

# Flexible Group Count Enables Hassle-Free Structured Pruning

## Supplementary Material

### 7. Limitation and broader impact

Although our evaluation game throughout the experiment and ablation study is objectively beyond most, if not all, available structured pruning literature, our findings on using dynamic group counts on GKP to optimize the performance gain from the improved pruning freedom are still mainly limited to the image classification task on the benign dataset. This limitation is a result of the CNN pruning community collectively opting for image classification tasks as a proxy for compression effectiveness, and thus, it is beyond our ability to adopt most, not to mention all featured methods, to a different task. That being said, we do extend our own proposed method — LeanFlex-GKP — to the diffusion task, which is one of the other major use cases that rely on CNN backbones.

Further, given that GKP is a recently developed and recognized pruning granularity that has a lack of exposure in the ML efficiency field, a more comprehensive study of LeanFlex-GKP on various task domains can be investigated for future work. In terms of broader impacts, we are aware that the proposed research is often utilized in an on-device fashion, and we’d like to caution our users to carefully evaluate the adopted technology against the intended use before mass deployment, especially under a high-stake scenario.

### 8. Extended related works

**Structured pruning.** As discussed in Section 2.1 and illustrated Figure 1, structured pruning is a family of pruning methods that removes model components in groups, where the majority of methods are capable of delivering densely structured pruned models for immediate efficiency benefits. Below, we briefly introduce some important aspects of structured pruning and refer our reader to comprehensive survey work like Blalock et al. [2] and He and Xiao [17] for details.

Prior papers on densely structured pruning have carried out such pruning operations on (or determined by) filters, channels, layers, inputs, or combinations of the above [1, 3, 5, 6, 6–8, 10–12, 14, 15, 18–20, 23, 28, 28–33, 36, 37, 47–51, 54, 57, 59, 60, 64, 64]. Scholars often refer to this as the “**granularity**” or simply the “**type**” of a pruning method. Among them, filter/channel pruning is considered the most popular structured pruning granularity as it can gain immediate efficiency benefits after channel removal.

Outside the pruning granularity, the **pipeline of a pruning method** plays another major role in deciding which method to adopt. Pruning operations can be conducted from-

scratch (e.g., OTov2 by Chen et al. [3]), during training (e.g., Roy et al. [44]), or post-train (e.g., FPGM by He et al. [20]). Many pruning methods also require intervention (or access to information) at an earlier stage, then conduct actual pruning at a later stage. E.g., DMCP by Guo et al. [14] further adjusts weights of a pruned model before pruning, and TMI-GKP by Zhong et al. [60] requires access to the model training checkpoint to guide its pruning operations. Most pruning methods follow a *train - prune - fine-tune/retrain* pipeline, as the trained model provides a good starting and reference point for pruning and evaluating. However, this pipeline suffers the natural drawback of having to both train the unpruned baseline model and fine-tune/retrain the pruned model, where from-scratch methods or fine-tuning free methods like Narshana et al. [39] may effectively avoid such compute cost, though often with a trade-off of delivering lower accuracy retention.

Further, the **scheduling of a pruning method** drastically affects the efficiency and adaptability of a method. Most pruning methods can be roughly categorized into one-shot or iterative pruning. The former prunes all redundant model components all at once; the latter, as its name implies, conducts pruning gradually with weight updates between two pruning operations. One-shot pruning is often considered easier to deploy and more efficient to run, though iterative pruning is generally more performant on accuracy retention.

On the note of adaptability, the **data dependency of a pruning method** potentially plays another vital role in the adaptability of a method. With *data-agnostic* pruning methods do not require data access to determine what to prune, and *data-dependent* or *data-informed* methods do the opposite.

Other **implementations details** such as *hard* or *soft pruning* (whether the pruned components are entirely removed from the model to yield a pruned model with reduced dimension, or just zero-masked), reduction control and estimation (if one can reliably control and predict the memory and compute requirements of a pruned model before conducting the actual pruning), and hyperparameter tuning pressure (whether the method has a lot of hyperparameters to adjust, or if the method has a way to tune them automatically, like AAP by Zhao et al. [59] and AMC by He et al. [19]) also have their influences, especially on the user-friendliness aspect of a method.

Our method is a one-shot, post-train, data-agnostic, hard-pruning method with only one tunable hyperparameter (which is primarily determined by the layer dimension; see Appendix 10.2 for details). Yet, one may adjust the pruning rate setting of our method to reliably control and predict the

*MACs* and *Params* reduction of the pruned model (see the relation between pruning rate and pruned model size in experiments like Table 19 and 21). These characteristics put our method on top of the efficiency, adaptability, and user-friendliness lists over many other pruning methods under the context of densely structured CNN pruning.

**Grouped kernel pruning.** Grouped kernel pruning (GKP) is a special type of densely structured pruning granularity that is able to offer finer pruning freedom than filter or channel pruning methods. As demonstrated in Figure 2, GKP is, in essence, a combination of pruning a stack of kernels under the same filter group (a.k.a. grouped kernels) and reconstructing the remaining model to a grouped convolution format [24]. Granted, grouped convolution itself is proposed with efficiency motivations, and we may say that this granularity — outside a pruning context — naturally exists with the debut of AlexNet [26].

The combination of densely structured pruning with grouped convolution is first introduced in Yu et al. [55], but the method, unfortunately, did not attract significant attention from the community. One potential reason for this is due to the high complexity of its pruning procedure: feature maps-dependent iterative pruning plus fine-tuning with knowledge distillation. Other reasons include not performing a comprehensive ablation study to track its contributions, as well as a lack of comparable experiments to modern pruning arts. Another work, Guo et al. [13], also explores structured pruning with grouped convolution, but it still introduces sparsity to its pruned model with its zero-padded implementation to support unequal group sizes.

A refined procedure at the intersection of densely structured pruning and grouped convolution is brought by Zhong et al. [60] as TMI-GKP, where they stipulate a three-stage procedure consisting of 1) filter grouping, 2) grouped kernel pruning, and 3) grouped convolution reconstruction – as well as coining it with the term “*grouped kernel pruning*.” The GKP procedure and nomenclature have since been adopted by TMI-GKP’s concurrent, related, and follow-up works like He and Xiao [17], Park et al. [42], Zhang et al. [58].

## 9. Extended proposed method

### 9.1. Formal definition of $L_2$ & geometric median-based grouped kernel pruning

To better illustrate the formal definition of our purposed method, let  $W$  be the set containing all filters  $f$  in a certain convolutional layer of a CNN:

$$W : \{f_1, f_2, \dots, f_n\},$$

where  $n$  is the number of filters in that layer, a.k.a. the `out_channels`. The filter grouping procedure is then applied on such  $f_1$  to  $f_n$ .

The numbers of desire grouped filters (a.k.a. the `Conv2d(groups)` or group count) are chosen from *candidate group count* list (see Table 5), where one of them is ultimately selected after the *post-prune group count evaluation* (see Section 4.4).

