PGB: One-Shot Pruning for BERT via Weight Grouping and Permutation

HYEMIN LIM, Inha University, South Korea
JAEYEON LEE, Inha University, South Korea
DONG-WAN CHOI*, Inha University, South Korea

Large pretrained language models such as BERT suffer from slow inference and high memory usage, due to their huge size. Recent approaches to compressing BERT rely on iterative pruning and knowledge distillation, which, however, are often too complicated and computationally intensive. This paper proposes a novel semi-structured one-shot pruning method for BERT, called *Permutation and Grouping for BERT* (PGB), which achieves high compression efficiency and sparsity while preserving accuracy. To this end, PGB identifies important groups of individual weights by permutation and prunes all other weights as a structure in both multi-head attention and feed-forward layers. Furthermore, if no important group is formed in a particular layer, PGB drops the entire layer to produce an even more compact model. Our experimental results on $\text{BERT}_{\text{BASE}}$ demonstrate that PGB outperforms the state-of-the-art structured pruning methods in terms of computational cost and accuracy preservation.

JAIR Associate Editor: Scott Sanner

JAIR Reference Format:
Hyemin Lim, Jaeyeon Lee, and Dong-Wan Choi. 2026. PGB: One-Shot Pruning for BERT via Weight Grouping and Permutation. *Journal of Artificial Intelligence Research* 85, Article 24 (March 2026), 20 pages. DOI: 10.1613/jair.1.20723

1 Introduction

Transformer-based models (Vaswani et al. 2017) including BERT (Devlin et al. 2019), RoBERTa (Y. Liu et al. 2019) and GPT-3 (Brown et al. 2020) have achieved great performance on natural language processing (NLP) tasks. However, all these models suffer from a large number of parameters, which often limits their applications due to high computational cost and memory usage. To overcome this limitation, extensive research has been conducted to reduce the model size of transformer architectures.

Recent works on compressing BERT adopt two primary strategies, pruning (Han et al. 2015) and knowledge distillation (KD) (Hinton et al. 2015). Pruning can further be classified into two categories based on how many times pruning and recovery processes are performed: one-shot pruning (Lee et al. 2018) and iterative pruning (T. Chen et al. 2020; Luo et al. 2017). Even though one-shot pruning is simple and computationally efficient as it conducts only one pruning phase, it tends to be less effective to maintain high accuracy. Therefore, the dominant approach for BERT is taking iterative steps of pruning and recovery while training with original dataset.

Furthermore, recent pruning methods (Hou et al. 2020; Lagunas et al. 2021; Xia et al. 2022) attempt to overcome the low pruning performance with the help of KD, which has been successful in maintaining high performance in BERT (Jiao et al. 2020; Sanh, Debut, et al. 2019). However, the distillation process can be even more time-consuming

*Corresponding Author.

Authors' Contact Information: Hyemin Lim, hyemin8670@naver.com, Inha University, Incheon, South Korea; Jaeyeon Lee, ORCID: 0009-0004-8391-6459, dlwodus159@naver.com, Inha University, Incheon, South Korea; Dong-Wan Choi, ORCID: 0000-0003-3122-7518, dchoi@inha.ac.kr, Inha University, Incheon, South Korea.

DOI: 10.1613/jair.1.20723

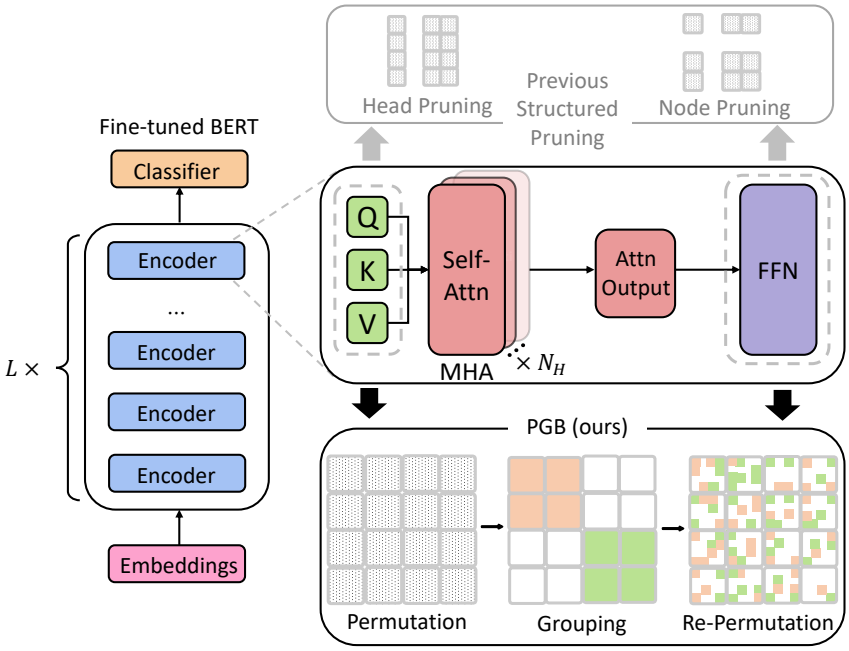

Fig. 1. Permutation and Grouping for BERT (PGB), where weight matrices are grouped and all individual weights not belonging to groups are pruned.

than iterative pruning, and it is often too complicated to identify what aspects of the teacher model should be matched to the student model, particularly in the BERT architecture.

In this paper, we design a novel one-shot pruning method for a task-specific BERT, aiming to achieve both high compression efficiency and high accuracy. Due to the lack of recovery chances, it is quite challenging for one-shot pruning to maintain high performance. Our proposal is to devise a semi-structured pruning method, called *Permutation and Grouping for BERT* (PGB), which combines the benefits of both unstructured and structured pruning. Thus, PGB effectively reduces the model size without sacrificing so much of the accuracy as in unstructured pruning, while ensuring computational efficiency as in structured pruning.

Our PGB method is illustrated in Figure 1. Basically, we apply a group-based pruning scheme (Su et al. 2020; Zhao and Luk 2019) to each structure of BERT, including multi-head attention (MHA) and feed-forward network (FFN). More specifically, PGB constructs important groups of individual weights for each layer to be preserved and prune all other weights that are not in such a group. Although training with group-based architectures from scratch has been studied with a different purpose (Chelombiev et al. 2021; Park et al. 2020), pruning by weight grouping has never been tackled in the context of transformer-based models. A major challenge is how to preserve the original performance of a given task-specific BERT after pruning unimportant weights. To this end, PGB performs an optimal permutation procedure so that more important weights are clustered as a structure, and adaptively determines the number of important groups for each layer of either MHA or FFN, occasionally dropping the entire layer. Finally, its re-permutation process safely rearranges each weight to its original position.

A thorough experimental study is conducted by applying our PGB method to $\text{BERT}_{\text{BASE}}$ (Devlin et al. 2019) on the GLUE (A. Wang et al. 2019) and SQuAD (Rajpurkar, J. Zhang, et al. 2016) benchmarks. Our experimental

results show that PGB outperforms the state-of-the-art (SOTA) pruning methods in terms of both efficiency and accuracy.

2 Background and Related Works

Pretrained language models (Brown et al. 2020; Devlin et al. 2019; Y. Liu et al. 2019) are mostly based on the transformer architecture (Vaswani et al. 2017) due to their effectiveness in various NLP tasks. This section first formally describes the typical transformer architecture and then discusses representative techniques for compressing large language models, namely distillation and pruning.

2.1 Transformer Architecture

The typical transformer architecture consists of encoder and decoder layers, each of which commonly contains two main components: multi-head attention (MHA) and feed-forward network (FFN). In an MHA layer, there are N_H self-attention heads, and each head $h \in [1, N_H]$ involves the following weight matrices, $W_h^Q, W_h^K, W_h^V, W_h^O \in \mathbb{R}^{d \times \frac{d}{N_H}}$. Then, the final output of the MHA layer is computed as follows:

$$MHA(X) = \sum_{h=1}^{N_H} Attn_h(X),$$

where $Attn_h(X)$ is the output of the standard self-attention unit.

The output of MHA is then fed into the corresponding FFN layer, which consists of weight matrices $W^{(1)} \in \mathbb{R}^{d \times d_{ffn}}$, $W^{(2)} \in \mathbb{R}^{d_{ffn} \times d}$, $b^{(1)} \in \mathbb{R}^{d_{ffn}}$ and $b^{(2)} \in \mathbb{R}^d$, where d_{ffn} (usually $4 \times d$) represents the dimension of the intermediate hidden features. The output of the FFN layer can be computed as follows:

$$FFN(A) = \sum_{i=1}^{d_{ffn}} GeLU(AW_{:,i}^{(1)} + b^{(1)})W_{i,:}^{(2)} + b^{(2)},$$

where $A = MHA(X)$.

