Table 1: Comparison between various structured pruning methods for PLMs with respect to different aspects. ✗ and ✓ represent whether the method had the specific feature or not. N/A means there have not been extensive experiments to prove the methods are applicable.

tains comparable performance on generative tasks. The main challenge of P-pruning is identifying the salient modules for a specific task. To tackle this, we propose Dual-CP (**Dual C**lustering **P**runing for heads and neurons), an efficient and retraining-free unsupervised pruning algorithm. Dual-CP consists of two procedures: module clustering, which clusters the modules (e.g. attention heads) of the pre-trained model, and centroid selection, which retains the salient element of each cluster and prunes the others. The key difference between this approach and task-agnostic model compression is that the informative sub-network of the pre-trained model is derived from the dataset of the specific task, making it adaptive for different tasks. It is also noted that P-pruning can be combined with any other unsupervised task-specific compression methods. Table 1 summarizes the comparison between existing representative structured pruning methods and our proposed approach for various main features focused on model compression. Our method achieves all the listed features, making it highly effective for compressing large pre-trained models for a specific task.

In summary, our contributions are three-fold:

- **Framework** We propose P-pruning, a novel task-specific compression framework, which can be combined with any unsupervised compression methods. Its goal is to prune redundant modules for the specific task before fine-tuning, resulting in significant time savings during the fine-tuning process.
- **Algorithm** Within the P-pruning framework, we propose Dual-CP, an efficient and retraining-free algorithm to tackle the challenge of P-pruning: identifying the salient sub-network in a given pre-trained model and task. Specifically, our Dual-CP involves two steps: (i) module clustering, which clusters the outputs of attention heads and neurons based on the task input, and (ii) centroid selection, which identifies the most important element to represent the entire cluster and prunes the others.
- **Performance** Our evaluation results on classification tasks, such as GLUE and SQuAD benchmarks, demonstrate lower performance degradation and significantly reduced fine-tuning time compared to traditional retraining-free pruning methods. Additionally, we empirically show that our method achieves more stable and satisfactory performance for language modeling tasks on WikiText-2, WikiText-103, and PTB datasets.

2. Related Works

2.1. Compression for PLMs

In recent years, multiple compression approaches have been proposed to reduce the memory footprint and accelerate the inference process. They are broadly categorized with different aspects as: (i) quantization (Zafrir et al., 2019; Shen et al., 2020; Frantar et al., 2022), (ii) low-rank approximation (Cahyawijaya, 2021; Hsu et al., 2022; Yu and Wu, 2023), (iii) efficient architecture design (Wang et al., 2020; Sun et al., 2020; Lan et al., 2019; Kitaev et al., 2020), (iv) knowledge distillation (Sun et al., 2019; Sanh et al., 2019; Jiao et al., 2019) and (v) pruning (Sanh et al., 2020; Kurtic et al., 2022). All of the above methods are compatible to achieve a higher compression rate as they address different kinds of redundancy. In this paper, we focus on pruning especially structured pruning, because structured pruning is friendly to hardware acceleration (Neill, 2020) and is able to compress the model without retraining.

2.2. Structured Pruning for Transformers

Transformer (Devlin et al., 2018) is a principal component of PLMs and contributes to the majority of model parameters and computation overhead. Therefore, in this paper we chose it as the target for pruning. Existing structured pruning algorithms can be divided into two groups: retraining-based (Hou et al., 2020; Liu et al., 2021; Wang et al., 2019) and retraining-free approaches (Kwon et al., 2022; Nova et al., 2023). Earlier retraining-based methods generate highly sparse but accurate models. However, they require careful retraining to ensure

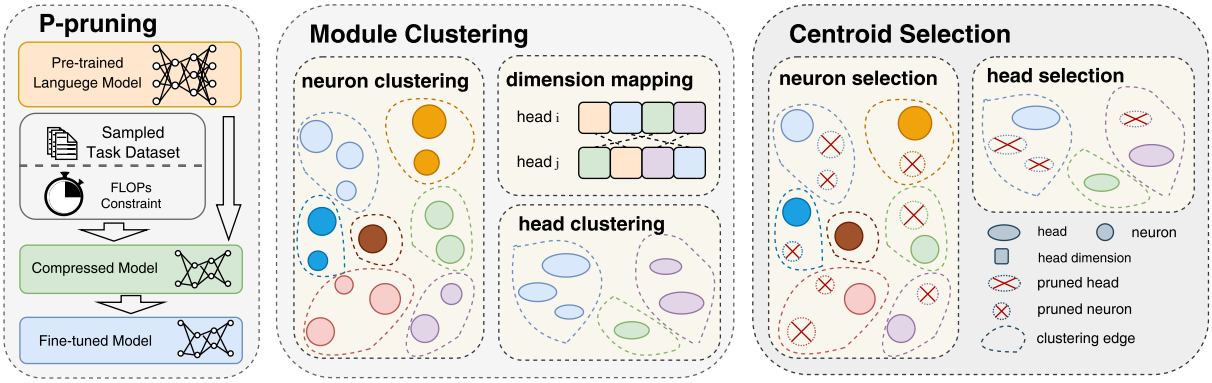

Figure 1: Framework overview of the proposed method. **Left**: the compression pipeline of P-pruning, which prunes the pre-trained model for efficient fine-tuning. **Middle**: the procedures of module clustering for attention heads and neurons respectively, in which we conduct dimension mapping before head clustering. **Right**: centroid selection that keeps the heads and neurons that contribute most and prune others.

the reconstruction of the output and maintain performance, which leads to a tenfold increase in training time (Xia et al., 2022; Lagunas et al., 2021) and introduces additional computational overhead and engineering efforts (e.g. hyperparameter search). By contrast, retraining-free algorithms address this problem by directly pruning the fine-tuned model without any retraining.