For simplicity, let's assume that the current *candidate group count* is  $m$  (the number of filters  $n$  must be divisible by  $m$  because in our implementation every group in the same layer has exactly the same number of filters to be able to reconstruct to a grouped convolution format), and let  $G$  be the set containing all filter groups  $g$  after KPP-aware filter clustering is applied on all the filters in  $W$  (Figure 5):

$$W : \{f_1, f_2, \dots, f_n\} \xrightarrow{\text{KPP-aware Filter Grouping}} G : \{g_1, g_2, \dots, g_m\},$$

where  $g_i$  is the  $i_{th}$  grouped filters in  $G$ .

Suppose there are  $\lambda$  number of kernels in a filter (a.k.a. `in_channel`), we shall have  $\lambda$  grouped kernels ( $gk$ ) in each filter group  $g$ :

$$g : \{gk_1, gk_2, \dots, gk_\lambda\}.$$

We then compute the geometric median  $\tau$  of each filter group  $g$  among all grouped kernels ( $gk$ ) in  $g$ :

$$G : \{g_1, g_2, \dots, g_m\} \xrightarrow{\text{Compute Geometric Median}} \{\tau_1, \tau_2, \dots, \tau_m\}.$$

After we get each group's geometric median, we start to prune grouped kernels  $gk$  in each filter group  $g$ . Inside each filter group  $g$ , we compute the L2-norm of each grouped kernels  $gk$  in  $g$  and add them to list  $Q$ :

$$g : \{gk_1, gk_2, \dots, gk_\lambda\} \xrightarrow{\text{Compute L2-Norm}} Q : \{q_1, q_2, \dots, q_\lambda\}, \text{ where } q_i = \|gk_i\|_2.$$

Also, we compute the euclidean distance of each grouped kernels  $gk$  to its group's geometric median  $\tau$  and add them to list  $D$ :

$$g : \{gk_1, gk_2, \dots, gk_\lambda\} \xrightarrow{\text{gk's Distance to } \tau} D : \{d_1, d_2, \dots, d_\lambda\}, \text{ where } d_i = \text{Euclidean}(gk_i, \tau).$$

Then we do Min-Max Normalization on list  $Q$  and  $D$  separately:

$$Q : \{q_1, q_2, \dots, q_\lambda\} \xrightarrow{\text{Min-Max Normalization}} Q' : \{q'_1, q'_2, \dots, q'_\lambda\},$$

such that

$$q'_i = \frac{q_i - \min(Q)}{\max(Q) - \min(Q)}.$$

Using the same equation above, we conduct the same Min-Max Normalization upon list  $D$ :

$$D : \{d_1, d_2, \dots, d_\lambda\} \xrightarrow{\text{Min-Max Normalization}} D' : \{d'_1, d'_2, \dots, d'_\lambda\}.$$

We then calculate the importance score  $I$  of each grouped kernels ( $gk$ ) (Figure 6):

$$g : \{gk_1, gk_2, \dots, gk_\lambda\} \xrightarrow{\text{calculate importance score}} I : \{I_1, I_2, \dots, I_\lambda\}, \text{ where } I_i = q'_i + d'_i.$$

Finally, assume that the pruning rate (ratio of grouped kernels to be pruned) is  $pr$ , we preserve  $1 - pr$  ratio of grouped kernels  $gk$  in group  $g$  with higher importance score  $I$ . So with  $\lambda$  number of  $gk$  and preserve rate  $1 - pr$ , we will have  $\lambda(1 - pr)$  numbers of preserved grouped kernels  $gk$  after pruning:

$$\underbrace{g : \{gk_1, gk_2, \dots, gk_\lambda\}}_{\lambda \text{ numbers of } gk} \xrightarrow{\text{pruning}} \underbrace{g^* : \{gk_1^*, gk_2^*, \dots, gk_\lambda^*\}}_{\lambda(1 - pr) \text{ numbers of } gk}.$$

So in general, for each *candidate group count*  $m$ , we will have GKP result  $G^*$ :

$$\underbrace{W : \{f_1, f_2, \dots, f_n\} \rightarrow G : \{g_1, g_2, \dots, g_m\}}_{\text{filter grouping stage}} \rightarrow \underbrace{G^* : \{g_1^*, g_2^*, \dots, g_m^*\}}_{gk \text{ pruning stage}}.$$

For demonstration, suppose we have three *candidate group count*  $[m_1, m_2, m_3]$ , we will have three pruning results  $G_1^*, G_2^*, G_3^*$ . Then we evaluate these three pruning results via *post-prune group count evaluation* (Section 4.4). Finally, based on *post-prune group count evaluation*, the optimal pruning result will be transformed into densely structured group convolution layer.

## 9.2. Formal definition of post-prune group count evaluation

For simplicity, let's assume that the current *candidate group count* is  $m$  ( $m$  number of groups in a layer), and suppose there are  $\lambda$  number of kernels in a filter. After grouped kernel pruning (Section 9.1), we can get pruned layer:

$$G^* : \{g_1^*, g_2^*, \dots, g_m^*\}.$$

Then we may compute the preserved grouped kernels geometric median  $\tau^*$  for each pruned group  $g^*$ :

$$G^* : \{g_1^*, g_2^*, \dots, g_m^*\} \xrightarrow{\text{Compute Geometric Median}} \{\tau_1^*, \tau_2^*, \dots, \tau_m^*\},$$

where  $\tau_i^*$  is the **geometric median of preserved gk** (different from Section 4.3, which are the geometric median for all  $gk$  including those being pruned). Let  $||$  be an operator that calculate cardinality in terms of grouped kernels. For group  $g_i^*$  with geometric median  $\tau_i^*$ :

$$A(g_i^*) = \frac{1}{|g_i^*|} \sum_{gk \in g_i^*} \text{Euclidean}(gk, \tau_i^*),$$

where  $A(g_i^*)$  is the average euclidean distance between  $gk$  of  $i^{th}$  group and its geometric median (intra-group similarity).

$$B(g_i^*) = \frac{1}{|G^* - g_i^*|} \sum_{gk \in G^* - g_i^*} \text{Euclidean}(gk, \tau_i^*),$$

where  $B(g_i^*)$  is the average euclidean distance between  $i^{th}$  group's geometric median and all  $gk$  that are not in  $i^{th}$  group (inter-group distinctions). Now, we may calculate the group count evaluation score  $S$  upon the pruning result of each candidate group count (see Figure 7) as:

$$S = \gamma \sum_{i=1}^m B(g_i^*) - A(g_i^*), \text{ where } \gamma = \frac{\lambda}{m}. \quad (1)$$

Since we have *candidate group count* list with different group counts (see Table 5), during *post-prune group count evaluation*, each group count will provide a unique score  $S$  using Equation (1).

For demonstration, following the example in the last paragraph of Section 9.1, suppose we have three *candidate group counts*  $[m_1, m_2, m_3]$ , we will have three pruning results  $G_1^*, G_2^*, G_3^*$ . Using Equation (1), we will have three *group count evaluation scores*  $S_1, S_2, S_3$ . So we only need to choose the pruning result  $G^*$  among  $G_1^*, G_2^*, G_3^*$  with largest  $S$  value in  $S_1, S_2, S_3$ .

