

---

# Structurally Prune Anything: Any Architecture, Any Framework, Any Time

---

Xun Wang<sup>1\*</sup> John Rachwan<sup>2\*</sup> Stephan Günnemann<sup>23</sup> Bertrand Charpentier<sup>2</sup>

<sup>1</sup>CISPA Helmholtz Center for Information Security <sup>2</sup>Pruna AI

<sup>3</sup>Department of Computer Science & Munich Data Science Institute, Technical University of Munich  
xun.wang@cispa.de

{john.rachwan,stephan.guennemann,bertrand.charpentier}@pruna.ai

## Abstract

Neural network pruning serves as a critical technique for enhancing the efficiency of deep learning models. Unlike unstructured pruning, which only sets specific parameters to zero, structured pruning eliminates entire channels, thus yielding direct computational and storage benefits. However, the diverse patterns for coupling parameters, such as residual connections and group convolutions, the diverse deep learning frameworks, and the various time stages at which pruning can be performed make existing pruning methods less adaptable to different architectures, frameworks, and pruning criteria. To address this, we introduce Structurally Prune Anything (SPA), a versatile structured pruning framework that can prune neural networks with any architecture, from any framework, and at any stage of training. SPA leverages a standardized computational graph and ONNX representation to prune diverse neural network architectures without the need for manual intervention. SPA employs a group-level importance estimation method, which groups dependent computational operators, estimates their importance, and prunes unimportant coupled channels. This enables the transfer of various existing pruning criteria into a structured group style. As a result, SPA supports pruning at any time, either before training, after training with fine-tuning, or after training without fine-tuning. In the context of the latter, we introduce Optimal Brain SPA (OBSPA), an algorithm that achieves state-of-the-art pruning results needing neither fine-tuning nor calibration data. In extensive experiments, SPA shows competitive to state-of-the-art pruning performance across various architectures, from popular frameworks, at different pruning times.

## 1 Introduction

The increasing complexity and scale of deep learning models He et al. (2015); Simonyan & Zisserman (2015); Dosovitskiy et al. (2020) have sparked significant research interest in compression methods. Compression methods, like pruning, aim to reduce model size and computational cost in order to increase inference speed, save energy, and enable deployment on computationally limited devices. In particular, pruning methods mostly fall into two main categories: unstructured pruning which involves setting specific parameters to zero while maintaining the overall network structure LeCun et al. (1989); Hassibi & Stork (1992); Dong et al. (2017); Han et al. (2015); Lee et al. (2019); Frantar et al. (2022); Xiao et al. (2019), and structured pruning which involves removing entire channels Li et al. (2016); He et al. (2018b, 2017); Lin et al. (2020); Liu et al. (2017a); Rachwan et al. (2022). While structured pruning advantageously results in direct computational and memory reduction, it is

---

\*equal contribution

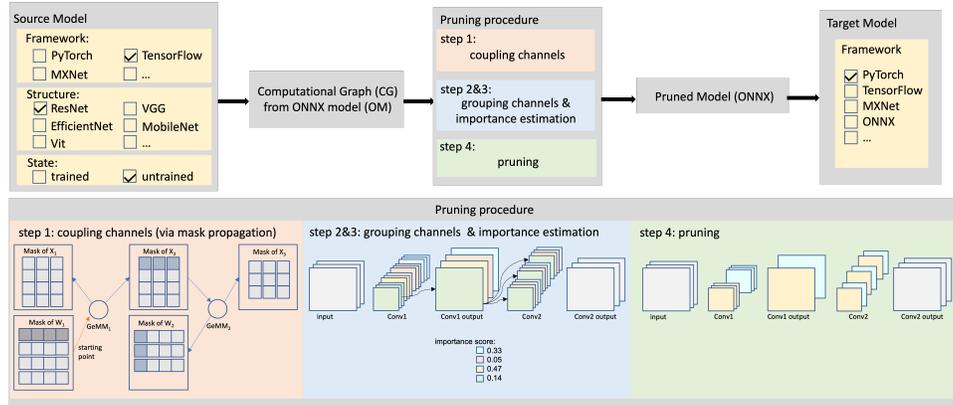


Figure 1: SPA overview. The source model can be chosen freely from different frameworks with different structures, either trained or not. A computational graph is built to store the dependency information between operators and data. The pruning procedure consists of four steps: coupling channels, grouping channels & importance estimation, and pruning. After pruning, the pruned model can be converted to other frameworks for further usage.

considered a more complex undertaking. Specifically, structured pruning methods often come with three main challenges.

**Challenge 1:** The first major challenge consists of the difficulty of applying different structured pruning methods to various model architectures. Indeed, structured pruning entails managing the interdependencies between coupled channels in different layers to modify the model structure without breaking the model connectivity (e.g. see residual connection in Fig. 5). Hence, when dealing with coupled channels, most of the existing approaches heavily rely on case-by-case analysis of different model architectures.

**Challenge 2:** The second challenge consists of unifying structured pruning methods in a single framework making pruning possible at any stage of training. Pruning can be done either *before*, *during*, or *after* training. The majority of works adhere to the pruning with fine-tuning approach, which we will refer to as the *train-prune-finetune* setting, and involves conducting finetuning after pruning pre-trained models to restore any performance degradation incurred during the pruning process. Another approach consists in pruning a model before training, which we will refer to as the *prune-train* setting, thus allowing to directly train a sparse model. Nonetheless, a more challenging yet advantageous scenario is the pruning without fine-tuning setting, which we will refer to as the *train-prune* setting Lazarevich et al. (2021), wherein no additional training is permitted after pruning a pre-trained model. Instead, the train-prune setting has only access to a limited set of calibration data Frantar et al. (2022); Frantar & Alistarh (2023), or, even more challenging, has access to no calibration data Srinivas & Babu (2015) for the pruning step.

**Challenge 3:** The third challenge is that existing pruning methods are not only often designed with specific architectures or training paradigms in mind, but they are also further entrenched by the deep learning frameworks they were developed. This framework specificity arises due to several factors: differences in computational graph, definition of specific layers, and the existence of unique APIs and optimization libraries. As such, a pruning method effective in one setting may require non-trivial adaptations to be ported to another framework or architecture, complicating its general applicability. Hence, the third challenge lies in crafting an approach robust enough to transcend the limitations imposed by framework-specific constraints and progress toward a unified, generalizable approach to model pruning.

Previous works have tried to address parts of these three challenges. For instance, DepGraph Fang et al. (2023) and OTO-v2 Chen et al. (2023) enables the automatic pruning of different networks by maintaining a dependency graph. However, they lack the ability to support models other than PyTorch and only support pruning after training with or without fine-tuning scheme. Further, DFPC Narshana et al. (2023) proposed a method to prune coupled channels data-free without fine-tuning, but it lacks the ability to adapt to different architectures and frameworks.