2.3. Retraining-free Pruning Methods

(Kwon et al., 2022) first utilized the Fisher information to prune the elements and regain performance through Mask-Tuning. However, this approach still necessitates the use of task labels to conduct the pruning process, which could pose challenges in scenarios where labeled data is unavailable. In response to this issue, (Nova et al., 2023) introduced KCM, an unsupervised pruning metric designed for retraining-free pruning. However, these algorithms still require a fully fine-tuned model to identify and remove redundant modules. It is worth noting that the fine-tuned model itself is more redundant and easier to compress (Sajjad et al., 2022), making this process somewhat wasteful in terms of fine-tuning for the modules that will eventually be pruned.

To prune the needless modules before fine-tuning, Sajjad et al. (2023) conducts experiments to show that a number of layers can be pruned without retraining prior to fine-tuning for tasks, without experiencing significant performance degradation. However, this pruning technique only operates at the layer level, which may not adequately meet computational constraints. Additionally, it does not provide guidance on determining the optimal number of layers to prune before fine-tuning while still maintaining an acceptable level of performance degradation. In this paper, we implement pruning in a more fine-grained manner, offering guidance on identifying the redundant heads and neurons specific to tasks.

3. Methodology

3.1. Preliminaries

Transformer Encoder Our method is both applicable of encoder- and decoder-based PLMs, but as most existing efficient structured pruning approaches focus on encoder-based models (e.g. BERT) which is a stack of transformer (Vaswani et al., 2017) encoder blocks, we only introduce the encoder architecture which shares a similar structure with the decoder. A transformer encoder block consists of a multi-head attention (MHA) layer followed by a point-wise feed-forward network (FFN).

For a given input $\boldsymbol{X} \in \mathbb{R}^{T\times d}$, where T and d represent the sequence length and the embedding dimension respectively, the output of the MHA layer and FFN layer can be formulated as $\mathrm{LayerNorm}(\boldsymbol{X} + \mathrm{Sub}(\boldsymbol{X}))$, where $\mathrm{Sub}(\boldsymbol{X})$ is the sublayer function either to be $\mathrm{MHA}(\boldsymbol{X})$ or $\mathrm{FFN}(\boldsymbol{X})$. To be specific, we define $\boldsymbol{H}^A \triangleq \mathrm{Attn}(\boldsymbol{X})$, where $\mathrm{Attn}(\boldsymbol{X})$ are the hidden features calculated by the query, key, value matrices. $\mathrm{MHA}(\boldsymbol{X})$ with H attention heads is calculated as

$$\mathrm{MHA}(\boldsymbol{X}) = \boldsymbol{H}^A\mathbf{W}^A + \boldsymbol{b}^A = \sum_{i=1}^{H} \boldsymbol{H}^A[i]\mathbf{W}^A[i] + \boldsymbol{b}^A \quad (1)$$

where $\boldsymbol{b}^A \in \mathbb{R}^d$ is a bias. In addition, $\boldsymbol{H}^A[i] \in \mathbb{R}^{T\times d_h}$ and $\mathbf{W}^A[i] \in \mathbb{R}^{d_h\times d}$ are the hidden features and output weight matrix for i-th head with $d_h = d/H$ respectively. Similarly, we define $\boldsymbol{H}^F \triangleq \sigma(\mathbf{W}\boldsymbol{X}+\boldsymbol{b})$, where $\sigma(.)$ denotes the activation function, typically GELU (Hendrycks and Gimpel, 2016) in BERT, $\mathbf{W} \in \mathbb{R}^{d\times N}$ and $\boldsymbol{b} \in \mathbb{R}^N$ are the input weight and bias of FFN respectively. $\mathrm{FFN}(\boldsymbol{X})$ with

Algorithm 1 Dual-CP.

Input: Pre-trained model $\mathcal{M}$, FLOPs constraint $\mathcal{C}$, a task dataset $\mathcal{D}$, constraint ratio λ and step δ.
Output: Mask variables $\boldsymbol{m}^A$ and $\boldsymbol{m}^H$.

Collect randomly sampled hidden inputs $\boldsymbol{H}^A_{(l)}, \boldsymbol{H}^F_{(l)}$ for each layer (l).
Initialize $\mathcal{C}^A$ and $\mathcal{C}^F$ to satisfy $\mathcal{C}^F = \lambda \mathcal{C}^A$ and $\mathcal{C} = \mathcal{C}^A + \mathcal{C}^F$
$\mathcal{T}^F, \mathcal{T}^A$ = *MC*($\boldsymbol{H}^F$, $\boldsymbol{H}^A$) ▷ Apply module clustering, see Algorithm 2.
for clustering tree of heads and neurons **do**
  Initialize t as 0.
  repeat
    $t \leftarrow t + \delta$ ▷ Determine the threshold to cluster elements.
    Derive $K^F_{(l)}$ or $K^A_{(l)}$ clusters with t for $\mathcal{T}^F$ or $\mathcal{T}^A$.
  until The FLOPs constraint $\mathcal{C}^A$ or $\mathcal{C}^F$ is satisfied.
end for
Initialize $\boldsymbol{m}^H$ and $\boldsymbol{m}^A$ as $\mathbf{0}$.
for (l) in layers of $\mathcal{M}$ **do**
  Select the elements with maximum average activations in each cluster.
  Set the corresponding mask variable (e.g. $\boldsymbol{m}^F_{(l)}[i]$ for i-th neuron) to be 1, while others remain as 0.
end for

N intermediate neurons is obtained by

$$\mathrm{FFN}(\boldsymbol{X}) = \boldsymbol{H}^F \mathbf{W}^F + \boldsymbol{b}^F = \sum_{j=1}^{N} \boldsymbol{H}^F[j] \mathbf{W}^F[j] + \boldsymbol{b}^F \quad (2)$$

where $\boldsymbol{H}^F[j] \in \mathbb{R}^{T\times 1}$, $\mathbf{W}^F[j] \in \mathbb{R}^{1\times d}$ are the activations and the output matrix of j-th neuron with bias $\boldsymbol{b}^F \in \mathbb{R}^d$.