In transformer-based models, this same structure is repeatedly defined across multiples layers (e.g., 12 layers in $\text{BERT}_{\text{BASE}}$ (Devlin et al. 2019)) and each layer has another multiple heads (e.g., 12 heads in $\text{BERT}_{\text{BASE}}$ (Devlin et al. 2019)). Consequently, they have numerous trainable parameters, which motivates the NLP community to develop various compression methods for these models.

2.2 Distillation

Knowledge distillation (KD) (Hinton et al. 2015) is a compression technique that trains a lightweight student model to mimic the soft output of a large teacher model, leading to competitive performance on downstream tasks. There are some studies where KD methods have been applied to transformer-based models. For example, DistilBERT (Sanh, Debut, et al. 2019) transfers knowledge to a student model of a fixed-size through a pre-training phase and an optional fine-tuning process. MiniLM (W. Wang et al. 2020) deeply describes the self-attention information of a task-agnostic teacher model by providing a detailed analysis of the self-attention matrices. Both TinyBERT (Jiao et al. 2020) and MobileBERT (Sun et al. 2020) transfer knowledge during pretraining using a layer-by-layer strategy. TinyBERT (Jiao et al. 2020) additionally performs distillation during fine-tuning.

Although KD-based methods are shown to be effective at preserving high accuracy, training a student model can be time-consuming; as reported by CoFi (Xia et al. 2022), TinyBERT (Jiao et al. 2020) requires 350 hours and CoFi (Xia et al. 2022) requires 20 hours for compression. Furthermore, it is not trivial to effectively distill the knowledge from a multi-layer teacher model with self-attention information to a student model with fewer layers, which involves the problem of layer selection and different loss functions.

2.3 Pruning

Pruning (Han et al. 2015) is another popular compression scheme that removes unimportant weights or structures from the target neural network. Following a massive volume of pruning techniques on convolutional neural networks (CNNs), pruning for transformer family has also been studied, falling into either unstructured or structured pruning.

Unstructured pruning. In unstructured pruning, we remove individual weights that are not important, often aiming to reduce the memory storage itself for the target model while maintaining performance, such as the methods based on *lottery ticket hypothesis* (T. Chen et al. 2020) and *movement pruning* (Sanh, Wolf, et al. 2020). However, in these unstructured pruning methods, it is difficult and complicated to make satisfactory speedup at inference time, often requiring a special hardware.

Structured pruning. Structured pruning is a simpler and more efficient way to reduce the model size, where we eliminate some structures of parameters that are considered less significant. In the case of transformer architectures, we can remove redundant attention heads (Michel et al. 2019; Voita et al. 2019; Z. Zhang et al. 2021), entire layers of either MHA or FFN (Fan et al. 2019; Sajjad et al. 2020), or neurons along with all their connecting weights of FFNs (X. Chen et al. 2020). However, such a coarse-grained pruning scheme inevitably suffers from severe drop in accuracy. Consequently, recent studies (Hou et al. 2020; Lagunas et al. 2021; Xia et al. 2022) have explored the combination of pruning with KD to further enhance the performance of compressed networks. Despite higher performance of these combined approaches, they sacrifice efficiency and simplicity in the compression process itself, as KD typically takes lengthy training times and involves complicated matching procedures.

Semi-structured pruning. This paper proposes a semi-structured pruning method for BERT in order to achieve a good balance between efficiency and accuracy. To our best knowledge, *block movement pruning* (BMP) (Lagunas et al. 2021) is the only one that can be categorized into semi-structured pruning for transformer family, which proposes a block-based pruning approach for each weight matrix of MHA and FFN layers.

Group-based pruning. Our method is inspired by a grouping strategy, which has been frequently employed to reduce the computational complexity of CNNs. In this approach, a set of filters or channels are grouped together, and the objective is to minimize connectivity and computation between these groups (Ioannou et al. 2017; Su et al. 2020; Xie et al. 2018; X. Zhang et al. 2018; Zhao and Luk 2019). These grouped architectures are also incorporated into transformer-based models by a few recent works (Chelombiev et al. 2021; Park et al. 2020). However, they focus on designing more efficient architectures to be trained from scratch, not for compressing a trained task-specific model. This work is the first study on group-based pruning on BERT, aiming to compress the task-specific BERT while maintaining its original accuracy.

3 Method

This section presents our one-shot semi-structured pruning method, called *Permutation and Grouping for BERT* (PGB), which applies a grouping procedure to weight matrices of each structure of a given task-specific BERT, as illustrated in Figure 2.

3.1 Overall Process

Unlike existing grouped transformer architectures (Chelombiev et al. 2021; Park et al. 2020) that are designed to be trained from scratch, our goal is to find the adaptive grouping for minimizing information loss in the original task-specific BERT. Therefore, as shown in Figure 2, our PGB method performs the grouped pruning process for each individual weight matrix, rather than partitioning every part of the model into the fixed number of groups

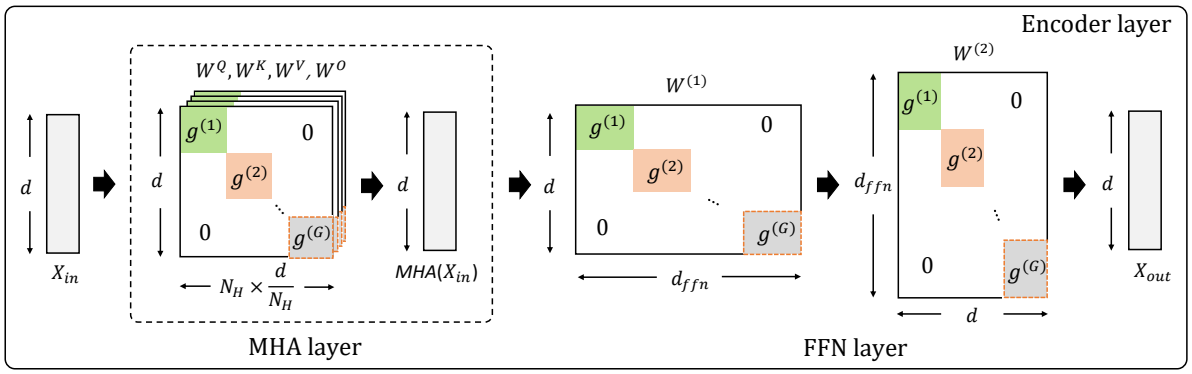

Fig. 2. The grouping procedure of PGB for weight matrices in BERT, which applies to both MHA and FFN layers

as in grouped transformers. In our pruned BERT architectures, the resulting number of groups for each layer can be different, and even a particular layer can be dropped as a whole when no important group is formed in the layer. After the pruning process, we apply the re-permutation step to every pruned weight matrix $\hat{W}$ to restore the original positions of the weights.

Problem formulation. We first formulate our problem of grouped pruning on BERT. Consider a task-specific BERT $[\Theta_1, ..., \Theta_L]$ with L layers, where Θ_i consists of weight matrices W^Q, W^K, W^V, W^O in its MHA sub-layers, and $W^{(1)}, W^{(2)}$ in its FFN sub-layers. Given a target compression rate γ and the task-specific dataset $\mathcal{D}$, our goal is to find the pruned architecture $[\widehat{\Theta}_1, ..., \widehat{\Theta}_L]$ such that:

$$\begin{array}{ll} \min & \sum_{i=1}^{L} ||\mathcal{F}(\mathcal{D}; \Theta_i) - \mathcal{F}(\mathcal{D}; \widehat{\Theta}_i)|| \\ \text{s.t.} & \sum_{i=1}^{L} C(\widehat{\Theta}_i) \approx \gamma \cdot \sum_{i=1}^{L} C(\Theta_i), \end{array} \tag{1}$$

where $\mathcal{F}(\mathcal{D}; \Theta)$ denotes the output of the model parameterized by Θ for the input data $\mathcal{D}$ and $C(\cdot)$ is the number of parameters (or FLOPs). As mentioned above, each $\widehat{\Theta}_i$ can have different number of groups unlike the equally grouped transformer architectures (Chelombiev et al. 2021; Park et al. 2020).