To jointly tackle the aforementioned three challenges, we propose Structurally Prune Anything (SPA), an architecture-and-framework-agnostic neural network pruning method, which supports different criteria that encompass the previous three settings we defined. We show an overview of our method in Fig. 1. Its contributions can be summarized as follows. **(1) Prune Any Framework:** We directly operate on a flexible computational graph compatible across frameworks. To this end, we use the ONNX format. With this procedure, we are the first pruning method that can handle the most common deep learning frameworks. **(2) Prune Any Architecture:** We propose a four-step procedure for the structured pruning of grouped channels. This procedure allows automatic pruning of neural networks with any structures, and the easy transfer of many existing pruning criteria for a grouped structured version, often achieving superior performance/efficiency trade-off. **(3) Prune Any Time:** We propose a group-level importance estimation method, enabling pruning at any training stage including prune-train, train-prune-finetune, and train-prune. In the latter setting, we propose a *novel* method Optimal Brain SPA (OBSPA) which achieves state-of-the-art results with ResNet50 on CIFAR10 and VGG19 on CIFAR100 without the need for calibration data.

## 2 Related Works

**Pruning criteria:** To determine which connection or neuron should be pruned, various pruning criteria are employed to identify their importance. Most pruning research has followed the approach pioneered by Han et al. (2015) of using weight magnitudes as importance scores. These include Li et al. (2016); He et al. (2018a). However, the drawback of only using weight magnitudes is that the network has to be pre-trained in order for it to achieve good performance. Therefore, some approaches have focussed on augmenting them with first-order and second-order information, which allows for the pruning to be applied even on a randomly initialized network Lee et al. (2019); Verdenius et al. (2020); Wang et al. (2020); Rachwan et al. (2022). Most recently, due to the rise of generative models and their growing costs, pruning research has shifted its focus towards removing the need to fine-tune after pruning. These approaches generate importance scores by solving complex optimization problems that attempt to preserve the per-layer outputs of the model Frantar et al. (2022); Frantar & Alistarh (2023). We recommend interested readers to refer to the following surveys He & Xiao (2023); Blalock et al. (2020) for a more comprehensive overview of the previously discussed approaches as well as additional ones such as activation-based He et al. (2017); Jian-Hao Luo & Lin (2017); Lin et al. (2020); Yu et al. (2017); Zhuang et al. (2018), and regularization based Liu et al. (2017b); You et al. (2019); Huang & Wang (2017); Ding et al. (2021) variants.

**Pruning coupled channels:** Research on pruning coupled parameters has been a prominent area of focus since the initial stages of structural pruning, with techniques like slimming Liu et al. (2017a) and ThiNet Jian-Hao Luo & Lin (2017) aiming to identify and remove such dependencies. However, manually analyzing parameter inter-dependencies can be an exceedingly arduous task, particularly when applied to complex networks such as DenseNet Huang et al. (2016). Some works have emerged to discover the complex relationships between layers by automatically uncovering the dependencies between the layers. Group Fisher pruning Liu et al. (2021) introduces a versatile channel pruning approach applicable to complex structures by building the network’s dependency graph. DFPC Narshana et al. (2023) prunes the coupled channels in a one-shot and data-free manner, it introduces the concept of Data Flow Couplings (DFCs). DFCs are tuples that describe a set of layers and the transformations between them that couple the channels of the output of one layer to the channels of the input of another layer. Most recently, OTO-v2 Chen et al. (2023) and DepGraph Fang et al. (2023) also address the problem by building a dependency graph. On one hand, OTO-v2 traces the operator connectivity in CNNs, residual, or transformer architectures but requires a specific training routine with Zero-Invariant-Group partitions. On the other hand, DepGraph traces the model’s gradient functions in the backward pass to generalize to multiple architectures such as CNNs, RNNs, GNNs, or transformers. Both OTO-v2 and DepGraph are restricted to Pytorch models and use dependency graphs which capture limited information thus requiring a more manual understanding of some networks like ViT.

**Pruning time:** Numerous pruning methods utilize distinct pruning configurations, which exhibit variations in terms of the initial state of the model subjected to pruning (i.e., whether it is a fully trained model or randomly initialized) and the necessity of fine-tuning the pruned model. In this paper, we are mainly interested in the following frameworks: (1) train-prune-finetune Han et al. (2015) where a pre-trained model is finetuned after the pruning step, (2) prune-train Lee et al. (2019);

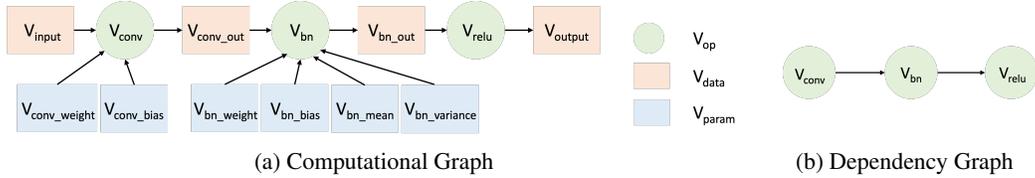


Figure 2: Comparison of Computational Graph and Dependency Graph. Fig. 2a is a computational graph. This graph is composed of three operators linked by the data nodes. Convolution and BatchNorm have parameters; they form the parameter nodes in the computational graph. Fig. 2b is the Dependency Graph of the same structure; only information on linked operators is stored.

Verdenius et al. (2020); Wang et al. (2020); Rachwan et al. (2022), where a randomly initialized model is pruned and then trained to convergence and (3) train-prune, where a pre-trained model is pruned without the need for further finetuning Lazarevich et al. (2021); Frantar et al. (2022); Srinivas & Babu (2015); Narshana et al. (2023). Some other interesting proposed frameworks are early pruning, where the model is slightly trained at the beginning, after which it is pruned and further fine-tuned Rachwan et al. (2022); You et al. (2020), pruning during training, where the pruning and training steps happen simultaneously Evci et al. (2020), as well as Chen et al. (2021, 2023), where pruned structures are learned during training.

### 3 Structurally Prune Anything

#### 3.1 Prune Any Framework

Our algorithmic analysis critically depends on the computational graph  $CG$ . For every neural network under consideration, the initial step involves constructing its computational graph. The computational graph is a directed graph that consists of three types of nodes: operator nodes  $v_{op}$ , which represent basic operators; normal data nodes  $v_{data}$ , which represent the input and output of operators, and parameter data nodes  $v_{param}$  which represent the operators’ parameter. Unlike the dependency graph, which only records the dependencies between operators, the computational graph provides essential insights into the relationships among operators and data connections that are necessary to detect dependencies between channels within any model architecture; it meticulously captures crucial information, including the sequencing of operators, the nature of operator-data connections, and the specific data shapes involved. See Fig. 2 for a comparison of the computational graph and dependency graph.

In our work, we establish a computational graph using the ONNX framework for pruning. The adoption of ONNX offers several notable advantages. First, ONNX provides a static trace of the model, facilitating the straightforward construction of a computational graph based on its explicit representation. Second, ONNX offers a standardized format for model representation. Regardless of how various layers are defined in different frameworks, once converted to ONNX, they assume a uniform sequence of fundamental ONNX operators. This standardization ensures that the analysis of the computational graph remains independent of the underlying frameworks, thus making it framework-agnostic. Third, ONNX enables seamless portability and cross-platform compatibility for models. Models can be effortlessly converted between different frameworks and ONNX. In our work, as depicted in Fig. 1, we initially convert models to the ONNX format. This step allows us to construct and examine the computational graph, as well as directly modify the ONNX model. Afterward, we have the option to convert the model back to its original framework.