Problem Definition Given a PLM $\mathcal{M}$ with inherent computational complexity $\mathcal{I}$ and constraint $\mathcal{C}$ (in which we utilize floating point operations (FLOPs) to serve as the computational complexity metric), the pruning task is to find the most informative subnetwork $\mathcal{S}$ which satisfies $\mathrm{FLOPs}(\mathcal{S}) < \mathcal{C}$ to retain the performance after fine-tuning to specific tasks, in which $\mathcal{C}/\mathcal{I}$ ranges from 0 to 1. To formalize this, we introduce two masks $\boldsymbol{m}^A \in \mathbb{R}^H$ and $\boldsymbol{m}^F \in \mathbb{R}^N$ for the outputs of attention heads and FFN neurons respectively, and the pruned MHA and FFN layers of $\mathcal{S}$ are calculated as

$$\begin{aligned} \mathrm{MHA}(\boldsymbol{X}; \boldsymbol{m}^A) &= \sum_{i=1}^{H} \boldsymbol{m}^A[i] \boldsymbol{H}^A[i] \mathbf{W}^A[i] + \boldsymbol{b}^A \\ \mathrm{FFN}(\boldsymbol{X}; \boldsymbol{m}^F) &= \sum_{j=1}^{N} \boldsymbol{m}^F[j] \boldsymbol{H}^F[j] \mathbf{W}^F[j] + \boldsymbol{b}^F, \end{aligned} \quad (3)$$

where $\boldsymbol{m}^A[i], \boldsymbol{m}^F[j] \in \{0, 1\}$. Originally, the mask variables are initialized to 1, which makes the output of $\mathcal{S}$ equal to $\mathcal{M}$. After pruning, the heads or neurons whose mask values remain as 1 are considered to be salient and reserved, while others are pruned. It is noticed that the mask variables are multiplied with the output matrices (e.g. $\boldsymbol{m}^F \circ \mathbf{W}^F$) after pruning, where $\circ$ denotes the Hardmard product. Therefore, they do not bring extra parameters and FLOPs.

3.2. Framework Overview

Compared to traditional task-specific compression, we start pruning from a pre-trained model instead of a fully fine-tuned model, where the latter is more redundant and easier for compression (Sajjad et al., 2022) but requires full fine-tuning before such lightweight operation. Therefore, we propose P-pruning to combine efficient fine-tuning with efficient compression, which is shown in the left part of Figure 1.

Existing pruning approaches (Kwon et al., 2022; Nova et al., 2023) require a fine-tuned model with a prediction head for the downstream task. With the added prediction head, (Kwon et al., 2022) uses gradients from the labeled dataset $\mathcal{D}_l$ to find the most important heads and neurons. However, this approach can not be applied for pre-trained models without a prediction head. In contrast, Our method prunes using the unlabeled dataset $\mathcal{D}$. Unlike KCM (Nova et al., 2023), which only prunes FFN neurons, our method prunes both attention heads and FFN neurons together for better compression.

Given a pre-trained model, FLOPs constraint, and sampled dataset for a downstream task, the main challenge of structured pruning before fine-tuning is to identify and remove redundant modules while preserving the key ones. To tackle this problem, we propose Dual-CP as shown in Algorithm 1, which consists of two procedures: (i) module clustering to group similar activations from attention heads and FFN neurons, and (ii) centroid selection to choose the most important element in each cluster and prune the rest.

3.3. Module Clustering

For a sampled dataset $\mathcal{D}$ with s tokens for a specific downstream task, we first conduct the forward

Algorithm 2 Module Clustering (MC).

Input: Hidden inputs $\boldsymbol{H}^F$ and $\boldsymbol{H}^A$.
Output: Clustering trees $\mathcal{T}^F$ and $\mathcal{T}^A$.

for (l) in layers of $\mathcal{M}$ **do**
  for each two neurons i and j **do**
    $\mathbf{C}^F_{(l)}[i,j] = |\text{sim}(\boldsymbol{H}^F_{(l)}[i], \boldsymbol{H}^F_{(l)}[j])|$
  end for
  for each two heads i and j **do**
    for each two dimensions m and n **do**
      $\mathbf{C}^{map}_{(l),i,j}[m,n] = |\text{sim}(\boldsymbol{H}^A_{(l)}[i,m], \boldsymbol{H}^A_{(l)}[j,n])|$
    end for
    $\mathbf{C}^A_{(l)}[i,j] = \frac{1}{H}\sum \text{Map}(\mathbf{C}^{map}_{(l),i,j})$
  end for
  $\mathbf{D}^F_{(l)} = \mathbf{J} - |\mathbf{C}^F_{(l)}|$
  $\mathbf{D}^A_{(l)} = \mathbf{J} - \mathbf{C}^A_{(l)}$
  Apply agglomerative hierarchical clustering to $\mathbf{D}^A_{(l)}$ and $\mathbf{D}^F_{(l)}$ to obtain the $\mathcal{T}^F$ and $\mathcal{T}^A$.
end for

pass for the pre-trained language model $\mathcal{M}$ with L transformer blocks, LH attention heads and LN FFN neurons, and collect the hidden activations for each head and neuron as Equations (1) and (2). To reduce the computaion costs, we only randomly sample T tokens to apply clustering, that is, $\boldsymbol{H}^A_{(l)} \in \mathbb{R}^{T\times d}$ and $\boldsymbol{H}^F_{(l)} \in \mathbb{R}^{T\times d}$ for l-th layer.