PGB outline. Algorithm 1 presents how our PGB method finds the group-based pruned architecture for a given task-specific BERT. The algorithm consists of two main phases, namely MHA pruning and FFN pruning. In accordance with previous studies (Lagunas et al. 2021; Xia et al. 2022), our approach focuses on pruning less weights in the MHA sub-layers, compared to those of the FFN sub-layers. Furthermore, since it is crucial for one-shot pruning to deal with a more challenging scenario of pruning a larger number of weights in either MHA or FFN sub-layers, we attempt to minimize the information loss in MHA sub-layers than in FFN sub-layers. To this end, we first take a more conservative approach for pruning weights in MHA sub-layers (Lines 2–5) by trying to find an optimal grouped architecture for each weight matrix W based on its importance, which is performed by the Group-Weight-Pruning subroutine (refer to Algorithm 2). Then, we proceed a similar yet more aggressive pruning procedure for the FFN sub-layers (Lines 8–12), where multiple FFN layers that are least important can entirely be dropped in order to meet the remaining budget C. Therefore, instead of taking a sequential process across layers, we prioritize the more important layers over the less important ones within the remaining budget C.

Algorithm 1: PGB-Compression

Input: $[\Theta_1, ..., \Theta_L] \triangleq$ a given model with L layers,
$\gamma \triangleq$ the target compression rate
Output: $[\widehat{\Theta}_1, ..., \widehat{\Theta}_L] \triangleq$ the pruned model

$C \leftarrow \gamma \cdot \sum_{i=1}^{L} C(\Theta_i)$
/* MHA Pruning */
for *each layer* $i \in [1, L]$ **do**
  for *each weight matrix* W *in MHA of* Θ_i **do**
    $\widehat{W} \leftarrow$ Group-Weight-Pruning(W)
    $\widehat{\Theta}_i$.Append($\widehat{W}$)
$C \leftarrow C - \sum_{i=1}^{L} C(\widehat{\Theta}_i)$
/* FFN Pruning */
while $C > 0$ **do**
  $j \leftarrow$ pick the unused layer with the largest importance score for FFN
  for *each weight matrix* W *in FFN of* Θ_j **do**
    $\widehat{W} \leftarrow$ Group-Weight-Pruning(W)
    $\widehat{\Theta}_j$.Append($\widehat{W}$)
    $C \leftarrow C - C(\widehat{W})$
return $[\widehat{\Theta}_1, ..., \widehat{\Theta}_L]$

3.2 Grouped Weight Pruning

In order to find the optimal grouping for each weight matrix of BERT, we adapt the technique of *group convolution* pruning (Su et al. 2020; Zhao and Luk 2019), which is originally intended to prune filters in CNNs, not individual weights as in our problem setting.

Although originally proposed for CNN filters, we adapt the pruning and permutation framework of (Zhao and Luk 2019) to the case of BERT weight matrices. Their algorithm, reproduced in AppendixA.6 (Algorithm 4), employs a heuristic alternating procedure where row and column permutations are updated in turn. While (Zhao and Luk 2019) do not provide a formal guarantee of global optimality, this procedure can be interpreted as a coordinate-wise search that iteratively improves the objective. In our setting, we adopt this heuristic and further demonstrate empirically that it yields near-optimal groupings in practice. The process of our grouped weight pruning is presented in Algorithm 2. For each weight matrix $W \in \mathbb{R}^{M \times N}$ in BERT, our method first adaptively determines the number G of groups by measuring the degree of importance of W, and drop the entire matrix if the corresponding importance measure is not sufficiently high (Lines 1–3). Then, we permute the matrix to form groups of important weights into block diagonal matrices $g^{(i)}$'s (Line 6). Once the permuted matrix $\widetilde{W}$ is obtained, we extract only the top-left corner block of $\widetilde{W}$ (i.e., $\widetilde{W}[1 : \frac{M}{G}, 1 : \frac{N}{G}]$) to form a group matrix $g^{(i)}$ (Line 7), and discard all the weights in the region $\widetilde{W}[1 : \frac{M}{G}, \ ]$ and $\widetilde{W}[\ , 1 : \frac{N}{G}]$ from $\widetilde{W}$ (Line 8). The final $\widehat{W}$ would be represented as follows:

$$\widehat{W} = \begin{vmatrix} g^{(1)} & 0 & \cdots & 0 \\ 0 & g^{(2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & g^{(G)} \end{vmatrix}. \tag{2}$$

Algorithm 2: GROUPED-WEIGHT-PRUNING

Input: $W \triangleq$ a weight matrix of $M \times N$
Output: $\widehat{W} \triangleq$ the pruned weight matrix

```
1  G ← Determine-Group-Numbers(W)
2  if G = 0 then
3      return null
4  W̃ ← W; Ŵ ← null
5  for each group i ∈ [1, G] do
6      W̃ ← Permutation(W̃)
7      g^(i) ← W̃[1 : M/G, 1 : N/G]
8      W̃ ← W̃[M/G :, N/G :]
9      Ŵ.Append-Diagonal-Block(g^(i))
10 return Ŵ
```

Finding the optimal permutation. The permutation procedure (Line 6 in Algorithm 2) returns the optimal arrangement $\widetilde{W}$ of individual weights within the given matrix $W \in \mathbb{R}^{M \times N}$. The objective is to cluster more important weights together, forming a group located at the top-left corner block of $\widetilde{W}$. To this end, we determine the optimal pair of permutation vectors π_r and π_c for the rows and columns of W, respectively, that are used to rearrange the weights of W, resulting in the formation of $\widetilde{W}$ as follows:

$$\begin{aligned} \max_{\pi_r, \pi_c} \quad & \mathcal{I}(\widetilde{W}[1 : \tfrac{M}{G}, 1 : \tfrac{N}{G}]) \\ \text{s.t.} \quad & \widetilde{W} = W_{\pi_r, \pi_c}, \end{aligned} \tag{3}$$

where $\mathcal{I}(\cdot)$ is the total importance score of a given weight matrix and W_{π_r, π_c} is the resulting matrix when permuting the rows and columns of W using π_r and π_c, respectively. We calculate the importance scores of weights using the second-order information (Hassibi et al. 1993; Molchanov et al. 2019; Singh and Alistarh 2020), which allows us to quantify relative significance of the weights. The following example shows such optimal permutations for a matrix $W \in \mathbb{R}^{4 \times 4}$ and $G = 2$, assuming that each number in the matrix indicates the importance score of the corresponding weight:

$$\begin{vmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{vmatrix} \xrightarrow[\pi_c=[1,4,2,3]]{\pi_r=[1,3,2,4]} \begin{vmatrix} 1 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{vmatrix}.$$

Heuristic solution for permutation. Since weight matrices in BERT are typically high-dimensional, we employ an efficient heuristic algorithm (Zhao and Luk 2019) that finds sub-optimal permutation vectors. This algorithm alternatively sorts rows or columns based on the summation of importance scores corresponding to the weights of either row vectors within the region $\widetilde{W}[1 : \frac{M}{G}, \ ]$ or column vectors within the region $\widetilde{W}[\ , 1 : \frac{N}{G}]$. Since each sorting process for the rows or the columns changes the arrangement of the corresponding columns or rows, respectively, we repeat this pairwise sorting process a few times (e.g., 6 times in our experiments using $\text{BERT}_{\text{BASE}}$).

Adaptive group numbers. The key property of our method is the adaptive determination of the number G of groups (Line 1 in Algorithm 2), based on the importance of weights within $W \in \mathbb{R}^{M \times N}$. Basically, as the number of important weights in W increases, we decrease G and prune a smaller number of weights, whereas if W

contains fewer important weights, we increase G to prune a greater number of weights. To this end, we devise the following strategy to adjust G based on the count of weights whose important scores exceed a specified threshold τ:

(1) Determine the count n_τ of weights in W with importance scores higher than τ.
(2) If $\frac{M \times N}{n_\tau} > G_{max}$, then prune the entire W.
(3) Otherwise, set G to a value less than $\frac{M \times N}{n_\tau}$.

The term $\frac{M \times N}{n_\tau}$ is derived from the fact that the number of parameters of a G-grouped matrix $\widehat{W}$ is equal to that of W divided by G, i.e., $\frac{M \times N}{G}$. Therefore, to ideally cover all n_τ weights, $\widehat{W}$ should have at most $\frac{M \times N}{n_\tau}$ groups. Also, we introduce the hyperparameter G_{max} to prevent the formation of excessive groups for non-critical weight matrices (e.g., $G_{max} = 6$ in our experiments).