#### 3.2 Prune Any Architecture

Given the neural network  $f_{\Theta}(x) = y$  where  $x$  is the input,  $y$  is the predicted output, and  $\Theta = \{\theta^{(1)}, \dots, \theta^{(L)}\}$  are the parameters with  $L$  layers, the goal of SPA is to automatically detect structural correlations within parameters  $\theta^{(1)}, \dots, \theta^{(L)}$ , and prune their less important channels or dimensions. To this end, SPA uses four steps:

**Step 1: Coupling channels via mask propagation.** Coupling channels are channels that are interconnected due to the dimensional constraints of subsequent layers (e.g. see same colored

channels in Figs. 1 and 5). Given the computational graph, we employ a mask propagation technique which intuitively aims at finding all the coupled channels for any target channel within any source node. To this end, it initially creates a mask for the target channel in the source node, and iteratively passes it through the operator nodes using predefined rules to identify correlated channels in other parameter nodes. These predefined rules are specific to the standard ONNX operators (see details in Appendix A.3). We explicitly describe the mask propagation algorithm in Alg. 1. First, it takes as input a computational graph  $CG$ , a source data node  $v_s$  and a mask  $m_{v_s}$  for the target channel that initializes propagation. Then, it iteratively visits neighboring operator nodes defined by  $neighbor(u, CG)$  (see 1.5 in Alg. 1), and propagates masks with the propagation rules defined by  $v_{op}.propagate(m_u, u)$  (see 1.7 in Alg. 1).

---

**Algorithm 1** Coupling channels via mask propagation

---

**Input:** computational graph  $CG$ , a source node  $s$ , a source mask  $m_{v_s}$  in which a target channel is masked.

**Output:** a dict  $M$  containing masks in which coupled channels are masked.

```

1:  $M = \{v_s : m_{v_s}\}; stack = (v_s, m_{v_s})$ 
2: # Visit all correlated data nodes
3: while  $stack$  do
4:    $u, m_u = stack.pop()$ 
5:   for  $op$  in  $neighbor(u, CG)$  do
6:     # Propagate  $m_u$  from  $u$  via  $v_{op}$ 
7:      $M_{neighbors} = v_{op}.propagate(m_u, u)$ 
8:     for  $v$  in  $M_{neighbors}$  not in  $M$  do
9:        $stack.push(v, M_{neighbors}[v])$ 
10:       $M.push(v, M_{neighbors}[v])$ 
11: return  $M$ 

```

---

**Step 2: Grouping coupled channels.** After utilizing the mask propagation method to effectively detect coupled channels in the previous step, we now propose to organize them into groups.

We use  $G = \{g_1, g_2, \dots\}$  to denote all groups. A specific group  $g_i$  contains a set of coupled channels  $CC$  which have the same pattern (e.g. as represented by the group of four colored sets of coupled channels in Fig. 5), hence  $g_i = \{CC_1, CC_2, \dots\}$ . Each coupled channel needs to be deleted as a whole. The individual channels into a given layer in coupled channels are denoted as  $C$ . Each parameter  $\theta$ , within a coupled channel, can be assigned an importance score using some score function  $S(\theta)$ .

The grouping algorithm is shown in Alg. 2. We are given a computational graph, and the algorithm returns all groups. The algorithm loops over all operators in the computational graph to detect coupled channels. To avoid redundant computation, only the output channels of the parameter nodes of each operator are analyzed since the input channels of the operator have been analyzed by its preceding operator.

**Step 3: Importance estimation.** After obtaining the groups, the next step is to assign to each set of coupled channels an importance score which is critical to effectively execute structured pruning. Indeed, this notion has been previously embraced by methodologies such as Group Fisher Liu et al. (2021) and DepGraph Fang et al. (2023), both of which validated its efficacy through empirical experiments. Our approach capitalizes on its inherent autonomous capability to recognize interconnected channels, thereby achieving a higher degree of generality and unity by providing support for a wide range of diverse aggregation and normalization of the individual weight scores. Hence, we propose the following scoring function for the coupled channels  $j$  in group  $i$ :

$$s_{i,j} = \underset{CC_l \in g_i}{Norm}(\{AGG(S(\theta_k), \forall \theta_k \in CC_j)\}) \quad (1)$$

The operator  $AGG$  aggregates all importance scores  $S(\theta_k)$  within the set of coupled channels  $CC_j$  into a singular score which is then normalized over the other coupled channels of the same group via the operator  $Norm$  to keep the scores of coupled channels from different groups within the same range for a fair assessment of relative importance. This scoring function is flexible and can encompass different weight scores  $S$  (e.g. L1 norm, first-order or second-order, and OBS Hassibi & Stork (1992) importance scores), different aggregation operators  $AGG$  (e.g. mean, max, and product), or different

---

**Algorithm 2** Grouping coupled channels

---

**Input** computational graph  $CG$ , set  $OPS$  with non analyzed operators  
**Output** Groups:  $G$

- 1:  $G \leftarrow \emptyset$
- 2: **while**  $OPS$  not empty **do**
- 3:      $v_{op} = OPS.pop()$
- 4:      $g = \emptyset; u = parameter\_node(v_{op})$
- 5:     # Add all coupled channel  $CC$  for group  $g$
- 6:     **for**  $C$  in  $u$ 's output channels **do**
- 7:          $m_u = create\_mask(u, C)$
- 8:          $CC = coupled\_ch(CG, u, m_u)$  ▷ Alg. 1
- 9:          $g.add(CC)$
- 10:      $G.insert(g)$
- 11:     # Mark visited all analyzed operators in group  $g$
- 12:     **for**  $v_{op}$  in  $analyzed\_ops(CG, g)$  **do**
- 13:          $OPS.remove(op)$
- 14: **return**  $G$

---

normalization scores  $Norm$  (e.g. summation, maximum, or median). The best choice of  $AGG$  and  $Norm$  function is not fixed over different models; it can be regarded as hyper-parameters that need to be tuned before pruning. We present the detailed algorithm in the Appendix as Alg. 3.

**Step 4: Pruning.** After obtaining the importance score for each set of coupled channels, we simply sort them to identify the least important ones. Subsequently, we locate these channels in the ONNX model, before finally removing them by adjusting the shape and data in the corresponding parameter nodes.

**Time complexity:** Within a single group  $g_i$ , we assume that there are  $|E_i|$  edges in this sub computational graph and  $m_i$  set of coupled channels. The analysis of a single channel takes  $\mathcal{O}(|E_i|)$  since the application of the predefined rules takes  $\mathcal{O}(1)$  and in the worst case we need to analyze every link between data nodes and operators. If we loop over all channels within one group as suggested in Alg. 2, it takes  $\mathcal{O}(|E_i| \cdot m_i)$ . However, a single mask propagation analysis per group is sufficient because all coupled channels within a group adhere to the same pattern. This reduces the complexity of analyzing one group to  $\mathcal{O}(|E_i|)$ . For the whole neural network, the analysis in each group is non-overlapping, so the overall complexity of grouping a neural network will still be  $\mathcal{O}(|E|)$  where  $|E| = \sum |E_i|$  is the number of edges of the network. The pruning procedure is simply a loop over all operators which takes  $\mathcal{O}(|V_{param}|)$  where  $|V_{param}|$  is the total operator number. The overall complexity of our pruning procedure is  $\mathcal{O}(|E| + |V_{param}|)$ .