As the pre-trained model is stacked by an embedding layer and a number of transformer blocks, in which the embedding layer is a look-up operation, and hence does not bring extra FLOPs, the total FLOPs are only derived by MHA and FFN layers, denoted as $\mathcal{I}^A, \mathcal{C}^A$ and $\mathcal{I}^F, \mathcal{C}^F$. $\mathcal{C}$ is formulated as $\mathcal{I} = \mathcal{I}^A + \mathcal{I}^F$ and $\mathcal{C} = \mathcal{C}^A + \mathcal{C}^F$. As we prune attention heads and FFN neurons simultaneously, which has different sensitivity to re-training free pruning (which is observed in the experimental results of (Kwon et al., 2022)), we assign different FLOPs constraint with them respectively and apply λ to balance the pruned FLOPs of MHA and FFN layers, which satisfies

$$\mathcal{I}^F - \mathcal{C}^F = \lambda(\mathcal{I}^A - \mathcal{C}^A). \tag{4}$$

Afterwards, we cluster neurons and attention heads respectively, which is summarized in Algorithm 2.

Neuron Clustering Clustering is applied between FFN neurons or attention heads for each layer separately, which is illustrated in the middle and right part of Figure 1. To cluster the similar FFN neurons, for layer (l), we first calculate the absolutized cosine similarity of the activations $\boldsymbol{H}^F_{(l)}[i]$ and $\boldsymbol{H}^F_{(l)}[j]$ between each two neurons i and j, which is formulated as

$$\mathbf{C}^F_{(l)}[i,j] = |\text{sim}(\boldsymbol{H}^F_{(l)}[i], \boldsymbol{H}^F_{(l)}[j])|, \tag{5}$$

where $i,j \in \{1,...,d_f\}$, and thereby we obtain the similarity matrix $\mathbf{C}^F_{(l)} \in \mathbb{R}^{d_f\times d_f}$. Each element in the similarity matrix ranges from 0 to 1, and the closer the element value is to 1, the more similar the corresponding two neurons are, which represents that they encode highly correlated information and thus are redundant.

To evaluate the most correlated neuron groups, we convert the similarity matrix into a distance matrix $\mathbf{D}^F_{(l)}$ by

$$\mathbf{D}^F_{(l)} = \mathbf{J} - |\mathbf{C}^F_{(l)}|, \tag{6}$$

where $\mathbf{J}$ denotes an all-1 matrix. As we take the absolutized value of each element in $\mathbf{C}^F_{(l)}$, we avoid the possibility of a negative distance value.

After that, we use agglomerative hierarchical clustering to cluster the distance matrix $\mathbf{D}^F_{(l)}$ of each layer. We apply average linkage to minimize the average distance between data points in a pair of clusters. The maximum distance between neurons in the same cluster is controlled by a threshold t ranging from 0 to 1. A high threshold results in few but large clusters, while a low threshold creates numerous small clusters. We keep only one neuron for each cluster and remove redundant ones. The value of t is automatically set to satisfy the FLOPs constraint $\mathcal{C}^F$. It starts at 0 and increases linearly up to 1 with a step δ until the remaining neurons in L layers meet the requirement. So far, we obtain $K^F_{(l)}$ clusters for the l-th layer, with cluster k_i containing neurons where $i \in \{1,...,K^F_{(l)}\}$.

Head Clustering For the attention architecture, instead of pruning one neuron each time, we prune a whole attention head, which brings a larger granularity of pruning. Different from clustering FFN neurons where each neuron has only one dimension, an attention head contains d_h dimensions. Thus, to measure the correlation of each two heads, we first map dimensions between them by calculating the cosine similarity of the internal dimension as

$$\mathbf{C}^{map}_{(l),i,j}[m,n] = |\text{sim}(\boldsymbol{H}^A_{(l)}[i,m], \boldsymbol{H}^A_{(l)}[j,n])|, \tag{7}$$

which indicates the similarity of m-th dimension of i-th head and n-th dimension of j-th head, and $i,j \in \{1,...,H\}$, $m,n \in \{1,...,d_h\}$ respectively. For each two heads, we first map the dimensions with the highest correlation as a pair, and then map the others in the same manner.

After dimension mapping, the similarity values of mapped dimensions are averaged and serve as the correlation $\mathbf{C}^A_{(l)}[i,j]$ of i-th and j-th attention heads, which is formulated as

$$\mathbf{C}^A_{(l)}[i,j] = \frac{1}{H}\sum \text{Map}(\mathbf{C}^{map}_{(l),i,j}), \tag{8}$$

where $\text{Map}(.)$ is a mapping function to map the head dimensions in descending order of similarity based on $\mathbf{C}^{map}_{(l),i,j}$, and yields a $\mathbb{R}^H$ similarity vector for heads i and j.