3.3 Re-Permutation

PGB finally performs the re-permutation procedure on every pruned weight matrix $\widehat{W}$ in all the layers of the model to identify the positions of the weights that correspond to the original model. This yields a re-permuted weight matrix $\widehat{W}^*$, wherein each weight is returned to its original positions. In this process, we utilize the permutation vectors π_r and π_c that have been stored, and proceed the following operation:

$$\widehat{W} \xrightarrow[argsort(\pi_c)]{argsort(\pi_r)} \widehat{W}^*,$$

where $argsort(\pi)$ returns the corresponding re-permutation vectors that rearrange the shuffled weights back to their original positions. Note that the final $\widehat{W}^*$ after re-permutation is in the same form resulting from fine-grained unstructured pruning, but actual computation at inference time is efficiently performed only with each $g^{(i)} \subseteq \widehat{W}$ as in grouped transformer architectures (Chelombiev et al. 2021; Park et al. 2020).

Weight compensation. To further restore the performance of the original task-specific BERT model, we update each unpruned weight in every $\widehat{W}^*$ by minimizing the following reconstruction error:

$$\min \left\| \mathcal{F}(X; \widehat{W}^*) - \mathcal{F}(X; W) \right\|_2^2, \tag{4}$$

where $\mathcal{F}(X; W)$ denotes the outputs for a sample dataset X.

Re-Finetuning. After all these steps, we perform re-finetuning in the same way as in the original BERT (Devlin et al. 2019) to recover the performance that is lost due to the pruning process.

4 Cost Analysis for Inference with PGB

As mentioned above, we make inference using only G groups of each pruned weight matrix $\widehat{W} \in \mathbb{R}^{M \times N}$ for each linear operation $X\widehat{W}^*$, where X is the S-length input sequence. By using this inference module, we ensure G times faster efficiency by reducing the previous time cost $S \cdot M \cdot N$ to $S \cdot G \cdot \frac{M}{G} \cdot \frac{N}{G}$.
Algorithm 3 provides a detailed outline of the linear operation process used at inference time with a pruned model resulting from the Group-Weight-Pruning procedure in Algorithm 2. It takes the input sequence $X \in \mathbb{R}^{S \times M}$, the grouped weight matrix $\widehat{W}$ obtained after PGB pruning, and the row and column permutation vectors π_r and π_c that have been determined during the pruning process. At this point, we take only the groups of remaining weights, $g^{(1)}, ..., g^{(G)}$ where each $g^{(i)} \in \mathbb{R}^{\frac{M}{G} \times \frac{N}{G}}$. Our inference process first starts with permuting the input X using π_c at each linear operation (Line 2). Subsequently, for the pruned weight matrix $\widehat{W}$ and the permuted matrix $\widetilde{X}$, we obtain the output $o^{(j)}$ for each group with its corresponding weight matrix $g^{(j)}$ and permuted input matrix

Algorithm 3: PGB-Linear

Input: $X \triangleq$ input X of $S \times M$,
$\widehat{W} \triangleq$ a pruned weight matrix with G diagonal groups, where each group is of $\frac{M}{G} \times \frac{N}{G}$,
$\pi_r, \pi_c \triangleq$ permutation vectors for the rows and columns of each weight matrix W

1 **for** *each linear operation* $X\widehat{W}^*$ *s.t.* $\widehat{W}^* \xleftarrow[argsort(\pi_c)]{argsort(\pi_r)} \widehat{W}$ **do**
2 $\quad \widetilde{X} \xleftarrow[\pi_c]{} X$
3 $\quad$ **for** *each group* $j \in [1, G]$ **do**
4 $\quad\quad g^{(j)} \leftarrow \widehat{W}[\frac{M}{G}(j-1) : \frac{M}{G} j\,,\, \frac{N}{G}(j-1) : \frac{N}{G} j]$
5 $\quad\quad \widetilde{X}^{(j)} \leftarrow \widetilde{X}[1 : S\,,\ \frac{M}{G}(j-1) : \frac{M}{G} j]$
6 $\quad\quad o^{(j)} \leftarrow \widetilde{X}^{(j)} g^{(j)}$
7 $\quad O \leftarrow \textsc{Concat}[o^{(1)}, o^{(2)}, \ldots, o^{(G)}]$
8 $\quad O^* \xleftarrow{argsort(\pi_r)} O$
9 $\quad$ **return** O^*

$\widetilde{X}^{(j)}$ (Lines 3–6). Once the outputs $o^{(1)}, ..., o^{(G)}$ for all groups are obtained, we concatenate them to form the output O, and permute the output O with respect to $argsort(\pi_r)$ to obtain the final desired output $O^* \in \mathbb{R}^{S \times N}$ (Lines 7–8), where $argsort(\pi_r)$ returns the permutation vector of indices that rearrange the output values back to their original positions.
It should be noted that while Algorithm 3 conceptually illustrates row and column permutations at each layer for clarity, these permutations are not explicitly applied to activations during inference. Instead, they can be folded into the weight matrices or canceled across consecutive layers, such that the runtime computation reduces to only group-wise matrix multiplications. In theory, this yields a $1/G$ reduction in FLOPs compared to the original dense linear layers used in BERT. In practice, the actual latency improvement may be somewhat smaller than this ideal factor due to memory access patterns and the lower efficiency of executing many smaller matrix multiplications compared to a single large operation. Nevertheless, PGB-Linear consistently achieves faster inference than the dense BERT layers and, importantly, does not incur any additional activation-level permutation overhead.

5 Experiments

5.1 Environment

Datasets. We conduct our experiments using two benchmark datasets, SQuAD (Rajpurkar, Jia, et al. 2018; Rajpurkar, J. Zhang, et al. 2016) and seven tasks (QNLI, QQP, SST-2, CoLA, STS-B, MRPC, and RTE) in GLUE (A. Wang et al. 2019) on $\text{BERT}_{\text{BASE}}$ (Devlin et al. 2019). In our experiments, we use 2K samples from the training data for each benchmark dataset. Also, we perform re-finetuning on the pruned model using the training data of each NLP downstream task and evaluate on the corresponding dev dataset for each task. For detailed information on the benchmarks, please refer to A.3.

Compared methods. In order to evaluate the performance of PGB, we conduct comparative experiments with the state-of-the-art structured pruning methods for BERT, including EBERT (Z. Liu et al. 2021), DynaBERT (Hou et al. 2020), and CoFi (Xia et al. 2022). In order to examine their pruning performance, we train each method without using distillation and data augmentation, both of which can commonly be applied to each pruning method.

Table 1. Performance comparison with structured pruning methods using 50% and 88% pruning rates on $\text{BERT}_{\text{BASE}}$, where N.A. indicates that the respective method do not achieve the specified level of sparsity.

Pruning Ratio	Method	# Param	QNLI Acc.	QQP Acc.	SST-2 Acc.	CoLA Mcc.	STS-B Spearman	MRPC Acc.	RTE Acc.	$\text{SQuAD}_{1.1}$ EM/F1	$\text{SQuAD}_{2.0}$ EM/F1
0%	$\text{BERT}_{\text{BASE}}$	85M	91.4	91.5	93.2	58.9	89.2	86.3	66.8	80.8/88.3	70.9/74.2
50%	EBERT	42M	89.9	90.6	90.8	N.A.	87.1	72.8	52.7	76.7/85.2	68.6/72.5
	DynaBERT	42.8M	88.3	90.7	91.6	51.2	86.4	77.8	63.5	-	-
	CoFi	42.3M	88.8	90.6	90.1	53.6	88.0	83.5	56.7	-	-
	PGB (Ours)	42.5M	**90.3**	**91.1**	**92.3**	**54.9**	**88.8**	**84.3**	**64.6**	**78.0/86.8**	**69.6/73.5**
88%	EBERT	10.9M	81.8	88.1	87.5	N.A.	84.9	N.A.	49.5	N.A.	N.A.
	DynaBERT	10.7M	83.5	86.8	88.5	18.7	82.9	72.6	53.1	-	-
	CoFi	10.4M	84.7	89.8	89.0	32.1	85.1	75.3	52.4	-	-
	PGB (Ours)	10.2M	**86.4**	**90.1**	**89.6**	**39.5**	**85.3**	**78.2**	**54.3**	**71.5/81.2**	**65.9/69.7**