### 3.3 Prune Any Time

In the previous sections, we developed the general SPA framework to automatically detect coupled channels and assign them an importance score. Leveraging the grouping analysis capabilities of SPA, we can incorporate many importance estimation criteria (denoted by  $S(\cdot)$  in Eq. (1)) into our framework. These pruning criteria are usually designed to be used at different training stages like in the train-prune-finetune, the prune-train, and the train-prune settings. Beyond enabling the application of pruning at different training stages, the SPA framework allows to transfer existing pruning criteria into a group-level structured version.

**Train-Prune-Finetune.** We support criteria that follow the train-prune-finetune scheme. The Magnitude-based criterion is the simplest method to determine a parameter's importance after training. By aggregating the L1-norm following Eq. (1), we have SPA-L1, a group-structured pruning criterion after training. Although the ONNX model is perfect for performing forward passes and building a standardized computational graph, it is not suitable for backward pass for the subsequent finetuning. In order to support the train-prune-finetune scheme, more specifically the finetuning phase, we need to convert the pruned ONNX model to any framework that supports gradient calculation, in our case, we choose PyTorch.

**Prune-Train.** We also support the prune-train scheme by applying the same group extension to before-training criteria; for example, we implement SPA-SNIP, SPA-Crop and SPA-GraSP which serve as group-based extensions of the three pre-training pruning criteria, SNIP Lee et al. (2019), CroP Rachwan et al. (2022) and GraSP Wang et al. (2020), respectively. Those three methods require the calculation of first or second-order derivatives of the parameters which is not natively supported by ONNX. To support gradient-based importance scores, SPA proposes to convert back the ONNX model into a framework supporting gradient computation like Pytorch. Thus, while SPA conveniently benefits from the computational graph from ONNX to achieve its framework and architecture agnostic properties (see Secs. 3.1 and 3.2), it also benefits from the practical gradient computations capacities from Pytorch. It is worth mentioning that the conversion between PyTorch and ONNX produces very limited computation overhead, which takes only seconds (see Tab. 6).

**Train-Prune.** For the more challenging pruning without fine-tuning setting, we propose a new algorithm, Optimal Brain SPA (OBSPA). We leverage the layer-wise sparsification operated by the unstructured pruning methods OBC Frantar et al. (2022) and its scalable version Frantar & Alistarh (2023), to create a novel structured pruning method which can be integrated into our SPA framework. In the original OBC method, the goal is to find a mask  $M$  and an updated weight matrix  $\hat{\Theta}$  that best preserves the output of each layer given some calibration data  $X$  and the original weight matrix  $\Theta$ , i.e:

$$\operatorname{argmin}_{M, \hat{\Theta}} \|\Theta X - (M \odot \hat{\Theta}) X\|_2^2 \quad (2)$$

Based on Frantar & Alistarh (2023), the mask  $M$  is determined according to the layer-OBS score (see Eq. (12)), and the weight matrix is updated based on the inverse Hessian  $H^{-1} = (X X^T + \lambda I)^{-1}$ .

Different from OBC, which uses masks with scattered zeros to facilitate unstructured pruning, OBSPA applies group-level importance estimation to obtain masks that have zeros of entire channels. While the weight updating procedure in OBSPA is similar to Frantar & Alistarh (2023), a crucial difference is that we need to structurally score each coupled channel as a whole with Eq. (1) to properly delete them without breaking the computational graph (see Fig. 7). Hence, in contrast with OBC, OBSPA can deliver real-world efficiency gains on GPU hardware.

Finally, a notable advancement of OBSPA compared to OBC pertains to the selection of calibration data employed for Hessian computation. Frantar & Alistarh (2023) use In Distribution (ID) data directly sampled from the training set. However, since the calibration data is only used to preserve the functionality of each layer, we made some relaxations on the previous setting to make it a data-free approach. In a more lenient data-free context, we lack access to the original training data but can employ data from Out-of-Distribution (OOD) sources. The most rigorous data-free scenario entails a lack of access to both ID and OOD data. Calibration samples are drawn from a uniform distribution in this "DataFree" setting. We evaluate both data-driven and data-free approaches in the experiment. Additionally, we propose a batch norm calibration method to improve the performance under ID and OOD settings (see Appendix B.3 for details).

## 4 Experiments

In this section, we show that SPA can prune any framework (see Sec. 4.1), any architecture (see Sec. 4.2), any time (see Sec. 4.3).

**Dataset.** This work mainly focuses on image classification tasks. We conduct extensive experiments with various datasets including CIFAR-10 (Krizhevsky et al., a), CIFAR-100 (Krizhevsky et al., b), ImageNette Howard and ImageNet-1k Deng et al. (2009). We also conduct experiments on text tasks and conduct experiments with SST-2 Socher et al. (2013) dataset, which is a sentiment classification task in NLP.

**Evaluation metrics.** The metric employed to evaluate the extent of performance preservation after pruning is classification accuracy. Similarly to Fang et al. (2023); Narshana et al. (2023), our evaluation of efficiency encompasses two primary measures: reduction in floating point operations (FLOPs), denoted as  $RF$ , and reduction in parameters, denoted as  $RP$ . It is important to emphasize that the  $RF$  metric carries greater significance, as it accurately reflects the actual reduction in

computational workload. We employ the fraction of reduced FLOPs and the fraction of reduced parameters, which range from 0 to 1, to facilitate the visualization of these metrics in the figures.

#### 4.1 Prune Any Framework

To validate that SPA is framework-agnostic, we investigated the pruning of ResNet-18 models derived from PyTorch, TensorFlow, MXNet, and Jax, using the ImageNette dataset as a benchmark for performance evaluation. Models were first initialized and trained within their respective frameworks, after which they were converted to the ONNX format, a reduction of approximately  $2\times$  in FLOPs utilization is targeted after pruning. In addition to the pruning outcomes, we also test the computational overhead incurred during the framework conversion process, all conversions can be completed within seconds (see Tab. 6 in Appendix).

*Observations:* In Tab. 1, the outcomes of pruning ResNet-18 models from diverse source frameworks are presented. We show that we successfully prune models from all four frameworks, this validates the framework-agnostic prowess of SPA. The experiment underscores that a model can be successfully converted to the ONNX format in seconds, and then pruned using SPA framework.

Table 1: Structure pruning with SPA from 4 important Deep Learning frameworks with ResNet-18 on ImageNette

Framework	ori acc.	pruned acc.	RF	RP
PyTorch	83.11%	82.96%	2.16 $\times$	2.05 $\times$
TensorFlow	82.62%	84.30%	1.94 $\times$	5.25 $\times$
MXNet	84.36%	82.77%	1.83 $\times$	8.03 $\times$
Jax	84.46%	83.33%	2.26 $\times$	3.64 $\times$

#### 4.2 Prune Any Architecture

To showcase SPA’s pruning ability across various architectures, we conducted pruning experiments on a range of 11 architectures including AlexNet, DenseNet-121, EfficientNet-b0, MobileNet-v2, RegNet\_x\_16gf, ResNet-50, Resnext-50\_32x4d, VGG-16, and Wide-ResNet-101\_2, ViT-base-patch16 on image classification task and DistilBERT on sentiment classification task. These architectures demonstrate a variety of building blocks including skip connections, MLP, convolutions, group convolutions, attention mechanisms, batch normalization, and more. The pruning process was executed within the context of the train-prune-finetune setting with the L1-based criterion being used as the designated importance score. In this experiment, we target a reduction of  $\sim 2\times$  in FLOPs for all models.