A large value of  $S$  implies a pruned layer with better intra-group similarity and inter-group distinctions. Utilizing this  $S$  evaluation metric, we are able to leverage the joint optimization effect of grouping and pruning on different group count settings and, therefore, obtain a better pruning result for the layer-in-question.

### 9.3. Pseudocode for the proposed method

---

**Algorithm 1** General Procedure of LeanFlex-GKP on a Single Convolutional Layer

---

**Input:** Candidate Group Count Queue  $Q$  ▷ candidate Conv2d(groups) like  $[2, 4, 8]$   
**Initialize:** Empty List  $P$  ▷ storage of pruning strategies w.r.t. each group count candidate  
**for**  $q \in Q$  **do:** ▷ looping though all group count candidates  
    Conduct  $k$ -Means<sup>++</sup> clustering on filters to get  $q$  amount of centers  
     $CS \leftarrow q$  candidate sequences of centers generated by multiple restarts ▷ see Figure 5  
    **Initialize:** Empty List  $FG$  ▷ to store filter grouping results  
    **for**  $s \in CS$  **do:**  
        Get filter grouping result  $fg_s$  for the  $s$  center sequence, as illustrated in Figure 5(c)  
         $FG.append(fg_s)$   
    Determine the optimal filter grouping result  $fg_{opt}$  with least intra-group distance (Figure 5)  
    **for** filter group  $g \in fg_{opt}$  **do:** ▷ Prune grouped kernels inside each filter group  
         $g_{pruned}^q \leftarrow$  Prune  $g$  w.r.t. the  $L_2$  & geometric median-based method stipulated in Figure 6  
         $P.append(g_{pruned}^q)$   
    Determine the best  $g_{opt} \in P$  according to Section 9.2 ▷ illustrated in Figure 7  
**return** Pruned convolutional layer  $g_{opt}$  and its corresponding group count  $q_{opt}$

---

## 10. Ablation studies

In company with our main experiment results showcased in Section 5 and Appendix 11, we provide ablation studies of our proposed LeanFlex-GKP method from different perspective-of-interests,

### 10.1. On different procedural recipes

In this section, we conduct ablation studies on our proposed methods by evaluating the contribution of our proposed algorithmic components. We utilized BasicBlock ResNets [16] on the CIFAR10 dataset [25] for their lightweighness. All comparisons are done so with the pruning rate being 43.75% (meaning 43.75% of the original model is removed; please refer to the LeanFlex-GKP reports in Table 21 for the exact reduction status).

In Table 2, we try to investigate the influence of different grouping approaches under our Flexible Group Count pipeline, where our proposed filter grouping method, **KPP-aware filter grouping** (Section 4.2) has the optimal empirical results.

Table 2. Ablation Study on Different Filter Grouping Approaches

Method Name	ResNet32	ResNet56	ResNet110
FGC + RandomGroup + GM&L2	92.81	93.72	94.67
FGC + No Restart KPP-aware Filter Grouping + GM&L2	<b>93.04</b>	93.76	94.61
FGC + Equal-size KPP + GM&L2	92.52	93.77	-
<b>FGC + KPP-aware Filter Grouping + GM&amp;L2 (ours)</b>	93.01	<b>94.00</b>	<b>94.92</b>

In Table 3, we try to evaluate the validity of our proposed  **$L_2$  & geometric median-based grouped kernel pruning** method in comparison with other pruning strategies (while under our flexible group count pipeline). It is observed that our flexible group count pipeline is best working with the GM&L2 grouped kernel pruning method we proposed in Section 4.3.

Table 3. Ablation Study on Different Grouped Kernel Pruning Approaches

Method Name	ResNet32	ResNet56
FGC + KPP-aware Filter Grouping + TMI (Greedy)	92.84	93.58
FGC + KPP-aware Filter Grouping + Distance to GM	92.92	93.73
FGC + KPP-aware Filter Grouping + L2Norm	92.99	93.61
<b>FGC + KPP-aware Filter Grouping + GM&amp;L2 (ours)</b>	<b>93.01</b>	<b>94.00</b>

In Table 4, we showcase the power of adding flexible group count into our GKP procedure as stipulated in **post-prune group count evaluation** (Section 4.4), it is observed that the `Conv2d(groups)` decided by our evaluation design has better performance than other constant or dynamic group count settings. Specifically, the “Eight Groups” strategy is utilized in TMI-GKP [60], and the “Random Groups Reassign” means to randomly re-distribute the optimal group counts (deemed by our post-prune group count evaluation method) across different layers; granted such reassignment is legal in term of layer dimensions.

Table 4. Ablation Study on Different Group Count Settings

Method Name	ResNet32	ResNet56	ResNet110
Eight Groups (TMI’s setting) + KPP-aware Filter Grouping + GM&L2	92.86	93.82	94.72
Max Groups + KPP-aware Filter Grouping + GM&L2	92.88	93.82	94.72
Random Groups Reassign + KPP-aware Filter Grouping + GM&L2	92.75	93.89	94.43
<b>FGC + KPP-aware Filter Grouping + GM&amp;L2 (ours)</b>	<b>93.01</b>	<b>94.00</b>	<b>94.92</b>

### 10.2. On different hyperparameter settings

We illustrate the hyperparameter settings for all reported LeanFlex-GKP experiments as Table 5. The only tunable hyperparameter for LeanFlex-GKP is Candidate Group Counts, i.e., a set of `Conv2D(groups)` setting considered. Granted most CNN architectures have different dimensions across their convolutional layer, this setting should be adjusted subject to subject to the layer’s `out_channels`. In our case, we basically checkout what is the largest `Conv2D(groups)` applicable to a particular convolutional layer, then generate the rest of group count candidates by reducing it by half.

Table 5. Hyperparameter Settings for LeanFlex-GKP’s Reported Results. **PR** stands for pruning rate, **Budget** represents the *train - fine-tune* budget in terms of number of epochs, **BS** implies batch sizes, and **Candidate Group Counts** indicate the different Conv2D (groups) settings considered. We provide settings in `torch` style code snippets