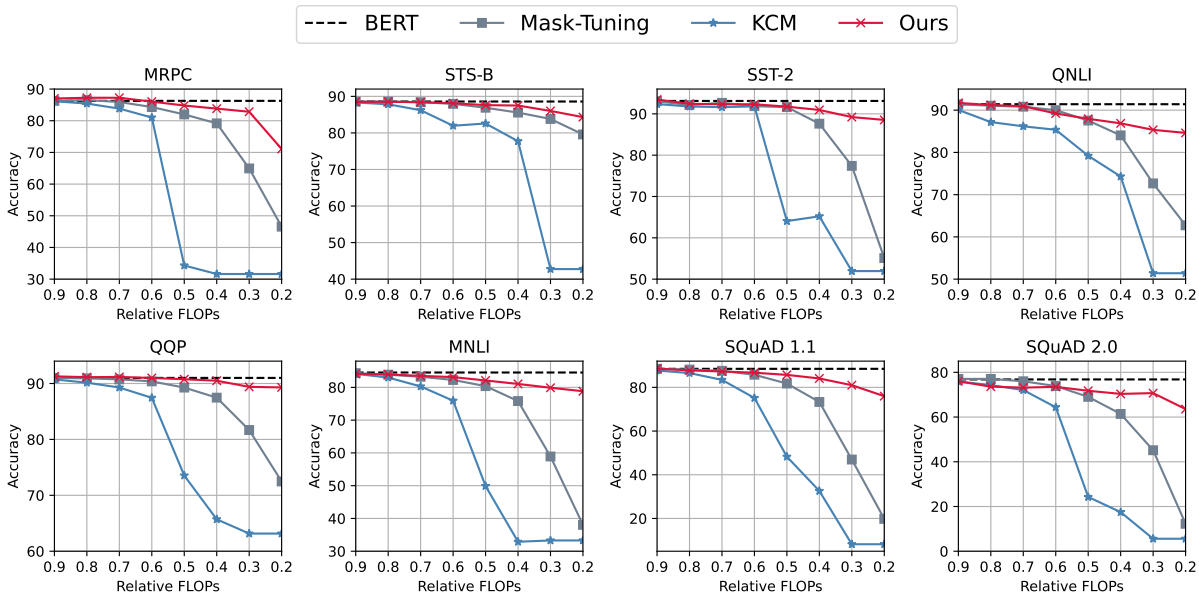

Figure 2: Accuracy of our pruning framework applied to $\text{BERT}_{\text{BASE}}$ on GLUE and SQuAD benchmarks, against other retraining-free methods, Mask-Tuning and KCM with different FLOPs constraints ranging from 90% to 20% (i.e., 10%~80% FLOPs are reduced). The black dashed line indicates the performance of $\text{BERT}_{\text{BASE}}$. Our method outperforms other competitors in almost all settings.

The obtained cosine similarity matrix $\mathbf{C}^A_{(l)} \in \mathbb{R}^{H\times H}$ is also converted to a distance matrix $\mathbf{D}^A_{(l)}$ in a similar manner with Equation (6) as

$$\mathbf{D}^A_{(l)} = \mathbf{J} - \mathbf{C}^A_{(l)}. \tag{9}$$

The subsequent clustering operations are mostly similar to the ones for FFN neurons as described above and obtain $K^A_{(l)}$ clusters for each layer. The only difference is that attention heads clustering is supposed to satisfy the attention FLOPs constraint $\mathcal{C}^A$ instead of $\mathcal{C}^F$. It is noted that different layers share the same threshold t for pruning heads or neurons respectively, which therefore brings unequal sparsity for each layer.

3.4. Centroid Selection

As mentioned earlier, we derive clusters $K^A_{(l)}$ and $K^F_{(l)}$ for the attention heads and FFN neurons of the l-th layer, respectively. Each cluster contains k_i elements, where $i \in \{1, ..., K^A_{(l)}$ or $i \in \{1, ..., K^F_{(l)}\}$. The challenge is to determine the centroid of each cluster that represents the entire cluster and prune the other elements. Our approach is to select the element with the highest average absolute activations as the centroid, as larger-scale elements have a greater impact on the result. We also have attempted to select the cluster center as the centroid, which has the lowest average distance to other elements in the cluster, but this did not result in a significant performance improvement.

4. Experiments

Setup We implement our framework with PyTorch[1] using the HuggingFace Transformers library[2] and download the weights of the pre-trained models in Transformers. For clustering, we use the Scipy library[3]. We evaluate the effectiveness of our method for $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018) and GPT-2 (Radford et al., 2019), which are encoder-based and decoder-based PLMs respectively.

Baselines We compare the performance of our algorithm with both retraining-based structured pruning methods, such as DynaBERT (Hou et al., 2020) and EBERT (Liu et al., 2021), and other retraining-free approaches, including Mask-Tuning (Kwon et al., 2022) and KCM (Nova et al., 2023). Both Mask-Tuning and KCM are post-pruning methods that require a fully fine-tuned model for pruning. The difference between them is that Mask-Tuning requires labels of the task dataset, while it is not necessary for KCM. Since there is no KCM implementation that is publicly available, we implemented it ourselves.