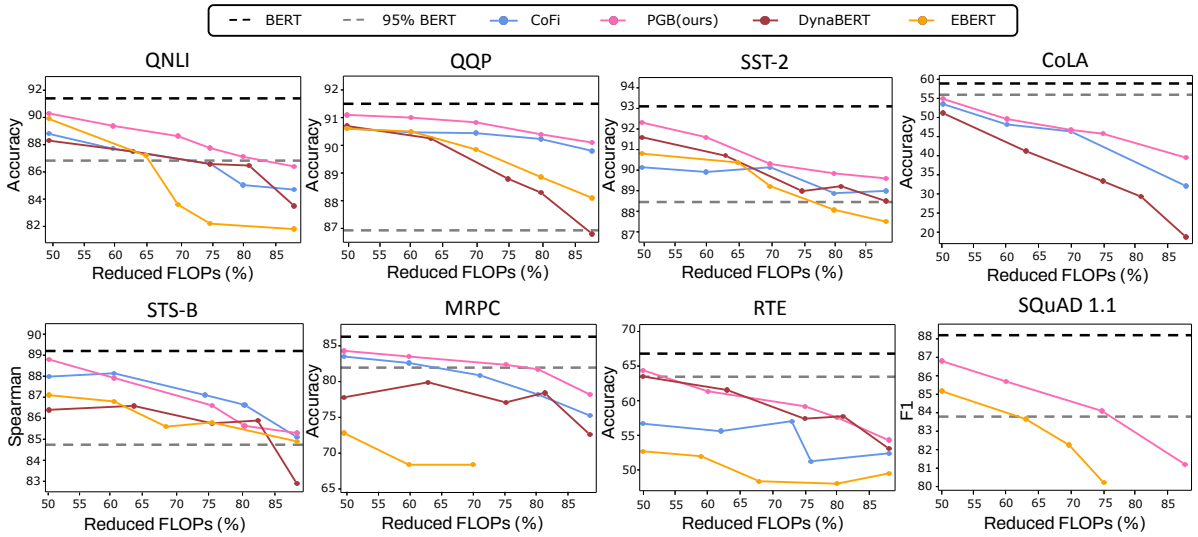

Fig. 3. Performance comparison with structured pruning methods varying the reduced FLOPs ratio on $\text{BERT}_{\text{BASE}}$.

Implementation details. Our PGB method is implemented using Python 3.7.15, PyTorch (Paszke et al. 2019) and CUDA 11.6, along with the Huggingface library (Wolf et al. 2020), which incorporates the latest NLP techniques. We set the hyperparmeter N_{perm} that controls the number of sorting operations in the permutation step, and G_{max} that indicates the maximum number of groups, ensuring that the size of the grouped model is within the given budget capacity. All the experiments are conducted on a PC with NVIDIA GeForce RTX A6000. We report that all experimental results are the average of 5 random seeds within a range between 0 and 10.

As our target model, we use fine-tuned BERT (Devlin et al. 2019) for specific tasks. Even though we mainly compare the performance using $\text{BERT}_{\text{BASE}}$, we also conduct experiments using $\text{RoBERTa}_{\text{BASE}}$ and $\text{DistilBERT}_{\text{BASE}}$ (Sanh, Debut, et al. 2019) (refer to 5.3 and 5.3). Full experimental details can be found at Appendix A, and our implementation is available at: https://github.com/bigdata-inha/BERT_Pruning.

Table 2. Performance comparison with other SOTA pruning methods on $\text{BERT}_{\text{BASE}}$.

Pruning ratio	Method	QNLI Acc.	QQP Acc.	SST-2 Acc.
0%	$\text{BERT}_{\text{BASE}}$	91.43	91.50	93.16
50%	BMP	89.40	90.30	90.71
	LayerDrop	87.60	90.40	90.30
	SNIP	89.50	88.90	91.80
	PGB (Ours)	**90.34**	**91.06**	**92.30**
80%	BMP	86.40	89.30	89.79
	PGB (Ours)	**87.12**	**90.40**	**89.84**

Table 3. Ablation study in PGB (ours) at 40% and 60% of reduced FLOPs before and after re-finetuning using GLUE.

Task	40%		60%	
	Vanilla	Re-finetuned	Vanilla	Re-finetuned
SST-2	92.2	92.7	90.3	91.6
QNLI	90.1	91.0	88.2	89.4
MRPC	84.8	85.1	80.9	83.5

5.2 Main Experimental Results

Performance comparison. Table 1 and Figure 3 show the performance comparison results of PGB with prior structured methods, using the seven tasks of GLUE and SQuAD. To equalize the resulting size of pruned models, we re-implement the released code of the compared methods by ourselves. For CoFi and DynaBERT, since there is no publicly available code for SQuAD, comparative experiments for these two methods were conducted exclusively on the GLUE benchmarks. Table 1 demonstrates the corresponding results with pruning rates 50% and 88%, where 88% is the maximum pruning rate that can be achieved by all the compared methods. It is clearly observed that the proposed PGB method outperforms the previous structured pruning methods in all task-specific BERT models, which indicates that PGB is not only the fastest pruning method as observed in Table 4, but also highly effective to preserve the information of the original model. This empirically implies that we can properly compress transformer architectures with one-shot pruning, without relying on complicated and time-consuming methods.

Figure 3 shows how the compression performance changes when varying the target model size in terms of the number of FLOPs. We can observe that PGB generally achieves the best performance among the compared methods. More importantly, PGB tends to work better at higher compression rates, as the performance degradation of PGB gets smaller than those of the other methods as the size of compressed model decreases. These results imply that PGB is capable of maintaining high performance even at extreme compression cases. Considering the compression speed of PGB, we can claim that PGB is a computationally efficient yet accurate compression method for transformer architectures.

Comparison with other SOTA methods. Table 2 presents additional comparisons between PGB and other state-of-the-art (SOTA) pruning methods, using the reported results from their respective experiments. The compared methods include: hybrid-based block movement pruning (BMP) (Lagunas et al. 2021), LayerDrop that drops certain layers of the model (Sajjad et al. 2020), and SNIP that prunes redundant mappings in residual modules

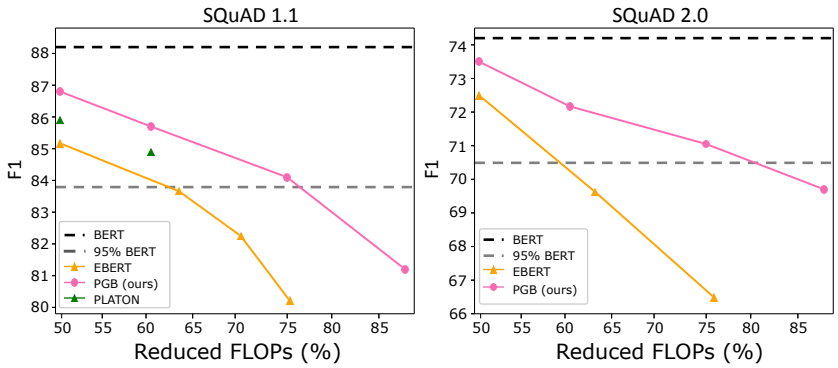

Fig. 4. Performance comparison with structured pruning methods varying the Reduced FLOPs ratio on SQuAD benchmarks.

Table 4. Comparison of the pruning efficiency using QQP with 88% pruning rate on $\text{BERT}_{\text{BASE}}$.

Method	Time for Pruning + Re-finetuning		FLOPs	Accuracy
	(epochs)	(hours)	(G)	(%)
EBERT (Z. Liu et al. 2021)	3 + 3	$\leq$ 3.5 + 2	2.78	88.1
DynaBERT (Hou et al. 2020)	2 + 3	$\leq$ 25 + 38	**2.60**	86.8
CoFi (Xia et al. 2022)	20 + 20	$\leq$ 26 + 23	2.97	89.8
PGB (ours)	**0 + 3**	$\leq$ **0.1 + 2**	**2.60**	**90.1**

(Lin et al. 2020). Similar to the findings in Table 1, PGB shows minimum performance degradation. Of particular interest is the comparison with BMP (Lagunas et al. 2021), as it is introduced as another semi-structured pruning method in Section 2. PGB turns out to outperform BMP even though the reported results of BMP are not from its fully semi-structured pruning scheme, but rather its improved hybrid version yet without distillation.

Experiments on SQuAD Benchmarks. We compare additional pruning methods with PGB for $\text{SQuAD}_{v1.1}$ (Rajpurkar, J. Zhang, et al. 2016) and $\text{SQuAD}_{v2.0}$ (Rajpurkar, Jia, et al. 2018). The compared methods exclude knowledge distillation (KD) and data augmentation. Specifically, our approach is compared with PLATON (Q. Zhang et al. 2022), representing the state-of-the-art in unstructured pruning, and EBERT (Z. Liu et al. 2021). In this context, PLATON utilized the extended $\text{PLATON}_{\text{structure}}$ based on structured pruning. Our PGB method shows minimal performance degradation compared to other methods, as shown in Figure 4.