Model	Dataset	PR	Budget	BS	Optimizer & Learning Rate	Candidate Group Counts
ResNet20 ResNet32 ResNet56 ResNet110	CIFAR10	43.75%	300 - 300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	S2 S1 S2 S1
ResNet32 ResNet56 ResNet110	CIFAR10	62.5%	300 - 300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	S1 S2 S1
ResNet56 ResNet110	CIFAR100	43.75%	200-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	S2 S2
ResNet56	Tiny-ImageNet	37.5%	100-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) MultiStepLR(milestones=[80, 90], gamma=0.1)	S2
ResNet101	Tiny-ImageNet	43.75%	300-20	256	SGD(lr=0.1, momentum=0.9, weight_decay=1e-4) StepLR(step_size=5, gamma=0.1)	[8, 16, 32]x1 [8, 16, 32, 64]x3
ResNet50	ImageNet	33.00%	Pretrained-100	256	SGD(lr=0.01, momentum=0.9, weight_decay=1e-4) StepLR(step_size=30, gamma=0.1)	All layers [4, 8, 16, 32]
VGG11	CIFAR10	43.75%	300-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	[8, 16, 32]x1 [16, 32, 64]x2 [32, 64, 128]x4
VGG13	CIFAR10	43.75%	300-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	[4, 8, 16]x2 [8, 16, 32]x2 [16, 32, 64]x2 [32, 64, 128]x3
VGG16	CIFAR10	43.75%	300-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	[4, 8, 16]x2 [8, 16, 32]x3 [16, 32, 64]x3 [32, 64, 128]x4
DenseNet40	CIFAR10	33.33% 50.00% 66.67%	Pretrained-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=1e-4) MultiStepLR(milestones=[150, 225], gamma=0.1)	[4, 6, 12] [7, 8, 12, 14, 21, 24] [8, 12, 13, 24]
DenseNet40	CIFAR10	52.00%	400-300	64	SGD(lr=0.01, momentum=0.9, weight_decay=1e-4) MultiStepLR(milestones=[150, 225], gamma=0.1)	[4, 6, 12] [7, 8, 12, 14, 21, 24] [8, 12, 13, 24]
MobileNetV2	CIFAR10	40.00%	300-300	64	SGD(lr=0.001, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	[2, 4, 8] [4, 8, 12, 24] [4, 8, 16] [8, 16, 32, 64] [8, 16, 32]
WideResNet-28-10	CIFAR10	50.00%	200-300	128	SGD(lr=0.01, momentum=0.9, weight_decay=5e-4) StepLR(step_size=100, gamma=0.1)	[8, 16, 32]

Note in Table 5 above we have S1 and S2 as candidate group count settings for BasicBlock ResNets on CIFAR10. In such cases, S1 stands for [4, 8, 16], [8, 16, 32], [16, 32, 64], where S2 stands for [8, 16], [8, 16, 32], [8, 16, 32, 64]. Table 6 reports the evaluation results when the models are grouped/pruned according to different candidate group count settings. Under most evaluated setups, our method’s performance is similar between the two settings.

Table 6. Basicblock ResNets on CIFAR10 when pruned according to setting S1 and S2.

Model	S1: PR = 43.75%	S2: PR = 43.75%	S1: PR = 62.50%	S2: PR = 62.50%
ResNet20	92.14	<b>92.49</b>	-	-
ResNet32	<b>93.01</b>	92.96	<b>92.40</b>	92.07
ResNet56	93.93	<b>94.00</b>	93.32	<b>93.54</b>
ResNet110	<b>94.92</b>	94.54	<b>94.35</b>	94.34

### 10.3. Pruning procedure speedup

We additionally investigate the wall-clock runtime of our proposed method at Table 7 regarding its pruning procedure (time required to obtain a ready-to-fine-tune pruned model in grouped convolution format).



Table 7. Wall-clock Runtime Comparison between LeanFlex-GKP (Ours) and TMI-GKP [60], LRF [23]’s Pruning Procedures. **N/A** means no result.

Method	ResNet32	ResNet56	ResNet110	WideResNet-28-10
LRF [23]	40m 37s	1h 49m 31s	3h 6m 9s	N/A
TMI-GKP [60]	1h 20m 10s	2h 36m 22s	5h 30m 18s	<b>over 240 hours</b>
<b>LeanFlex-GKP (ours)</b>	<b>10m 56s</b>	21m 32s	<b>44m 15s</b>	<b>3h 10m 28s</b>

Our method provides a massive speed advantage over TMI-GKP while both being post-train, one-shot, and data-agnostic.

#### 10.4. Inference speedup: standard convolution v.s. grouped convolution

Table 8. Inference speed comparison between a standard convolution operator and a GKP convolution operator. Input size set as (64, in\_channel, 32, 32).

Operator	Forward	Macs	Params
Unpruned Standard Conv2d	2.50 ms	118.380 G	2.359 M
GKP-pruned Conv2d (G=2, PR=0.5)	1.61 ms	59.190 G	1.180 M
GKP-pruned Conv2d (G=2, PR=0.625)	1.28 ms	44.393 G	0.885 M
GKP-pruned Conv2d (G=4, PR=0.5)	1.82 ms	59.190 G	1.180 M
GKP-pruned Conv2d (G=4, PR=0.625)	1.46 ms	44.393 G	0.885 M

We must emphasize that while grouped convolution has many hardware-friendly properties such as being dense and structured, **in practice, not all shapes of group convolution operations provide the same amount of speedup — some shapes can even be slower.** Ever since the development of `torch 2.0` and `torch.compile`, we have observed a significant inference speed improvement on grouped convolution (previously, all shapes of grouped convolution were strictly slower). However, many aspects of this operation are still under-optimized without proper kernel implementations. There are, of course, many more inference frameworks than the vanilla torch, and they may exhibit different shape preferences and properties. We leave such systematic aspects of grouped convolution for future works.

### 11. Extended experiment results

Our proposed method, LeanFlex-GKP, follows the classic *train - prune - fine-tune* pipeline under a data-agnostic setting. This implies all model components are pruned all at once prior to fine-tuning, without having access to the training or fine-tuning data. Our method is implemented in a hard pruning fashion, which means the pruned model for fine-tuning is already compressed. We refer our readers to Table 5 for specific experiment details such as epoch budget and hyperparameter settings, as we have thereat documented detailed experiment settings for all 21 reported results of LeanFlex-GKP.

As introduced in Section 5, we evaluate the effectiveness of our method against many other densely structured pruning methods on ResNet20/32/56/110 with the BasicBlock, ResNet50/101 with the BottleNeck implementation [16], VGG11/13/16 [46], DenseNet40 [22], MobileNetV2 [45], and WideResNet [56]. The datasets we used include CIFAR10/100 [25], Tiny-ImageNet [52], and ImageNet-1k [4].

#### 11.1. Compared methods

Our methods is compared against 25 different pruning methods as illustrated in Table 9. Where notions like C/F/GK/K/L in the **Granularity** column respectively represent Channel/Filter/Grouped Kernel/Kernel/Layer pruning. **Procedure** indicates if the pruned model is generated iteratively (requires weight update between conducting the first pruning act and having the fully pruned model) or in a one-shot manner (pruned all at once without weight update in between). **Zero-Masked?** column investigates if a model is hard pruned (no zero-masked weight) before fine-tuning.



Table 9. Overview of Compared Methods.