Datasets For $\text{BERT}_{\text{BASE}}$, we conduct experiments on GLUE (Wang et al., 2018), $\text{SQuAD}_{1.1}$ (Rajpurkar et al., 2016), and $\text{SQuAD}_{2.0}$ (Rajpurkar et al., 2018) benchmarks under a wide range of

[1] https://github.com/pytorch
[2] https://github.com/huggingface/transformers
[3] https://github.com/scipy/scipy

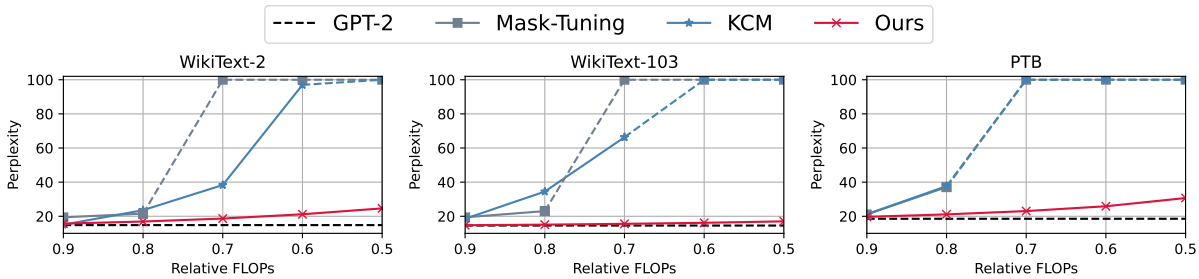

Figure 3: Performance of our method applied to GPT-2 on WikiText-2, WikiText-103 and PTB datasets against Mask-Tuning and KCM with different FLOPs constraints. The black dashed line indicates the performance of GPT-2. The blue and grey dashed lines represent a catastrophic performance degradation out-of-range, so we scale it to 100.

FLOPs constraints, which range from 0.9 to 0.1 with a step 0.1. Among the GLUE tasks, we select MNLI, QQP, QNLI, SST-2, STS-B, and MRPC tasks as the result of their stable performance. For GPT-2, we evaluate the performance after compression on the language modeling task with WikiText-2, WIkiText-103 (Merity et al., 2016), and PTB (Mikolov and Zweig, 2012) datasets. These datasets vary in size and contents. It worth noting that we reported the Spearman correlation for the STS-B task, accuracy for the other GLUE tasks, and perplexity for the language modeling task.

Hyperparameters For our proposed method, we randomly sample 3K raw data from the training set of the target task and set T as 200 to conduct module clustering. We set $\lambda = 5.0$ as the FFN modules account for more parameters and computation overhead, and $\delta = 0.001$ to obtain a more accurate threshold to apply clustering. When fine-tuning pruned BERT for the GLUE benchmark, we follow the setting of the BERT paper (Devlin et al., 2018), while for SQuAD datasets, the batch size is fixed to 12, and other settings keep the settings for GLUE. The settings of GPT-2 for language modeling task are fine-tuning 10, 8, 20 epochs with batch size 32 and learning rate 1e-4, 5e-4, 1e-4 on WikiText-2, WikiText-103, and PTB datasets respectively. For fine-tuning, we evaluate and save the best checkpoint at the end of each epoch. The results reported for Mask-Tuning and KCM are averaged across 10 runs with different random seeds.

4.1. Results on Classification Tasks

We first compare the accuracy drop of compressed $\text{BERT}_{\text{BASE}}$ generated by our approach on classification tasks against prior retraining-free structured pruning methods in Table 2 as previous works do. As we vary the FLOPs constraint from 90% to 20%, i.e. reducing 10% to 80% of the original FLOPs, it is clear that our method (red line) outperforms other competitors for the majority of tasks and FLOPs constraints, demonstrating the superiority of our

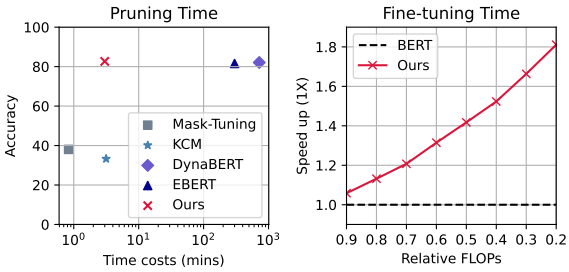

Figure 4: Comparison of time consumption to obtain a compressed fine-tuned model for MNLI task under 60% FLOPs constraint, which is divided into pruning time (**left**) and fine-tuning time (**right**).

algorithm. This distinction is particularly notable when the FLOPs constraint is below 30% for all tasks. In such cases, many heads and neurons (especially salient ones) are pruned to reconstruct the output for Mask-Tuning and KCM, resulting in significant performance degradation.

4.2. Results on Generative Tasks

Figure 3 compares the accuracy of language modeling tasks for GPT-2 compressed by Mask-Tuning, KCM, and our method. To the best of our knowledge, previous methods have not conducted experiments on generative tasks without retraining. Generative tasks are more sensitive to model changes, making reconstruction of the output harder compared to classification tasks. It shows that our method outperforms all competitors, especially with a strict FLOPs constraint of less than 80%. This demonstrates the applicability of our method to both encoder-based and decoder-based models, which is crucial as generative models have become increasingly popular in recent years.