Efficiency of pruning procedures. To evaluate the efficiency of each pruning method, we measure how long each method takes to prune the original model and re-finetune the pruned model, as shown in Table 4. In particular, we report the case of pruning with 88% on QQP, where the time cost was maximized. Thanks to its one-shot pruning scheme, our PGB method is clearly the fastest pruning method among all the compared methods, while maintaining the best accuracy after pruning and re-finetuning. PGB takes at most 2.1 hours to obtain the final compressed model, whereas most of the other methods takes more than a day to get the same-sized model. Table 4 also reports the computational cost of each compressed model in terms of the number of FLOPs. Although PGB is a semi-structured pruning method, it manages to achieve a comparable model efficiency, without any specific hardware support, to those of the fully structured pruning methods, EBERT (Z. Liu et al. 2021), DynaBERT (Hou et al. 2020) and CoFi (Xia et al. 2022).

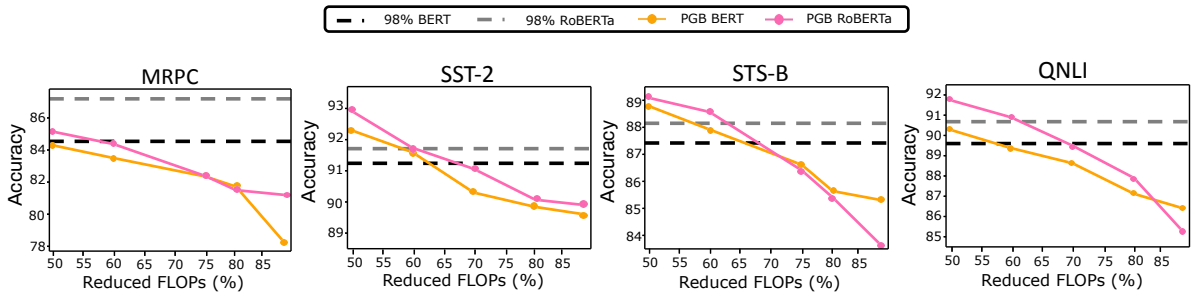

Fig. 5. PGB with $\text{BERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$

Table 5. Performance of PGB on $\text{DistilBERT}_{\text{BASE}}$ using 40% and 60% pruning rates

Pruning Ratio	Method	QNLI	SST-2	MRPC	STS-B
0%	$\text{DistilBERT}_{\text{BASE}}$	88.6	91.3	84.8	85.8
40%	PGB (ours)	88.1	91.2	84.6	85.5
60%		86.5	89.3	83.8	84.3

5.3 More Experimental Results

Pruning on DistilBERT. We conduct additional experiments on PGB with pruning rates of 40% and 60% for $\text{DistilBERT}_{\text{BASE}}$, and the results are presented in Table 5. While $\text{DistilBERT}_{\text{BASE}}$ is half the size of $\text{BERT}_{\text{BASE}}$, as indicated in Table A3, the model configuration remains the same, excluding the number of layers. Consequently, the values of permutation (N_{perm}) and the maximum group number (G_{max}) are identical. The experimental results in Table 5 demonstrate that even for a smaller model, our grouping-based pruning scheme does not result in substantial information degradation.

Pruning on RoBERTa. Figure 5 shows the proposed PGB results with $\text{RoBERTa}_{\text{BASE}}$. Similar to BERT, PGB with pruning rates of 60% on RoBERTa is able to maintain over 98% of the performance of the original model. When the Pruning rate surpasses 60%, RoBERTa demonstrates better performance compared to BERT. However, as the sparsity ratio increases, the performance of BERT surpasses RoBERTa.

5.4 Ablation Studies

Performance of PGB without re-finetuning. Table 3 presents the performance comparison between vanilla PGB, which does not perform re-finetuning, and PGB with re-finetuning. We measure the accuracy of the compressed model with 40% and 60% of reduced FLOPs before and after re-finetuning on task-specific BERT models. We can observe that vanilla PGB itself is already effective to maintain high performance after pruning. For the SST-2 and QNLI tasks, the performance of vanilla PGB is either matches or surpasses that of other pruning methods shown in Figure 3. This demonstrates the capability of our grouped pruning operation to maintain the original performance of large transformer-based models with just a single round of pruning.

Hyperparameter Sensitivity. We conduct ablation studies to investigate the sensitivity to the hyperparameter utilized in the proposed PGB method. In Figures 6 and 7, we present visualization of the accuracy of the compressed model as we vary hyperparameter values, G_{max} and τ. Figure 6 indicates that when the number of groups is limited to a small number (i.e., $G_{max} = 3$), there is a significant decrease in performance, which is probably

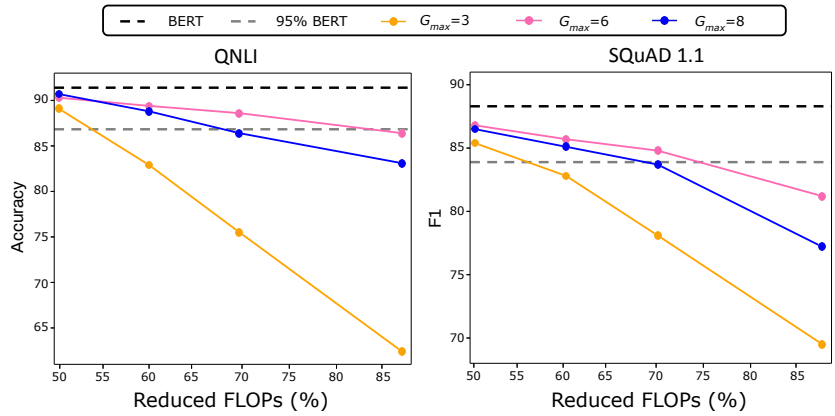

Fig. 6. Ablation study with respect to the number of Group (G_{max}) with τ=1e-5

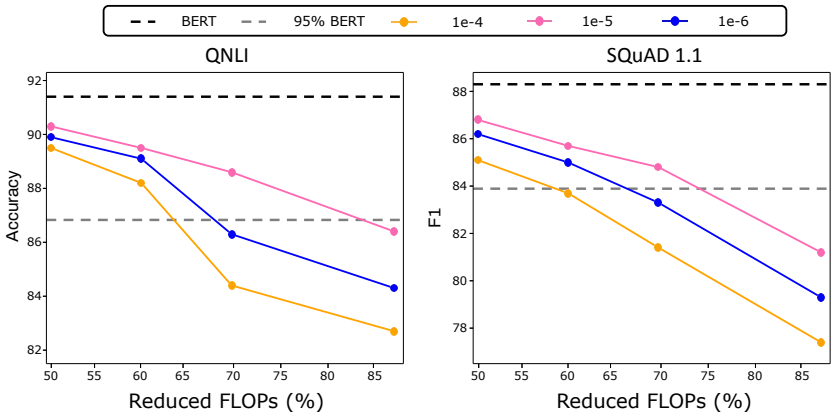

Fig. 7. Ablation study with respect to threshold (τ) with $G_{max} = 6$

because some important weights can be pruned in order to reach each target compressed size, just like in typical structured pruning. At the same time, however, it does not always mean that the larger the G_{max} value, the better the final performance, as the performance also drops in the case of $G_{max} = 8$. In terms of the threshold τ, Figure 7 demonstrates that there is a certain optimal point of τ to maximize the final performance. In our experiments, we found that $G_{max} = 6$ and $\tau = 1e - 5$ produce the most promising results for compressed models.

Pruning Ratios between MHA and FFN. Table 6 reports the module-wise contributions to the reduced FLOPs under different global targets. We observe a clear asymmetry: across 40–60% budgets, the majority of pruning consistently occurs in the FFN sub-layers, while MHA sub-layers remains largely preserved.

Effect of Permutation. We examine the impact of the permutation step by comparing PGB with an unstructured variant that removes the permutation and directly prunes individual weights based on their second-order importance scores, under the same target layer-wise pruning rate. As summarized in Table 7, both methods achieve comparable performance at moderate reduced FLOPs (40%), but the unstructured baseline exhibits a

Table 6. Module-wise percentage-point (pp) contributions of MHA and FFN to the reduced FLOPs under different global targets.