Method	Venue	Granularity	Procedure	Zero-Masked?
CC [30]	CVPR	C	One-shot	N
DepGraph [8]	CVPR	C	One-shot	N
DHP [29]	ECCV	F	Iterative (from-scratch)	Y
FPGM [20]	CVPR	F	Iterative	Y
GAL [33]	CVPR	F	Iterative	Y
HRank [32]	CVPR	F	Iterative	Y
L1Norm [28]	ICLR	F	One-shot	N
LRF [23]	AAAI	C	One-shot	N
NPPM [11]	CVPR	C	One-shot	N
OTOv2 [3]	ICLR	F	Iterative (from-scratch)	Y
SFP [18]	IJCAI	F	Iterative	Y
ThiNet [37]	ICCV	F	One-shot	-
TMI-GKP [60]	ICLR	GK	One-shot	N
<b>LeanFlex-GKP (Ours)</b>	-	GK	One-shot	N
AAP [59]	AISTATS	F	Iterative	-
AMC [19]	ECCV	C	-	N
ChipNet [48]	ICLR	C	Iterative	N
GDP [15]	ICCV	C	One-shot	-
DOP [54]	BMVC	C	One-shot	Y
KPGP [57]	APIN	GK	One-shot	N
WM [64]	NeurIPS	C	Iterative	-
Layer-wise Proxy [7]	IEEE ICIP	L	One-shot	N
DCP [64]	NeurIPS	C	Iterative	-
DMC [10]	CVPR	C	Iterative	-
MDP [12]	CVPR	C	-	-
SCOP [47]	NeurIPS	F	-	-

## 11.2. Accuracy gap between competitive methods and LeanFlex-GKP

Table 10. The accuracy gap between LeanFlex-GKP and several competitive methods: TMI-GKP [60], LRF [23], CC [30], NPPM [11], SFP [18]

Method Setting	CC	TMI	NPPM	LRF	SFP
ResNet56 TinyImageNet	<b>+4.71</b>	+0.15	<b>+5.18</b>	-0.28	<b>+2.02</b>
ResNet101 TinyImageNet		<b>+3.61</b>		<b>+0.66</b>	+0.40
ResNet50 Imagenet		+0.11			<b>+17.12</b>
VGG11	+0.31				
VGG13	+0.06				
VGG16	+0.01	0.08			
DenseNet40 (pr=0.33)	+0.24				
DenseNet40 (pr=0.50)	+0.35				
DenseNet40 (pr=0.52)	<b>+0.50</b>				
DenseNet40 (pr=0.67)		+0.35			
ResNet56 CIFAR100	<b>+0.68</b>	+0.32	<b>+0.54</b>		<b>+2.31</b>
ResNet110 CIFAR100	+0.42	<b>+0.63</b>	<b>+1.25</b>	+0.05	
ResNet32 (pr=0.625)	+0.01		+0.48	-0.39	<b>+2.12</b>
ResNet56 (pr=0.625)	-0.03		+0.47		<b>+1.30</b>
ResNet110 (pr=0.625)	+0.06		+0.42	+0.25	<b>+1.37</b>
ResNet20 (pr=0.4375)	<b>+0.69</b>	+0.31	<b>+0.63</b>	+0.26	<b>+1.34</b>
ResNet32 (pr=0.4375)	+0.00	+0.02	-0.12	-0.03	<b>+1.07</b>
ResNet56 (pr=0.4375)	-0.04	+0.05	+0.45	+0.07	<b>+0.85</b>
ResNet110 (pr=0.4375)	<b>+0.61</b>	+0.02	<b>+0.76</b>	+0.43	+0.48

## 11.3. Margin of error

We additionally provide the margin of error of Table 1.

Table 11. Performance of LeanFlex-GKP for multiple runs of fine-tuning.

Method s- Dataset	Baseline	Acc Reported	Avg Acc ( $\pm$ Margin)	Pruning Rate
VGG16 - CIFAR10	93.94	94.15	94.06 ( $\pm 0.09$ )	0.4375
ResNet32 - CIFAR10	92.80	92.40	92.31 ( $\pm 0.09$ )	0.625
ResNet110 - CIFAR10	94.26	94.35	94.26 ( $\pm 0.09$ )	0.625
MobileNetV2 - CIFAR10	93.87	94.36	94.21 ( $\pm 0.15$ )	0.5
ResNet56 - CIFAR100	71.53	72.11	72.24 ( $\pm 0.21$ )	0.4375
ResNet110 - CIFAR100	73.20	73.63	73.45 ( $\pm 0.17$ )	0.4375
ResNet56 - TinyImagenet	56.13	55.67	55.68 ( $\pm 0.14$ )	0.375

## 11.4. Stable Diffusion Conv2d Pruning Result

In addition to image classification tasks, we also evaluate LeanFlex-GKP’s performance on the image generation (diffusion) task. Specifically, we apply LeanFlex-GKP to the UNet component of SDXL-Base-1.0 [43]. We exclusively fine-tuned the LeanFlex-GKP pruned SDXL-Base-1.0 while freezing all other non-UNet components for 30 epochs on the MSCOCO-2014-5k [35], then measured its output with 20 GPT-4 [40] generated prompts via the CLIP score [21]. Our method demonstrates decent — beyond unpruned — performance with 50% pruned (Table 12).

Table 12. SDXL-Base-1.0 w/ UNet pruned by LeanFlex-GKP

Model	Max CLIP Score	Avg CLIP Score
Baseline (UNet unpruned)	30.54	27.99
LeanFlex-GKP (PR = 50%)	36.98 ( $\uparrow$ 6.44)	28.60 ( $\uparrow$ 0.61)

## 11.5. Full experiment results

The terms and notations utilized in the following experiment results follow the definitions defined in Section 5: **DA** represents if the method is data-agnostic (pruning can be done without access to data), **IP** indicates if a method is considered an iterative pruning method (utilizing a train-prune cycle), and **RB** reports recovery budget (in terms of epochs). All other reported criteria are in terms of %. **BA** and **Pruned** respectively report the unpruned (baseline) accuracy and the pruned accuracy. Methods marked with \* are drawn from their original or (third-party) replicated publication; the rest are replicated by us to ensure a fair comparison. Generally speaking, a method that is **DA** ✓, **IP** ✗, and demands a smaller **RB** is likely to be more user-friendly.

Table 13. Results of ResNet50 Model on ImageNet-1K Dataset. Results in **bold red** indicate being the second best among comparisons.

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	↓ MACs	↓ Params
<b>ResNet50 on ImageNet-1K</b>			MACs $\approx$ 4122.828M		Params $\approx$ 25.557M			
SFP* [18]	✗	✓	100	76.13	58.50	↓ 17.63	36.08	32.31
FPGM* [20]	✗	✓	100	76.13	75.04	↓ 1.09	35.93	28.36
TMI-GKP* [60]	✓	✗	100	76.15	75.53	↓ 0.62	33.21	33.74
ThiNet* [37]	✗	✓	100	72.88	72.04	↓ 0.84	36.70	-
OTov2* [3]	✗	✓	120	76.13	75.38	↓ 0.75	37.70	26.58
(post-train)								
DOP* [54]	✗	✗	120	76.47	74.29	↓ 2.18	60.00	-
Layer-wise Proxy* [7]	✗	✗	-	76.14	75.0	↓ 1.14	5.50	-
KPGP* [57]	✓	✗		76.15	75.50	↓ 0.65	33.70	33.20
<b>LeanFlex-GKP (ours)</b>	✓	✗	100	76.13	<b>75.62</b>	<b>↓ 0.51</b>	33.06	30.34