4.3. Time Consumption for Pruning & Fine-tuning

Figure 4 demonstrates the time costs to obtain a compressed model fine-tuned on MNLI task between previous retraining-based, -free, and our

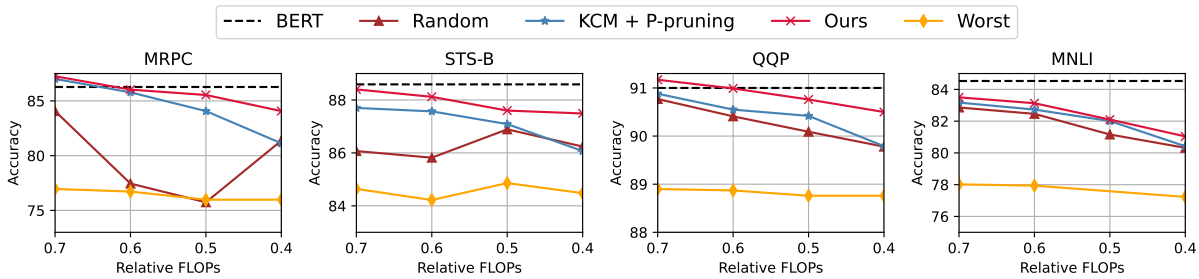

Figure 5: Performance of P-pruning derived by KCM and Dual-CP (Ours) on MRPC, STS-B, MNLI, and QQP tasks. 'Random' indicates randomly pruning attention heads and neurons, and 'Worst' denotes the condition with the most redundancy.

Method	MNLI	QQP	QNLI	SST-2	STS-B	MRPC	$\text{SQuAD}_{1.1}$	$\text{SQuAD}_{2.0}$
$\text{BERT}_{\text{BASE}}$	84.53	91.00	91.41	93.12	88.59	86.27	88.48	76.82
Random	81.52	90.24	88.49	90.83	85.82	77.45	84.79	72.18
+MC	**83.42** +1.90	90.51 +0.27	**89.75** +1.26	91.74 +0.91	86.70 +0.88	81.86 +4.41	85.37 +0.58	70.01 -2.17
+CS	83.13 +1.61	**90.99** +0.75	89.22 +0.77	**92.32** +1.49	**88.12** +2.30	**86.03** +8.58	**86.70** +1.91	**73.51** +1.33

Table 2: Performance improvements by applying the module clustering (MC) and centroid selection (CS) of Dual-CP respectively under 60% FLOPs constraint.

approaches under 60% FLOPs constraint with a GeForce RTX 3090 GPU. We divide this process into two procedures: fine-tuning and pruning, and record the time costs of both stages respectively.

On one hand, our method (shown in the left subfigure in Figure 4) achieves comparable performance to traditional retraining-based algorithms like EBERT and DynaBERT, but with significantly less time and computation overhead. Additionally, our method is as efficient as Mask-Tuning and KCM in conducting pruning, taking only several minutes, while achieving higher accuracy than them. In summary, our algorithm strikes the best balance between accuracy and pruning costs.

However, despite similar pruning costs compared to other retraining-free methods, our method is much more efficient when fine-tuning for the downstream task. It achieves a speedup of 1.8× and reduces FLOPs by 40%. It should be mentioned that this acceleration can be further improved by changing the checkpointing strategy, such as only saving the model after training is completed.

Above findings also reveals that, compared to other retraining-free structured pruning approaches, our method is particularly useful for a large compression rate especially generative tasks, which generates a more stable and accurate model more efficiently than any other retraining-free algorithms.

4.4. Ablation Studies

In this paper we have proposed two main contributions: (i) P-pruning, a novel compression framework, which utilizes the task-specific data to prune before fine-tuning and allows for cost reduction in fine-tuning; and (ii) Dual-CP, an effective pruning algorithm for identifying the important modules without relying on labels.

Is the pruning framework necessary? To evaluate the effectiveness of the pruning framework, we first perform an ablation study by replacing our Dual-CP with KCM and comparing their performance on various tasks. We choose KCM instead of Mask-Tuning which exhibits better performance shown in previous experiments as Figure 2, because the nature of our proposed paradigm cannot leverage the labels as described in section 3.2, and KCM is the only one retraining-free method with no need for labels. As Figure 5 shows, among all the tasks, our method outperforms KCM even if applying P-pruning at the same time, which demonstrates the superiority of our method.

To evaluate the redundancy of the pruned modules in our method, we perform a 'Worst' condition, in which case we preserve all the similar heads and neurons in a large-size cluster and prune other clusters in Section 3.3 to keep redundancy. In contrast, we additionally perform a 'Random' condition which randomly prunes heads and neurons. As Figure 5 elaborates, the 'Worst' condition shows the lowest accuracy among all tasks even compared to random pruning, which yields unstable performance under different FLOPs constraints. This inferior performance also proves that the pruned similar heads and neurons are indeed redundant, and the superiority of our approach is not rooted in the fine-tuning process after pruning, but in the criteria we choose to prune.

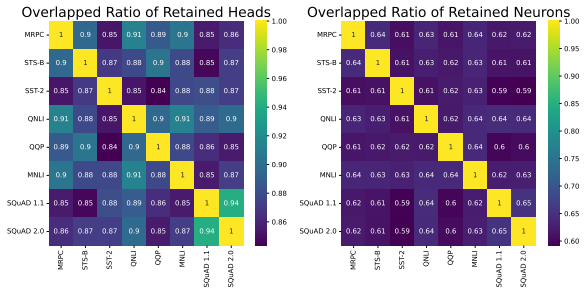

Figure 6: The proportion of shared heads and neurons across pruned models for different tasks.

How important are the procedures in Dual-CP? We also perform another ablation experiment under 60% FLOPs constraint to evaluate the validity of the procedures in Dual-CP, e.g. module clustering (MC) and centroid selection (CS). As Table 2 shows, +MC means only applying clustering to the heads and neurons but randomly selecting centroids, and +CS is the same as the results of 'Ours' in the above experiments. We take the performance derived by random pruning as the benchmark. It is observed that both module clustering and centroid selection contribute to more accurate salient module selection and higher accuracy, especially for the MRPC and STS-B task.