Target (FLOPs)	MHA contrib (pp)	FFN contrib (pp)	reduced FLOPs	Gap (pp)
40%	5.6%	33.3%	38.9%	−1.1
50%	7.5%	42.5%	50.0%	+0.0
60%	11.7%	48.0%	59.7%	−0.3

Table 7. Semi-structured (PGB) vs. Unstructured (2nd-order) at the same **layer-wise** reduced FLOPs on $\text{BERT}_{\text{BASE}}$.

reduced FLOPs	Method	SST-2	QNLI	MRPC
–	$\text{BERT}_{\text{BASE}}$	93.2	91.4	86.3
40%	Unstructured	92.1	90.9	84.5
	PGB (ours)	**92.7**	**91.0**	**85.1**
60%	Unstructured	89.9	87.3	78.3
	PGB (ours)	**91.6**	**89.4**	**83.5**

pronounced degradation at higher FLOPs reduction (60%). In contrast, PGB not only preserves higher accuracy but also induces semi-structured sparsity that is well suited for efficient block-wise computation. These findings indicate that the permutation step is critical role in preserving accuracy under aggressive pruning.

Effect of Grouping Strategy. We further examine the impact of different grouping strategies by introducing two diagnostic variants of PGB, in addition to the original PGB (adaptive G). Table 8 compares three variants under a matched layer-wise 60% FLOPs reduction on $\text{BERT}_{\text{BASE}}$. (1) **Fixed-G** fixes a single global group number (G=6) across all layers, thereby isolating the effect of grouped pruning without adaptive group-number selection. (2) **Full-diag** sets each weight as its own group (G=|diag(W)|), removing the structural coupling among weights and thus quantifying the contribution of adaptive grouping. As shown, PGB consistently achieves the highest accuracy across all tasks, demonstrating the clear advantage of adaptive group-number search. Even compared with Full-diagonal pruning, which removes group-level regularity, PGB still achieves 1–3 pp higher accuracy, indicating that adaptively balancing structured and unstructured sparsity improves generalization. In contrast, Fixed-G shows a noticeable performance drop, particularly on tasks requiring fine-grained contextual reasoning. This degradation mainly arises because enforcing a uniform G across all layers leads to excessive pruning in MHA sub-layers, where parameters are relatively fewer and more information-sensitive. In comparison, adaptive PGB assigns smaller G to important attention matrices and larger G to redundant FFN layers, resulting in a more balanced sparsity allocation and better preservation of representational capacity.

6 Conclusion

This paper introduced PGB, one-shot semi-structured pruning with a grouping strategy, as a fast and simple compression approach for transformer-based models. PGB efficiently compresses task-specific BERT models into lightweight and accurate versions within a few hours, contrasting with other SOTA methods that take more than a day to achieve comparable results. By finding an adaptively grouped architecture, PGB combines the advantages of structured pruning and unstructured pruning, offering both computational efficiency and high

Table 8. Effect of grouping strategy at matched 60% reduced FLOPs on $\text{BERT}_{\text{BASE}}$. For the Fixed-G baseline, we set $G = 6$, chosen as a representative global group size.

Method	SST-2	QNLI	MRPC
PGB (ours, adaptive G)	**91.6**	**89.4**	**83.5**
Fixed-G (Vanilla)	85.9	85.5	77.1
Fixed-G (Re-finetuned)	87.2	86.8	78.7
Full-diag (Re-finetuned)	90.3	87.9	80.6

accuracy. Through extensive experiments, we validated that PGB is a practical solution for quickly compressing complex transformer architectures without significant performance degradation.

Acknowledgments

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No.2022-0-00448, Deep Total Recall: Continual Learning for Human-Like Recall of Artificial Neural Networks), and in part by INHA UNIVERSITY Research Grant.

References

T. B. Brown et al.. 2020. "Language Models are Few-Shot Learners." In: *Advances in Neural Information Processing Systems*, 1877–1901.

D. M. Cer, M. T. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. 2017. "SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation." In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Ed. by S. Bethard, M. Carpuat, M. Apidianaki, S. M. Mohammad, D. M. Cer, and D. Jurgens, 1–14.

I. Chelombiev, D. Justus, D. Orr, A. Dietrich, F. Gressmann, A. Koliousis, and C. Luschi. 2021. "Groupbert: Enhanced transformer architecture with efficient grouped structures." *arXiv preprint arXiv:2106.05822*.

T. Chen, J. Frankle, S. Chang, S. Liu, Y. Zhang, Z. Wang, and M. Carbin. 2020. "The lottery ticket hypothesis for pre-trained bert networks." *Advances in neural information processing systems*, 33, 15834–15846.

X. Chen, Y. Cheng, S. Wang, Z. Gan, Z. Wang, and J. Liu. 2020. "Earlybert: Efficient bert training via early-bird lottery tickets." *arXiv preprint arXiv:2101.00063*.

J. Devlin, M. Chang, K. Lee, and K. 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*, 4171–4186.

W. B. Dolan and C. Brockett. 2005. "Automatically Constructing a Corpus of Sentential Paraphrases." In: *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005*.

A. Fan, E. Grave, and A. Joulin. 2019. "Reducing Transformer Depth on Demand with Structured Dropout." *CoRR*, abs/1909.11556.

S. Han, J. Pool, J. Tran, and W. J. Dally. 2015. "Learning both Weights and Connections for Efficient Neural Network." In: *Advances in Neural Information Processing Systems*, 1135–1143.

B. Hassibi, D. G. Stork, and G. J. Wolff. 1993. "Optimal Brain Surgeon and general network pruning." In: *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*. IEEE, 293–299.

G. E. Hinton, O. Vinyals, and J. Dean. 2015. "Distilling the Knowledge in a Neural Network." *CoRR*, abs/1503.02531.

L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu. 2020. "DynaBERT: Dynamic BERT with Adaptive Width and Depth." In: *Advances in Neural Information Processing System (NeurIPS)*. Vol. 33.

Y. Ioannou, D. Robertson, R. Cipolla, and A. Criminisi. 2017. "Deep roots: Improving cnn efficiency with hierarchical filter groups." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1231–1240.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. 2020. "TinyBERT: Distilling BERT for Natural Language Understanding." In: *Findings of Empirical Methods in Natural Language Processing (EMNLP)*, 4163–4174.

F. Lagunas, E. Charlaix, V. Sanh, and A. M. Rush. 2021. "Block Pruning For Faster Transformers." In: *Empirical Methods in Natural Language Processing (EMNLP)*, 10619–10629.

N. Lee, T. Ajanthan, and P. H. Torr. 2018. "Snip: Single-shot network pruning based on connection sensitivity." *arXiv preprint arXiv:1810.02340*.

Z. Lin, J. Z. Liu, Z. Yang, N. Hua, and D. Roth. 2020. "Pruning Redundant Mappings in Transformer Models via Spectral-Normalized Identity Prior." In: *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020* (Findings of ACL). Vol. EMNLP 2020. Association for Computational Linguistics, 719–730.

Y. Liu et al.. 2019. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." *CoRR*, abs/1907.11692.

Z. Liu, F. Li, G. Li, and J. Cheng. 2021. "EBERT: Efficient BERT inference with dynamic structured pruning." In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 4814–4823.

J.-H. Luo, J. Wu, and W. Lin. 2017. "Thinet: A filter level pruning method for deep neural network compression." In: *Proceedings of the IEEE international conference on computer vision*, 5058–5066.

P. Michel, O. Levy, and G. Neubig. 2019. "Are sixteen heads really better than one?" *Advances in neural information processing systems*, 32.

P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. 2019. "Importance Estimation for Neural Network Pruning." In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 11264–11272.

S. Park, G. Kim, J. Lee, J. Cha, J.-H. Kim, and H. Lee. 2020. "Scale down Transformer by Grouping Features for a Lightweight Character-level Language Model." In: *Proceedings of the 28th International Conference on Computational Linguistics*, 6883–6893.

A. Paszke et al.. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." *CoRR*, abs/1912.01703.

P. Rajpurkar, R. Jia, and P. Liang. 2018. "Know what you don't know: Unanswerable questions for SQuAD." *arXiv preprint arXiv:1806.03822*.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. 2016. "SQuAD: 100, 000+ Questions for Machine Comprehension of Text." In: *Empirical Methods in Natural Language Processing (EMNLP)*, 2383–2392.