Table 14. Results of ResNet56/101 Model on Tiny-ImageNet Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	↓ MACs	↓ Params
<b>ResNet56 on Tiny-ImageNet</b>			MACs $\approx$ 506.254M		Params $\approx$ 0.865M			
TMI-GKP [60]	✓	✗	300	56.13	55.52	↓ 0.61	37.05	36.76
L1Norm-A [28]	✓	✗	300	56.13	55.41	↓ 0.72	35.51	32.14
L1Norm-B [28]	✓	✗	300	56.13	55.21	↓ 0.92	36.43	41.04
SFP [18]	✗	✓	300	56.13	53.65	↓ 2.48	33.96	35.38
FPGM [20]	✗	✓	300	56.13	54.14	↓ 1.99	33.53	34.68
HRank [32]	✗	✓	300	56.13	54.16	↓ 1.97	37.39	30.98
DHP [29]	✗	✓	100	56.13	45.73	↓ 10.40	36.42	-
NPPM [11]	✗	✗	300	56.13	50.49	↓ 5.64	36.42	17.92
LRF [23]	✗	✗	300	56.13	<b>55.95</b>	<b>↓ 0.18</b>	35.90	34.68
CC [30]	✗	✗	300	56.13	50.96	↓ 5.17	22.18	14.34
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	56.13	<b>55.67</b>	<b>↓ 0.46</b>	37.05	36.76
<b>ResNet101 on Tiny-ImageNet</b>			MACs $\approx$ 10081.092M		Params $\approx$ 42.902M			
TMI-GKP [60]	✓	✗	20	65.71	65.05	↓ 0.66	43.25	43.53
SR-GKP [61]	✓	✗	20	65.51	67.21	↑ 1.70	43.25	43.53
SFP [18]	✗	✓	20	65.51	68.06	↑ 2.55	43.56	42.43
FPGM [20]	✗	✓	20	65.51	66.95	↑ 1.44	43.13	43.84
<b>LeanFlex-GKP (ours)</b>	✓	✗	20	65.51	<b>68.46</b>	<b>↑ 2.95</b>	43.25	43.53

Table 15. Results of MobileNetV2 Model on CIFAR10 Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	$\downarrow$ MACs	$\downarrow$ Params
<b>MobileNetV2 on CIFAR10</b>			MACs $\approx$ 98.768M		Params $\approx$ 2.383M			
DCP* [64]	$\times$	-	400	94.47	94.69	$\uparrow$ 0.22	26.00	-
WM* [64]	$\times$	-	400	94.47	94.17	$\downarrow$ 0.30	26.00	-
MDP* [12]	$\times$	-	-	95.02	95.14	$\uparrow$ 0.12	28.71	-
ChipNet* [48]	$\times$	$\checkmark$	300	93.55	92.58	$\downarrow$ 0.97	20.00	-
SCOP* [47]	$\times$	-	400	94.48	94.24	$\downarrow$ 0.24	49.30	-
GDP* [15]	$\times$	-	350	94.89	<b>95.15</b>	$\uparrow$ 0.26	46.22	-
DMC* [10]	$\times$	-	160	94.23	94.49	$\uparrow$ 0.26	40.00	-
<b>LeanFlex-GKP (ours)</b>	$\checkmark$	$\times$	300	93.87	94.36	$\uparrow$ <b>0.49</b>	38.32	36.00

Table 16. Results of VGG11/13/16 Model on CIFAR10 Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	$\downarrow$ MACs	$\downarrow$ Params
<b>VGG11 on CIFAR10</b>			MACs $\approx$ 153.5M		Params $\approx$ 9.3M			
CC [30]	$\times$	$\times$	300	92.34	92.24	$\downarrow$ 0.10	42.32	56.77
L1Norm [28]	$\checkmark$	$\times$	300	92.34	91.77	$\downarrow$ 0.57	41.44	35.01
<b>LeanFlex-GKP (ours)</b>	$\checkmark$	$\times$	300	92.34	<b>92.55</b>	$\uparrow$ <b>0.21</b>	43.41	43.68
<b>VGG13 on CIFAR10</b>			MACs $\approx$ 229.4M		Params $\approx$ 9.4M			
CC [30]	$\times$	$\times$	300	93.95	93.97	$\uparrow$ 0.02	42.56	54.11
L1Norm [28]	$\checkmark$	$\times$	300	93.95	93.26	$\downarrow$ 0.69	42.95	35.09
<b>LeanFlex-GKP (ours)</b>	$\checkmark$	$\times$	300	93.95	<b>94.03</b>	$\uparrow$ <b>0.08</b>	43.58	43.68
<b>VGG16 on CIFAR10</b>			MACs $\approx$ 313.4M		Params $\approx$ 14.7M			
CC [30]	$\times$	$\times$	300	93.94	94.14	$\uparrow$ 0.20	43.18	-
GAL [34]	$\times$	$\checkmark$	300	93.94	91.29	$\downarrow$ 2.65	35.16	47.40
HRank [32]	$\times$	$\checkmark$	300	93.94	93.57	$\downarrow$ 0.37	32.28	40.82
L1Norm [28]	$\checkmark$	$\times$	300	93.94	92.88	$\downarrow$ 1.06	42.71	37.85
KPGP* [57]	$\checkmark$	$\times$	300	94.27	<b>94.17</b>	$\downarrow$ 0.10	43.15	43.59
TMI-GKP [60]	$\checkmark$	$\times$	300	93.94	94.07	$\uparrow$ 0.13	43.15	43.59
<b>LeanFlex-GKP (ours)</b>	$\checkmark$	$\times$	300	93.94	<b>94.15</b>	$\uparrow$ <b>0.21</b>	43.15	43.59

Table 17. Results of DenseNet40 on CIFAR10 Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	$\downarrow$ MACs	$\downarrow$ Params
<b>DenseNet40 on CIFAR10</b>			MACs $\approx$ 282.2M		Params $\approx$ 1.5M			
GAL* [34]	$\times$	$\checkmark$	-	94.81	94.61	$\downarrow$ 0.20	35.30	35.60
HRank* [32]	$\times$	$\checkmark$	-	94.81	94.24	$\downarrow$ 0.57	41.00	36.50
CC ( $pr=0.33$ ) [30]	$\times$	$\times$	300	94.81	94.75	$\downarrow$ 0.06	32.97	51.42
CC ( $pr=0.50$ ) [30]	$\times$	$\times$	300	94.81	94.58	$\downarrow$ 0.23	49.85	64.48
CC ( $pr=0.67$ ) [30]	$\times$	$\times$	300	94.81	94.22	$\downarrow$ 0.59	66.55	75.88
<b>LeanFlex-GKP (<math>pr=0.33</math>) (ours)</b>	$\checkmark$	$\times$	300	94.81	<b>94.99</b>	$\uparrow$ <b>0.18</b>	32.43	32.58
<b>LeanFlex-GKP (<math>pr=0.50</math>) (ours)</b>	$\checkmark$	$\times$	300	94.81	<b>94.93</b>	$\uparrow$ <b>0.12</b>	48.64	48.82
<b>LeanFlex-GKP (<math>pr=0.67</math>) (ours)</b>	$\checkmark$	$\times$	300	94.81	<b>94.72</b>	$\downarrow$ <b>0.09</b>	64.85	65.16
TMI-GKP ( $pr=0.52$ ) [60]	$\checkmark$	$\times$	300	94.66	94.76	$\uparrow$ 0.10	52.49	57.22
<b>LeanFlex-GKP (<math>pr=0.52</math>) (ours)</b>	$\checkmark$	$\times$	300	94.66	<b>95.11</b>	$\uparrow$ <b>0.45</b>	52.49	57.22