*Observations:* The outcomes, as presented in Tab. 2, underscore the power of SPA in supporting a wide range of neural network architectures containing all of the aforementioned building blocks. Even with the simple L1-based criterion, the pruned models achieve very competitive performance compared to their dense counterparts.

#### 4.3 Prune Any Time

**Prune with fine-tuning.** By harnessing the channel grouping capability of SPA, we unlock the potential for extending a multitude of established criteria to a structured group-level pruning paradigm. We aim to underscore the efficacy of our grouped importance estimation method under the pruning with fine-tuning setting on criteria applied both before and after training. We compare the performance of the group L1-based criterion, a train-prune-finetune criterion, to the ungrouped L1 criterion. Then, we delve into the prune-train criteria, where we compare the extended grouping of three prevalent unstructured approaches – SNIP, CroP, and GraSP – alongside their structured counterparts, SNAP, structured-CroP, and structured-GraSP. Finally, we also evaluate the OBSPA with additional fine-tuning. The postfix "it" denote that pruning is applied in an iterative manner. The evaluation is performed on ResNet-18/CIFAR-10, VGG-16/CIFAR-100, DenseNet-121/ImageNet, ResNet-50/ImageNet and Vit\_b\_16/ImageNet.

*Observations:* Through Figs. 3 and 9, we first conclusively showcase SPA’s versatility in accommodating diverse methodologies. For unstructured criteria like L1-based, SNIP, CroP, and GraSP, the

Table 2: Structured pruning with SPA on 11 architectures on CIFAR10 for image classification models and SST-2 for DistilBERT.

Model	ori acc.	pruned acc.	RF	RP
AlexNet	89.99%	89.80%	1.98×	1.46×
DenseNet-121	93.30%	94.20%	2.14×	2.35×
EfficientNet-b0	94.15%	92..06%	2.14×	1.86×
MobileNet-v2	92.33%	92.54%	2.33×	2.07×
RegNet_x_16gf	93.83%	93.75%	2.13×	1.83×
ResNet-50	93.26%	93.42%	2.13×	1.98×
ResNext-50_32x4d	93.95%	93.99%	2.07×	2.05×
VGG16	93.82%	94.06%	2.05×	2.45×
WideResNet-101	93.50%	93.41%	2.00×	1.88×
ViT-base	95.35%	96.10%	2.05×	1.50×
DistilBERT	91.06%	88.88%	2.04×	1.50×

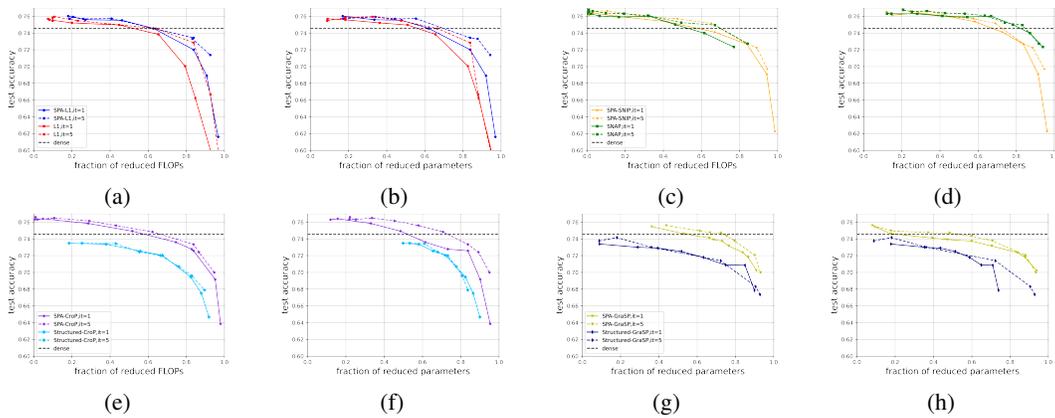


Figure 3: Trade off between accuracy and FLOPs/parameters with VGG-16 on CIFAR-100 (Figs. 3a to 3h). SPA efficiently implements both the structured and grouped versions of train-prune-finetune criteria like L1 and prune-train criteria like SNAP, CroP and GraSP

Table 3: Structured pruning of ResNet-50 on ImageNet with fine-tuning. "N/R" indicate non-reported results in original papers.

method	top1 acc.	top5 acc.	RF	RP
Base Model	76.15%	92.86%	1×	1×
DFPC Narshana et al. (2023)	75.83%	N/R	1.98×	1.84×
OTO-v2 Chen et al. (2023)	75.2%	92.2%	2.68×	2.02×
DepGraph Fang et al. (2023)	75.83%	N/R	2.07×	N/R
SPA-L1	74.83%	92.57%	2.84×	2.60×
SPA-L1	76.39%	93.29%	2.18×	1.85×
OBSPA	76.59%	93.40%	2.22×	1.90×

Table 4: Structured pruning of ResNet-50 and VGG-19 on CIFAR-10 and CIFAR-100 without finetuning

method	CIFAR-10						CIFAR-100					
	ResNet-50			VGG-19			ResNet-50			VGG-19		
	acc. drop	RF	RP									
DFPC	-4.74%	1.46	2.07	-3.38%	1.68	3.16	-8.53%	1.27	1.22	-1.92%	1.26	1.50
OBSPA (ID)	-0.95%	1.48	1.51	-0.99%	1.71	1.44	-3.73%	1.46	1.32	-0.80%	1.54	1.28
OBSPA (OOD)	-1.13%	1.48	1.52	-1.67%	1.73	1.35	-3.70%	1.47	1.34	-1.13%	1.54	1.28
OBSPA (DataFree)	-1.34%	1.48	1.51	-1.64%	1.80	1.35	-5.24%	1.37	1.23	-1.59%	1.47	1.28

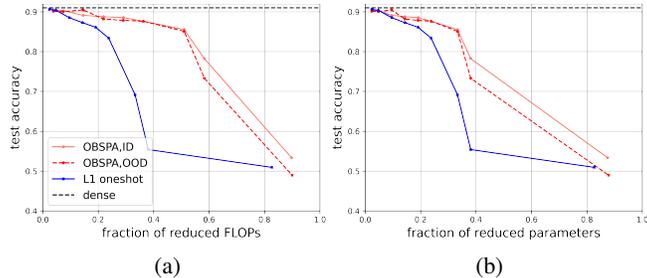


Figure 4: Trade off between accuracy and FLOPs/parameters with DistilBERT on SST-2 sentiment classification task.

extension to group-structured pruning is easily achieved through the SPA group analysis. Moreover, we interestingly observe that the performance of the SPA grouped pruning criteria either matches or outperforms their original structured counterparts. We intuitively explain this observation by the fact that, in contrast with the original structured version of the algorithms, the SPA grouped versions accounts for all information in a set of coupled channels by aggregating the importance scores over *all* its weights. We also observe that gradual iterative pruning consistently yields superior outcomes compared to one-shot channel pruning across nearly all methods. Finally, SPA matches or outperforms the performance of previous dependency graph approaches on ImageNet in Tabs. 3, 7 and 8.