H. Sajjad, F. Dalvi, N. Durrani, and P. Nakov. 2020. "Poor Man's BERT: Smaller and Faster Transformer Models." *CoRR*, abs/2004.03844.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. 2019. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter." *CoRR*, abs/1910.01108.

V. Sanh, T. Wolf, and A. Rush. 2020. "Movement pruning: Adaptive sparsity by fine-tuning." *Advances in Neural Information Processing Systems*, 33, 20378–20389.

S. P. Singh and D. Alistarh. 2020. "WoodFisher: Efficient Second-Order Approximation for Neural Network Compression." In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. 2013. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1631–1642.

Z. Su, L. Fang, W. Kang, D. Hu, M. Pietikäinen, and L. Liu. 2020. "Dynamic group convolution for accelerating convolutional neural networks." In: *Computer Vision–ECCV 2020: 16th European Conference*, 138–155.

Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. 2020. "MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2158–2170.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. "Attention is All you Need." In: *Advances in Neural Information Processing Systems*, 5998–6008.

E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. 2019. "Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned." *arXiv preprint arXiv:1905.09418*.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. 2019. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding." In: *International Conference on Learning Representations (ICLR)*.

W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. 2020. "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers." In: *Advances in Neural Information Processing Systems*, 5776–5788.

A. Warstadt, A. Singh, and S. R. Bowman. 2019. "Neural Network Acceptability Judgments." *Transactions of the Association of Computational Linguistics (TACL)*, 7, 625–641.

T. Wolf et al.. 2020. "Transformers: State-of-the-art natural language processing." In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 38–45.

M. Xia, Z. Zhong, and D. Chen. 2022. "Structured Pruning Learns Compact and Accurate Models." In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1513–1528.

G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G.-J. Qi. 2018. "Interleaved structured sparse convolutional neural networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8847–8856.

Q. Zhang, S. Zuo, C. Liang, A. Bukharin, P. He, W. Chen, and T. Zhao. 2022. "Platon: Pruning large transformer models with upper confidence bound of weight importance." In: *International Conference on Machine Learning*. PMLR, 26809–26823.

X. Zhang, X. Zhou, M. Lin, and J. Sun. 2018. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856.

Z. Zhang, F. Qi, Z. Liu, Q. Liu, and M. Sun. 2021. "Know what you don't need: Single-Shot Meta-Pruning for attention heads." *AI Open*, 2, 36–42.

R. Zhao and W. Luk. 2019. "Efficient Structured Pruning and Architecture Searching for Group Convolution." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 1961–1970.

A Experimental Details

A.1 Details for Comparison Methods

We conduct comparative experiments on structured pruning for BERT: CoFi (Xia et al. 2022), BMP (based Hybrid Filled) (Lagunas et al. 2021), DynaBERT (Hou et al. 2020) and EBERT (Z. Liu et al. 2021). All these methods belong to the iterative pruning approach, which involves performing pruning during training to improve performance. CoFi (Xia et al. 2022) consists of 2 stages: pruning and final finetuning. Each stage involves 20 epochs of training with layer-wise distillation. BMP (Lagunas et al. 2021) selects appropriate components to prune at the block level over 20 epochs training stages. Subsequently, prediction layer distillation is performed on the pruned model. DynaBERT (Hou et al. 2020) utilizes a 2-stage training process with knowledge distillation to dynamically prune the width and depth size, followed by a final finetuning stage. The number of training epochs varies depending on the data size. For large datasets, the width-adaptive and width- and depth-adaptive stages are performed for 1 epoch of training each, while for small datasets, each stage is performed for 3 epochs of training. Additionally, in all cases, 3 additional epochs of finetuning are conducted. EBERT (Z. Liu et al. 2021) involves 3 epochs of joint training during the pruning stage, followed by 3 epochs of final finetuning on the pruned model.

A.2 Hyperparameters

The detailed experimental setup for PGB is provided in Table A1. During the PGB pruning process, we utilize two hyperparameters, namely N_{perm} and G_{max}. Also, we use the hyperparameters of the original BERT model and those of prior pruning methods for BERT.

Table A1. Hyperparameter settings of PGB in experiments

Hyperparameters	
batch size	32 (GLUE), 16 (SQuAD)
pruning/finetuning epochs	0 / 3
finetuning learning rate	2e-5, 3e-5
max sequence length	128 (GLUE), 384 (SQuAD)
G_{max}	6
N_{perm}	6
τ	1e-5
inference batch size	128 (GLUE) / 32 (SQuAD)

A.3 Details About Benchmark Datasets

We perform experiments using the GLUE (A. Wang et al. 2019) and SQuAD (Rajpurkar, Jia, et al. 2018; Rajpurkar, J. Zhang, et al. 2016) benchmark datasets. Each task of GLUE and SQuAD datasets can fall into the following categories:

- Question Answering: $\text{SQuAD}_{v1.1}$, $\text{SQuAD}_{v2.0}$
- Sentence Pair Similarity: QQP, MRPC (Dolan and Brockett 2005), STS-B (Cer et al. 2017)
- Natural Language Inference: RTE, QNLI
- Single Sentence Task: SST-2 (Socher et al. 2013), CoLA (Warstadt et al. 2019)

The detailed information regarding the dataset size and metric for each task is provided in Table A2.

Table A2. Dataset summary of GLUE and $\text{SQuAD}_{\text{v1.1}}$.

Task	#Train	#Dev	Metrics
QNLI	105k	5.5k	Acc.
QQP	364k	40k	Acc.
SST-2	67k	872	Acc.
CoLA	8.5k	1k	Matthew's corr.
STS-B	7k	1.5k	Spearman corr.
MRPC	3.7k	408	Acc.
RTE	2.5k	276	Acc.
$\text{SQuAD}_{\text{v1.1}}$	88k	10.5k	EM/F1
$\text{SQuAD}_{\text{v2.0}}$	132k	12k	EM/F1

A.4 Model Configurations

The experimental models used in this paper include $\text{BERT}_{\text{BASE}}$, $\text{DistilBERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ architectures, each of which has its respective parameter configuration, as described in Table A3. The notations d and d_{ffn} represent the dimensionality of matrices in MHA layers and FFN layers, respectively.

Table A3. Specifications of $\text{BERT}_{\text{BASE}}$, $\text{DistilBERT}_{\text{BASE}}$ and $\text{RoBERTa}_{\text{BASE}}$ models

Model	#Params	#Encoder Layer	Hidden dim.	#Heads	d	d_{ffn}
$\text{BERT}_{\text{BASE}}$	85M	12	768	12	768	3072
$\text{RoBERTa}_{\text{BASE}}$	100M	12	768	12	768	3072
$\text{DistilBERT}_{\text{BASE}}$	43M	6	768	12	768	3072

A.5 Rows and Columns for each Weight Matrix in BERT

Our PGB method performs pruning on individual parameters of W^Q, W^K, W^V, W^O in the MHA sub-layers, as well as $W^{(1)}$ and $W^{(2)}$ in the FFN sub-layers. The number of rows and columns for each weight matrix is as follows:

- $(d, N_H \times \frac{d}{N_H}) : W^Q, W^K, W^V, W^O$ in MHA
- $(d, d_{ffn}), (d_{ffn},\ d) : W^{(1)}, W^{(2)}$ in FFN

A.6 Efficient Group Pruning and Channel-Permutation of (Zhao and Luk 2019)

For completeness, we reproduce the group pruning and channel permutation algorithm of (Zhao and Luk 2019), as shown in Algorithm 4. The method employs a heuristic alternating optimization procedure, where row and column permutations are updated in turn, followed by pruning. Although no global optimality guarantee is provided, each alternating step monotonically reduces the reconstruction error, leading to convergence to a coordinate-wise optimum.

Received 12 October 2025; accepted 25 January 2026

Algorithm 4: Pruning and Channel-Permutation (adapted from (Zhao and Luk 2019))

Input: Weight matrix $W \in \mathbb{R}^{M \times N}$, target sparsity s, max iterations T
Output: Pruned and permuted weight matrix $\widehat{W}$

Initialize row permutation $\pi_r \leftarrow$ identity permutation
Initialize column permutation $\pi_c \leftarrow$ identity permutation
for $t \leftarrow 1$ ***to*** T **do**
    $\pi_r \leftarrow$ Update-Row-Permutation(W, π_c)
    $\pi_c \leftarrow$ Update-Column-Permutation(W, π_r)
    $W \leftarrow W_{\pi_r, \pi_c}$
    $W \leftarrow$ Prune(W, s)
$\widehat{W} \leftarrow W_{\pi_r, \pi_c}$