Table 18. Results of WideResNet Model on CIFAR10 Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	$\downarrow$ MACs	$\downarrow$ Params
<b>WideResNet-28-10 on CIFAR10</b>			MACs $\approx$ 5959.4M		Params $\approx$ 36.5M			
<b>LeanFlex-GKP (<math>pr=0.5</math>) (ours)</b>	$\checkmark$	$\times$	300	95.09	<b>95.97</b>	$\uparrow$ <b>0.88</b>	49.73	49.62

Table 19. Results of ResNet32/56/110 on CIFAR10 dataset with a pruning rate of  $\approx 62.5\%$ 

Method	DA	IP	RB	BA	Pruned	$\Delta\text{Acc}$	$\downarrow \text{MACs}$	$\downarrow \text{Params}$
<b>ResNet32 on CIFAR10</b>			MACs $\approx 69.5\text{M}$		Params $\approx 0.46\text{M}$			
L1Norm-A [28]	✓	✗	300	92.80	89.96	$\downarrow 2.84$	61.86	65.21
L1Norm-B [28]	✓	✗	300	92.80	90.01	$\downarrow 2.79$	62.36	67.39
CC [30]	✗	✗	300	92.80	92.39	$\downarrow 0.41$	61.29	54.35
SFP [18]	✗	✓	300	92.80	90.28	$\downarrow 2.52$	59.74	60.65
FPGM [20]	✗	✓	300	92.80	91.32	$\downarrow 1.48$	58.28	59.57
NPPM [11]	✗	✗	300	92.80	91.92	$\downarrow 0.88$	61.15	56.52
DHP [29]	✗	✓	300	92.80	91.73	$\downarrow 1.07$	50.92	-
LRF [23]	✗	✗	300	92.80	<b>92.79</b>	$\downarrow \mathbf{0.01}$	56.95	56.52
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	92.80	<b>92.40</b>	$\downarrow \mathbf{0.40}$	61.56	61.74
<b>ResNet56 on CIFAR10</b>			MACs $\approx 126.6\text{M}$		Params $\approx 0.85\text{M}$			
L1Norm-A [28]	✓	✗	300	93.24	91.79	$\downarrow 1.45$	62.43	57.64
L1Norm-B [28]	✓	✗	300	93.24	91.56	$\downarrow 1.68$	62.25	62.35
CC [30]	✗	✗	300	93.24	<b>93.57</b>	$\uparrow \mathbf{0.33}$	61.54	50.58
SFP [18]	✗	✓	300	93.24	92.24	$\downarrow 1.00$	58.61	60.24
FPGM [20]	✗	✓	300	93.24	92.64	$\downarrow 0.60$	58.33	59.88
NPPM [11]	✗	✗	300	93.24	93.07	$\downarrow 0.17$	58.49	47.05
HRank [32]	✗	✓	300	93.24	90.63	$\downarrow 2.61$	60.56	51.88
DHP [29]	✗	✓	300	93.24	91.66	$\downarrow 1.58$	60.54	-
AAP* [59]	-	-	-	92.84	92.21	$\downarrow 0.63$	52.72	-
AMC* [19]	-	-	-	92.80	91.90	$\downarrow 0.90$	50.00	-
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	93.24	<b>93.54</b>	$\uparrow \mathbf{0.30}$	61.76	61.99
<b>ResNet110 on CIFAR10</b>			MACs $\approx 255.0\text{M}$		Params $\approx 1.73\text{M}$			
L1Norm-A [28]	✓	✗	300	94.26	92.50	$\downarrow 1.76$	61.58	64.16
L1Norm-B [28]	✓	✗	300	94.26	94.04	$\downarrow 0.22$	60.29	72.25
CC [30]	✗	✗	300	94.26	94.29	$\uparrow 0.03$	61.34	58.38
SFP [18]	✗	✓	300	94.26	92.98	$\downarrow 1.28$	58.70	60.29
FPGM [20]	✗	✓	300	94.26	94.11	$\downarrow 0.15$	58.35	60.17
NPPM [11]	✗	✗	300	94.26	93.93	$\downarrow 0.33$	60.81	56.87
HRank [32]	✗	✓	300	94.26	91.94	$\downarrow 2.32$	61.90	62.49
DHP [29]	✗	✓	300	94.26	92.73	$\downarrow 1.53$	74.16	-
LRF [23]	✗	✗	300	94.26	94.10	$\downarrow 0.16$	62.94	63.12
ChipNet* [48]	✗	✓	300	93.98	93.78	$\downarrow 0.20$	62.41	-
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	94.26	<b>94.35</b>	$\uparrow \mathbf{0.09}$	64.22	62.19

Table 20. Results of ResNet56/110 on CIFAR100 Dataset

Method	DA	IP	RB	BA	Pruned	$\Delta$ Acc	$\downarrow$ MACs	$\downarrow$ Params
<b>ResNet56 on CIFAR100</b>			MACs $\approx$ 126.567M			Params $\approx$ 0.859M		
TMI-GKP [60]	✓	✗	300	70.85	71.11	$\uparrow$ 0.26	43.22	43.19
L1Norm-A [28]	✓	✗	300	71.53	68.61	$\downarrow$ 2.92	43.05	40.86
L1Norm-B [28]	✓	✗	300	71.53	68.32	$\downarrow$ 3.21	42.16	48.20
CC [30]	✗	✗	300	71.53	71.43	$\downarrow$ 0.10	43.52	28.52
SFP [18]	✗	✓	300	71.53	69.80	$\downarrow$ 1.73	44.29	44.82
FPGM [20]	✗	✓	300	71.53	69.48	$\downarrow$ 2.05	43.38	43.19
DHP [29]	✗	✓	300	71.53	68.33	$\downarrow$ 3.2	40.91	29.14
NPPM [11]	✗	✗	300	71.53	71.57	$\uparrow$ 0.04	33.54	13.04
HRank [32]	✗	✓	300	71.53	69.84	$\downarrow$ 1.69	37.39	31.32
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	71.53	<b>72.11</b>	<b><math>\uparrow</math> 0.58</b>	43.22	43.18
<b>ResNet110 on CIFAR100</b>			MACs $\approx$ 255.001M			Params $\approx$ 1.734M		
TMI-GKP [60]	✓	✗	300	72.99	72.79	$\downarrow$ 0.20	43.31	43.37
L1Norm-A [28]	✓	✗	300	73.20	69.85	$\downarrow$ 3.35	43.74	44.41
L1Norm-B [28]	✓	✗	300	73.20	69.32	$\downarrow$ 3.88	42.22	51.96
CC [30]	✗	✗	300	73.20	73.21	$\uparrow$ 0.01	43.43	19.78
NPPM [11]	✗	✗	300	73.20	72.38	$\downarrow$ 0.82	42.77	18.69
LRF [23]	✗	✗	300	73.20	73.58	$\uparrow$ 0.38	43.38	42.16
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	73.20	<b>73.63</b>	<b><math>\uparrow</math> 0.43</b>	43.31	43.36

Table 21. Results of ResNet20/32/56/110 on CIFAR10 dataset with a pruning rate of  $\approx 43.75\%$ . Results in **bold red** indicate being the second best among comparisons.