**Prune without fine-tuning.** In this section, our focal point is to showcase the state-of-the-art performance achieved by OBSPA in the challenging train-prune setting. Following the precedent established by DFPC, we assess the classification performance of pre-trained ResNet-50, ResNet-101, and VGG-19 models on both CIFAR-10 and CIFAR-100 datasets. We also test OBSPA’s performance on NLP tasks, and compare OBSPA with L1-based one-shot pruning on pruning a DistilBERT that conducts sentiment classification on SST-2 dataset. Additionally, experiments involving ResNet-50 on the ImageNet dataset have been included in the Appendix (see Appendix C.3) to further substantiate our findings. We conducted experiments in both data-driven and data-free settings. In the experiments, CIFAR-10 serves as OOD dataset for CIFAR-100, and CIFAR-100 serves as OOD dataset for CIFAR-10. We use ax Wang et al. (2019), another text dataset that contains Natural Language Inference (NLI) problems as OOD dataset for SST-2.

*Observations:* We establish a comprehensive comparison between our algorithm and the data-free pruning approach DFPC. Tab. 4 shows the result of pruning a ResNet-50 and a VGG-19. The outcomes demonstrate the superiority of OBSPA over DFPC. Specifically, when achieving identical levels of FLOPs reduction, our data-free technique exhibits a mere 1.34% accuracy drop on the CIFAR-10 classification task with ResNet-50, a remarkable contrast to DFPC’s 4.74% drop. This substantial-performance disparity is also noteworthy on the more complex CIFAR-100 dataset. Notably, for the CIFAR-100 classification with ResNet-50, our data-free approach showcases a 10% greater FLOPs reduction coupled with a 3.29% less reduction in accuracy deterioration compared to DFPC. This promising trend is consistently replicated across the ResNet-101 and VGG-19, the ResNet-101 experiment is listed in Appendix. Furthermore, we compare OBSPA with a basic L1-based one-shot pruning criterion with DistilBERT on SST-2, as suggested in Fig. 4, OBSPA achieves a much better performance/efficiency trade-off. Finally, OBSPA is also much faster than DFPC. We compare the pruning time of our OBSPA algorithm to DFPC, see results in Appendix Tab. 13. We achieved an impressive 6× speedup for pruning ResNet-50 on both CIFAR and ImageNet-1k dataset.

## 5 Conclusion

In this work, we introduce SPA, a novel pruning framework that not only automates the pruning of neural networks across diverse architectures but also accommodates models originating from various frameworks. By capitalizing on its inherent capability to aggregate interdependent channels, SPA can convert many pruning criteria into structured pruning algorithms at the group level making it applicable at any time in the training process. Finally, we propose OBSPA, a structured pruning without fine-tuning algorithm which achieves state-of-the-art performance.

## Broader Impact

This paper presents work that aims to advance the field of efficient Machine Learning (ML). Beyond increasing the speed of ML models, a primary goal of efficiency gains is to reduce the energy and emissions impact of ML applications which is an urgent environmental challenge Dhar (2020). Despite the cost reduction that ML compression methods can offer, we encourage practitioners to be aware of the risk of rebound effect and make non-energy policy a standard practice Dhar (2020).

## References

- Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Gutttag, J. V. What is the state of neural network pruning? *ArXiv*, abs/2003.03033, 2020.
- Chen, T., Ji, B., Tianyu, D., Fang, B., Wang, G., Zhu, Z., Liang, L., Shi, Y., Yi, S., and Tu, X. Only train once: A one-shot neural network training and pruning framework. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.
- Chen, T., Liang, L., Tianyu, D., Zhu, Z., and Zharkov, I. Otov2: Automatic, generic, user-friendly. In *International Conference on Learning Representations*, 2023.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, June 2019.
- Dhar, P. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2:423 – 425, 2020. URL <https://api.semanticscholar.org/CorpusID:225488526>.
- Ding, X., Hao, T., Tan, J., Liu, J., Han, J., Guo, Y., and Ding, G. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4510–4520, 2021.
- Dong, X., Chen, S., and Pan, S. J. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NIPS*, 2017.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=YicbFdNTTy>.
- Evcı, U., Elsen, E., Castro, P., and Gale, T. Rigging the lottery: Making all tickets winners, 2020. URL <https://openreview.net/forum?id=ryg7vA4tPB>.
- Fang, G., Ma, X., Song, M., Mi, M. B., and Wang, X. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16091–16101, 2023.
- Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. *ArXiv*, abs/2301.00774, 2023.
- Frantar, E., Singh, S. P., and Alistarh, D. Optimal Brain Compression: a framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 36, 2022.
- Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

- Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1992.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.
- He, Y. and Xiao, L. Structured pruning for deep convolutional neural networks: A survey. *ArXiv*, abs/2303.00566, 2023.
- He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1398–1406, 2017.
- He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In *International Joint Conference on Artificial Intelligence*, 2018a.
- He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4335–4344, 2018b.
- Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., and Song, D. Natural adversarial examples. *CVPR*, 2021.
- Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.
- Howard, J. Imagewang. URL <https://github.com/fastai/imagenette/>.
- Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2016.
- Huang, Z. and Wang, N. Data-driven sparse structure selection for deep neural networks. *ArXiv*, abs/1707.01213, 2017. URL <https://api.semanticscholar.org/CorpusID:575794>.
- Jian-Hao Luo, J. W. and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pp. 5058–5066, 2017.
- Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). a. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- Krizhevsky, A., Nair, V., and Hinton, G. Cifar-100 (canadian institute for advanced research). b. URL <http://www.cs.toronto.edu/~kriz/cifar.html>.
- Lazarevich, I., Kozlov, A., and Malinin, N. Post-training deep neural network pruning via layer-wise calibration. *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pp. 798–805, 2021.
- Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Madry, A. FFCV: Accelerating training by removing data bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*, 2023. [https://github.com/libffcv/ffcv/.commit xxxxxxx](https://github.com/libffcv/ffcv/.commit/xxxxxxx).
- LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NIPS*, 1989.
- Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.
- Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *ArXiv*, abs/1608.08710, 2016.
- Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. Hrank: Filter pruning using high-rank feature map. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1526–1535, 2020.
- Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J., Wang, X., Chen, Y., Yang, W., Liao, Q., and Zhang, W. Group fisher pruning for practical network compression. In *International Conference on Machine Learning*, 2021.

- Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2755–2763, 2017a.
- Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017b.
- Lubana, E. S. and Dick, R. P. A gradient flow framework for analyzing network pruning. In *International Conference on Learning Representations*, 2021. URL <https://openreview.net/forum?id=rumv7QmLUue>.
- Narshana, T., Murti, C., and Bhattacharyya, C. DFPC: Data flow driven pruning of coupled channels without data. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=mhnHqRqcjYU>.
- Rachwan, J., Zügner, D., Charpentier, B., Geisler, S., Ayle, M., and Günnemann, S. Winning the lottery ahead of time: Efficient early network pruning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*. PMLR, 2022.
- Radosavovic, I., Kosaraju, R. P., Girshick, R. B., He, K., and Dollár, P. Designing network design spaces. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10425–10433, 2020.
- Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C., Ng, A., and Potts, C. Parsing With Compositional Vector Grammars. In *EMNLP*. 2013.
- Srinivas, S. and Babu, R. V. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference*, 2015.
- Tan, M. and Le, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 09–15 Jun 2019. URL <https://proceedings.mlr.press/v97/tan19a.html>.
- Verdenius, S., Stol, M., and Forré, P. Pruning via Iterative Ranking of Sensitivity Statistics. *arXiv e-prints*, art. arXiv:2006.00896, June 2020.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.
- Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=SkgsACVKPH>.
- Xiao, X., Wang, Z., and Rajasekaran, S. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Neural Information Processing Systems*, 2019.
- Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2016.
- You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Toward more efficient training of deep networks. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=BJxsrgStvr>.
- You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

- Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2017.
- Zagoruyko, S. and Komodakis, N. Wide residual networks. In *BMVC*, 2016.
- Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., and Zhu, J.-H. Discrimination-aware channel pruning for deep neural networks. In *Neural Information Processing Systems*, 2018. URL <https://api.semanticscholar.org/CorpusID:53102564>.

## A Method Details

In this section, we provide a more detailed explanation of our method as well as implementation details.

### A.1 SPA group visualization

We provide in Fig. 5 an example of a group of a residual structure with four sets of coupled channels. Within this group, each color represents a coupled channel that must be pruned altogether.

### A.2 Building Computational Graph

Starting with an ONNX model, we apply `onnx-graphsurgeon`, a tool developed in the NVIDIA’s TensorRT toolkit<sup>2</sup>. This library enables the effortless generation and modification of ONNX models, allowing us to transform the model into a `graphsurgeon` graph, which we referred to as "*gs\_graph*." This *gs\_graph* serves as a straightforward intermediate representation characterized by interconnected Nodes, each functioning as an operator. Every Node maintains its own set of inputs and outputs. To enhance subsequent analysis, we construct our Computational Graph using *gs\_graph* which is used in Sec. 3 as a foundation for SPA. Instead of relying solely on operator Nodes, we introduce separate nodes for operators, parameters, and intermediate data. This approach allows us to define propagation methods on the nodes we generate.

### A.3 Coupling channels via mask propagation

Our approach hinges on the development of mask propagation rules tailored to individual core ONNX operators, the rules provide information on how channels are correlated within a single ONNX operator. Once these propagation rules are established for all operators within a network structure, we gain the capability to comprehensively analyze this network. By formulating rules for the majority of foundational operators, our methodology can effectively analyze a broad spectrum of neural network architectures. Furthermore, in the event that new operators are introduced, we can seamlessly extend our analysis by defining specific rules for these novel operators. This adaptability ensures our method remains versatile and up-to-date in addressing evolving neural network structures.

Our implementation supports more than 150 different operators, which are building blocks of deep learning architectures. As an example, we take the important example of the propagation through one and two General Matrix Multiplication (GeMM) operators defined by ONNX. First, we show a simplified definition of GeMM.

**Function:**

- compute  $Y = X * W + B$

**Inputs:**

- $X$ : input tensor with shape  $(M, K)$
- $W$ : input tensor with shape  $(K, N)$
- $B$ : optional input tensor, if not specified, the computation is done as if  $B$  is a scalar 0. The shape of  $B$  should be unidirectional broadcastable to  $(M, N)$ .

**Outputs:**

- $Y$ : output tensor with shape  $(M, N)$

**Propagation through one GeMM operator:** We establish the propagation rule for the GeMM operator when every possible dimension (i.e. first dimension denoted by 0 or second dimension denoted by 1) of every possible involved variable (i.e.  $X, W, B, Y$ ) is masked. Given the input mask of a single data node among all data nodes linked to the operator, the analysis procedure yields masks for the remaining data nodes. Detailed guidelines governing the analysis of GeMM are documented in Tab. 5. To illustrate, considering the first column Tab. 5, it implies that the removal of the first

---

<sup>2</sup><https://github.com/NVIDIA/TensorRT>

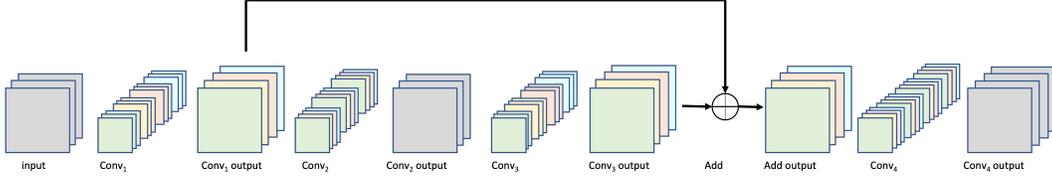


Figure 5: Showcase of a group of a residual structure. Four convolutions with a residual skip form this residual structure. All colored blocks form a group. Within this group, each color represents a coupled channel that must be pruned altogether.

dimension in input  $X$  necessitates the simultaneous removal of the first dimension in both  $B$  and output  $Y$ .

Table 5: Analysis rule of GeMM operator. Given an input mask covering dimension 0 or 1 of any variable  $X$ ,  $W$ ,  $B$ ,  $Y$ , the analysis rule defines the dimensions which should be covered in the output masks for the other variables.

Input mask	X:0	X:1	W:0	W:1	B:0	B:1	Y:0	Y:1
Output mask	B:0,Y:0	W:0	X:1	B:1, Y:1	X:0,Y:0	W:1, Y:1	X:0,W:0	W:1,B:1

**Propagation through two GeMM operators:** With the analysis rule of GeMM, we then provide an illustrative depiction of our analysis applied to two connected GeMM operators in Fig. 6. The computational graph depicts the linkage of two interconnected GeMM operators, each containing a pair of input nodes (one serving as the operator input, and the other as the weight matrix) as well as an output data node. To simplify the illustration, we consider the GeMM operator without a bias term. For input and output data nodes, each column corresponds to a distinct sample, while the row count corresponds to the number of features. For the weight nodes the column number indicates the input feature number, and the row number indicates the output feature number. As an example,  $X_1$  serves as the input for  $\text{GeMM}_1$ , encompassing 3 samples, each possessing 4 features. The output of  $\text{GeMM}_1$  comprises 4 features, hence the weights of  $\text{GeMM}_1$  form a  $4 \times 4$  matrix, and the resulting output,  $X_2$ , assumes a shape of  $4 \times 3$ .

The mask propagation analysis starts by applying a mask to one target channel of the source node. In Fig. 6, we aim to eliminate the first output channel of  $W_1$ . The algorithm first finds the corresponding operators of this data node. In this case,  $\text{GeMM}_1$  is the only operator that needs to be analyzed since  $W_1$  belongs to it and there are no other operators that generate  $W_1$ . By applying predefined rules defined in Tab. 5, we are given a new mask of  $X_2$ , indicating the necessity of deleting the first feature of  $X_2$ , as well as the fact that  $X_1$  is not affected. Then we apply the same methods on the new mask of  $X_2$ , it will first find both  $\text{GeMM}_1$  and  $\text{GeMM}_2$  as affected operators, but we will skip  $\text{GeMM}_1$  since it is already analyzed. Through this analysis step, we are returned the new mask of  $W_2$ , which indicates that we also need to delete the first input channels of  $W_2$ . We are also informed that  $X_3$ , the output of  $\text{GeMM}_2$  will not be affected. The analysis will end here because the mask of  $W_2$  will incur no analysis on new operators. In this way, we get the coupled channels of the initial target channel in the form of masks.