Is the Pruned Model Task-specific? The last question is whether is it really necessary to use the task datasets to conduct clustering, and how distinct are the pruned models derived by different task datasets. Therefore, we compare the ratio of shared attention heads and neurons between the compressed models derived from various tasks as elaborated in Figure 6. It is obvious that the pruned models for different tasks exhibit diversity, especially for neurons, which demonstrates that our method is indeed task-specific. It is also noticed that the model derived for SQuAD 1.1 and SQuAD 2.0 have the highest similarity as the result of the same task and partly shared dataset.

5. Conclusion

In this work, we propose P-pruning, a novel compression framework that task-specifically prunes the pre-trained language model before fine-tuning, and therefore reduces the fine-tuning costs. Furthermore, we propose a pruning algorithm to conduct P-pruning, which consists of (i) module clustering that clusters similar attention heads and intermediate neurons according to the activations derived by the task dataset, and (ii) centroid selection that pick the heads and neurons with the maximum averaged magnitude of activations in each cluster and prune others in the same cluster. The experimental results demonstrate that our method achieves the highest accuracy for both BERT and GPT-2 on classification and generative tasks. Furthermore, our method shows the best trade-off between accuracy and pruning time, and is also demonstrated to be the most efficient for fine-tuning.

6. Limitations & Future Work

Although this study brings significant time reduction for retraining-free pruning while maintaining satisfactory performance, it still has three important potential limitations.

1) Firstly, the time consumption of Dual-CP may increase significantly as the pre-trained model size increases. The computational overhead of Dual-CP is mainly determined by module clustering, in which the cosine similarity is calculated for each two neurons or heads. Therefore, the time cost is highly dependent on the number of internal dimensions, and may be inefficient for large language models, and we hope to fix it in future work.

2) Secondly, the module clustering may be inefficient for long sequence lengths. The similarity calculation is conducted for the hidden inputs, which will be fairly low universally as the sampled sequence length extends, therefore unable to utilize more calibration instances to guarantee the validity of pruning.

3) Finally, the instances of the calibration dataset are randomly sampled from the task dataset, which may miss informative sentences. However, as our method is data-driven, a more well-designed data selection method should be proposed to screen out the most informative sentences or tokens.

7. Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 62106140), STCSM (No. 21511101100, No. 22DZ2229005), and 111 plan (No. BP0719010).

8. Bibliographical References

Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. 2020. wav2vec 2.0: A framework for self-supervised learning of speech representations. Advances in neural information processing systems, 33:12449–12460.

Samuel Cahyawijaya. 2021. Greenformers: Improving computation and memory efficiency in transformer models via low-rank approximation. arXiv preprint arXiv:2108.10808.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In International Conference on Learning Representations.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323.

Manish Gupta and Puneet Agrawal. 2022. Compression of deep learning models for text: A survey. ACM Transactions on Knowledge Discovery from Data (TKDD), 16(4):1–55.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (GELUs). arXiv preprint arXiv:1606.08415.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. DynaBERT: Dynamic BERT with adaptive width and depth. Advances in Neural Information Processing Systems, 33:9782–9793.

Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. 2021. HuBERT: Self-supervised speech representation learning by masked prediction of hidden units. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 29:3451–3460.

Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. arXiv preprint arXiv:2207.00112.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling BERT for natural language understanding. arXiv preprint arXiv:1909.10351.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451.

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.

Woosuk Kwon, Sehoon Kim, Michael W Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. 2022. A fast post-training pruning framework for transformers. Advances in Neural Information Processing Systems, 35:24101–24116.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. 2021. Block pruning for faster transformers. arXiv preprint arXiv:2109.04838.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.

Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. 2021. EBERT: Efficient BERT inference with dynamic structured pruning. In Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4814–4823.

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

Tomas Mikolov and Geoffrey Zweig. 2012. Context dependent recurrent neural network language model. In 2012 IEEE Spoken Language Technology Workshop (SLT), pages 234–239. IEEE.

James O' Neill. 2020. An overview of neural network compression. arXiv preprint arXiv:2006.03669.

Azade Nova, Hanjun Dai, and Dale Schuurmans. 2023. Gradient-free structured pruning with unlabeled data. arXiv preprint arXiv:2303.04185.

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.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. arXiv preprint arXiv:1806.03822.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2023. On the effect of dropping layers of pre-trained transformer models. Computer Speech & Language, 77:101429.

Hassan Sajjad, Nadir Durrani, and Fahim Dalvi. 2022. Neuron-level interpretation of deep nlp models: A survey. Transactions of the Association for Computational Linguistics, 10:1285–1303.

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

Victor Sanh, Thomas Wolf, and Alexander Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. Advances in Neural Information Processing Systems, 33:20378–20389.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-BERT: Hessian based ultra low precision quantization of BERT. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 8815–8821.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. arXiv preprint arXiv:1908.09355.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. arXiv preprint arXiv:2004.02984.

Chaofan Tao, Lu Hou, Haoli Bai, Jiansheng Wei, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. 2023. Structured pruning for efficient generative pre-trained language models. In Findings of the Association for Computational Linguistics: ACL 2023, pages 10880–10895.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In International conference on machine learning, pages 10347–10357. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. Advances in neural information processing systems, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768.

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.

Hao Yu and Jianxin Wu. 2023. Compressing transformers: Features are low-rank, but weights are not!

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit BERT. In 2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS), pages 36–39. IEEE.

Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, et al. 2023. A comprehensive survey on pretrained foundation models: A history from BERT to ChatGPT. arXiv preprint arXiv:2302.09419.