Method	DA	IP	RB	BA	Pruned	$\Delta\text{Acc}$	$\downarrow \text{MACs}$	$\downarrow \text{Params}$
<b>ResNet20 on CIFAR10</b>			MACs $\approx 40.9\text{M}$		Params $\approx 0.27\text{M}$			
TMI-GKP [60]	✓	✗	300	91.99	92.18	$\uparrow 0.19$	42.86	43.33
L1Norm-A [28]	✓	✗	300	91.99	90.54	$\downarrow 1.45$	43.11	35.19
L1Norm-B [28]	✓	✗	300	91.99	90.83	$\downarrow 1.16$	43.87	19.63
CC [30]	✗	✗	300	91.99	91.80	$\downarrow 0.19$	43.47	36.30
SFP [18]	✗	✓	300	91.99	91.15	$\downarrow 0.84$	40.32	41.85
FPGM [20]	✗	✓	300	91.99	91.51	$\downarrow 0.48$	43.34	43.33
NPPM [11]	✗	✗	300	91.99	91.86	$\downarrow 0.13$	43.49	35.19
LRF [23]	✗	✗	300	91.99	92.23	$\uparrow 0.24$	43.08	43.70
DepGraph [8]	✓	✗	300	91.99	91.38	$\downarrow 0.61$	42.96	41.11
KPGP* [57]	✓	✗	300	92.46	92.10	$\downarrow 0.36$	55.10	55.70
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	91.99	<b>92.49</b>	$\uparrow 0.50$	42.86	43.33
<b>ResNet32 on CIFAR10</b>			MACs $\approx 69.5\text{M}$		Params $\approx 0.46\text{M}$			
TMI-GKP [60]	✓	✗	300	92.80	92.99	$\uparrow 0.19$	43.08	43.32
L1Norm-A [28]	✓	✗	300	92.80	91.45	$\downarrow 1.35$	42.63	45.69
L1Norm-B [28]	✓	✗	300	92.80	91.58	$\downarrow 1.22$	42.96	32.54
CC [30]	✗	✗	300	92.80	93.01	$\uparrow 0.21$	43.49	32.76
SFP [18]	✗	✓	300	92.80	91.94	$\downarrow 0.86$	41.89	42.67
FPGM [20]	✗	✓	300	92.80	92.41	$\downarrow 0.39$	43.36	43.53
NPPM [11]	✗	✗	300	92.80	<b>93.13</b>	$\uparrow 0.33$	43.00	29.74
DHP [29]	✗	✓	300	92.80	92.26	$\downarrow 0.54$	42.30	39.01
LRF [23]	✗	✗	300	92.80	93.04	$\uparrow 0.24$	44.17	43.97
DepGraph [8]	✓	✗	300	92.80	93.04	$\uparrow 0.24$	40.79	33.26
OTOv2 [3]	✗	✓	300	92.80	90.97	$\downarrow 1.83$	38.28	44.77
OTOv2 (post-train) [3]	✗	✓	300	92.80	92.14	$\downarrow 0.66$	49.77	35.80
KPGP* [57]	✓	✗	300	92.71	92.68	$\downarrow 0.03$	43.1	43.4
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	92.80	93.01	$\uparrow 0.21$	43.08	43.32
<b>ResNet56 on CIFAR10</b>			MACs $\approx 126.6\text{M}$		Params $\approx 0.85\text{M}$			
TMI-GKP [60]	✓	✗	300	93.24	93.95	$\uparrow 0.71$	43.23	43.49
L1Norm-A [28]	✓	✗	300	93.24	92.44	$\downarrow 0.80$	46.27	42.91
L1Norm-B [28]	✓	✗	300	93.24	92.62	$\downarrow 0.62$	43.02	31.30
CC [30]	✗	✗	300	93.24	<b>94.04</b>	$\uparrow 0.80$	44.82	27.78
SFP [18]	✗	✓	300	93.24	93.15	$\downarrow 0.09$	43.54	43.61
GAL [33]	✗	✓	300	93.24	91.27	$\downarrow 1.97$	22.38	17.94
FPGM [20]	✗	✓	300	93.24	93.60	$\uparrow 0.36$	43.38	43.49
NPPM [11]	✗	✗	300	93.24	93.55	$\uparrow 0.21$	44.02	29.54
HRank [32]	✗	✓	300	93.24	92.27	$\downarrow 0.97$	37.39	31.54
DHP [29]	✗	✓	300	93.24	92.42	$\downarrow 0.82$	42.09	43.73
LRF [23]	✗	✗	300	93.24	93.93	$\uparrow 0.69$	43.89	42.56
DepGraph [8]	✓	✗	300	93.24	93.79	$\uparrow 0.55$	39.82	26.71
OTOv2 [3]	✗	✓	300	93.24	91.57	$\downarrow 1.67$	36.96	43.70
OTOv2 (post-train) [3]	✗	✓	300	93.24	93.02	$\downarrow 0.22$	47.70	35.01
KPGP* [57]	✓	✗	300	93.75	93.72	$\downarrow 0.03$	43.20	43.50
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	93.24	<b>94.00</b>	$\uparrow 0.76$	43.23	43.49
<b>ResNet110 on CIFAR10</b>			MACs $\approx 255.0\text{M}$		Params $\approx 1.73\text{M}$			
TMI-GKP [60]	✓	✗	300	94.26	94.90	$\uparrow 0.64$	43.31	43.52
L1Norm-A [28]	✓	✗	300	94.26	92.75	$\downarrow 1.51$	43.74	44.56
L1Norm-B [28]	✓	✗	300	94.26	92.96	$\downarrow 1.30$	43.17	36.69
CC [30]	✗	✗	300	94.26	94.31	$\uparrow 0.05$	44.54	39.47
SFP [18]	✗	✓	300	94.26	94.44	$\uparrow 0.18$	43.42	43.52
GAL [33]	✗	✓	300	94.26	93.42	$\downarrow 0.84$	29.14	31.37
FPGM [20]	✗	✓	300	94.26	94.18	$\downarrow 0.08$	43.39	43.52
NPPM [11]	✗	✗	300	94.26	94.16	$\downarrow 0.10$	42.46	35.19
HRank [32]	✗	✓	300	94.26	92.96	$\downarrow 1.30$	18.57	5.38
DHP [29]	✗	✓	300	94.26	92.53	$\downarrow 1.73$	60.25	64.58
LRF [23]	✗	✗	300	94.26	94.49	$\uparrow 0.23$	43.37	42.30
OTOv2 [3]	✗	✓	300	94.26	91.58	$\downarrow 2.68$	37.83	42.44
OTOv2 (post-train) [3]	✗	✓	300	94.26	93.99	$\downarrow 0.27$	38.11	42.50
KPGP* [57]	✓	✗	300	93.76	94.01	$\uparrow 0.25$	43.30	43.50
<b>LeanFlex-GKP (ours)</b>	✓	✗	300	94.26	<b>94.92</b>	$\uparrow 0.66$	43.31	43.52