#### A.4 Importance Estimation

Alg. 3 is used to assign each coupled channel an importance score. The assessment of importance scores for individual parameters is first undertaken through designated criteria. Subsequently, the aggregation of these scores within each prunable dimension yields a consolidated measure. In pursuit of a global pruning strategy, scores are normalized within each group, thereby ensuring uniformity across all groups.

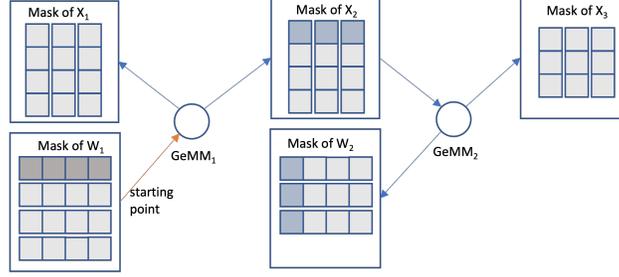


Figure 6: Example of operator-level analysis of a two connected GeMM. The analysis starts by masking the first output channels of  $W_1$ , through a series of mask propagation, the first feature dimension of  $x_2$  and the first input channel of  $W_2$  are also masked. The propagation order is illustrated through arrows.

---

### Algorithm 3 Group-level importance estimation

---

**Input** Groups:  $G$ , importance estimation criterion

**Output** score for each coupled channel

- 1: assign each parameter a score with the salience estimation criterion
  - 2:  $scores = \emptyset$  ▷ initialize score
  - 3: **for**  $g$  in  $G$  **do**
  - 4:    $scores_g = \emptyset$  ▷ initialize score of the group
  - 5:   **for**  $CC$  in  $g$  **do**
  - 6:      $score_{CC} = AGG(S(\theta_k))$  for all  $\theta_k$  in  $CC$
  - 7:      $scores_g.insert(score_{CC})$
  - 8:    $scores.insert(Norm(scores_g))$
  - 9: **return**  $scores$
- 

## A.5 Pruning Criteria

Pruning requires selectively removing redundant parameters (or connections) in the neural network. In order to do so, one has to come up with a good criterion to identify such redundant connections. In this section, we introduce some popular criteria that are applied to our method.

We first introduce important notations. Assume we have a neural network  $F : y = f_{\Theta}(x)$  with parameter  $\Theta$ , that maps the input data  $x \in \mathbb{R}^m$  to the output  $y \in \mathbb{R}^n$ ,  $\Theta$  denotes the parameters of the neural networks, a specific parameter is denoted as  $\theta$ . The neural networks have multiple layers, we use  $L$  to denote the total layer number and  $l$  to denote a specific layer. The parameters are optimized based on the loss function  $\mathcal{L}$ . We use  $g$  and  $H$  to denote the first-order derivative and second-order derivative (Hessian) of the loss with respect to the parameters, For a specific parameter,  $g(\theta) = \frac{\partial \mathcal{L}}{\partial \theta}$ ,  $H(\theta) = \frac{\partial^2 \mathcal{L}}{\partial \theta^2}$ , the importance score is  $S(\theta)$ .

**Magnitude-based criterion** directly uses the magnitude of each parameter as its importance score, parameters below a certain threshold are regarded as redundant. It can be simply defined as Eq. (3).

$$S(\theta) = |\theta| \quad (3)$$

**SNIP** Lee et al. (2019) is a sensitivity-based unstructured pruning criterion to be applied before training. To calculate the sensitivity of each parameter, an auxiliary gate variable  $c$  over the model's parameter is defined. They then initialize all  $c = 1$  and do not update them anymore. the criterion is defined as the derivative of the loss w.r.t. the gates according to Eq. (4).

$$S(\theta) = \frac{\partial \mathcal{L}(\theta \odot c)}{\partial c} = g(\theta) \odot \theta \quad (4)$$

**SNAP** Verdenius et al. (2020) proposed method to extend SNIP to structured pruning criterion by applying the auxiliary gates  $c = 1$  over each node's activation, which is denoted as  $h$ , the  $i$ th activation in layer  $l$  is denoted as  $h_i^{(l)}$  the importance score will be calculated with respect to the activation instead of a single parameter as defined in Eq. (5).

$$S(h_i^{(l)}) = \frac{\partial \mathcal{L}(h_i^{(l)} \odot c_i^{(l)})}{\partial c_i^{(l)}} \quad (5)$$

**GraSP** Wang et al. (2020) is based on the second-order derivative (Hessian) of the loss with w.r.t. the parameters. The goal of GraSP is to preserve or even increase the gradient flow. The Eq. (6) is used to measure the change of the gradient flow after pruning the parameter. If the score is positive, removing the corresponding parameter will reduce the gradient flow, and if the score is negative, removing the parameter will increase the gradient flow.

$$S(\Theta) = -\Theta^T H(\Theta)g(\Theta) \quad (6)$$

**CroP** Lubana & Dick (2021); Rachwan et al. (2022) also applies the second-order derivative to calculate the importance. The score of CroP is calculated as Eq. (7). The idea of this criterion is to preserve the gradient flow or training dynamics during training.

$$S(\Theta) = |\Theta^T H(\Theta)g(\Theta)| \quad (7)$$

**Structured-GraSP** (Eq. (8)) and **Structured-CroP** (Eq. (9)) apply a similar idea as SNAP to add auxiliary gate variables over activation to extend the unstructured criterion to a structured one.

$$S(h^{(l)}) = -H(c^{(l)})g(c^{(l)}) \quad (8)$$

$$S(h^{(l)}) = |H(c^{(l)})g(c^{(l)})| \quad (9)$$

**OBD** LeCun et al. (1989) and **OBS** Hassibi & Stork (1992) use the Hessian of the loss w.r.t. the parameters to calculate the importance score, the higher the value of Hessian, the higher the importance of the parameters. For the  $j$ th parameter  $\theta_j$ , see Eq. (10) for OBD score and Eq. (11) for OBS score. However, this approach requires the calculation of Hessian of all parameters of the neural networks, making it intractable to compute for large networks.

$$(OBD) \quad S(\theta_j) = \frac{\theta_j^2 H_{j,j}}{2} \quad (10)$$

$$(OBS) \quad S(\theta_j) = \frac{\theta_j^2}{2H_{j,j}^{-1}} \quad (11)$$

**OBC** Frantar et al. (2022) applies the method of OBS layer-wise to make the calculation tractable. Instead of minimizing the influence on the final loss in OBS, OBC minimizes the reconstruction error per layer, see Eq. (2) for problem definition, the goal is to find the optimal weight mask as well as an optimal update of the weight matrix to minimize the reconstruction error. OBC introduces a greedy solver that removes weights one-at-a-time, then fully reconstruct the remaining weights after each iteration via an efficient closed-form equations. The importance of the  $j$ th parameter of the  $l$ th layer is determined by their influence on the reconstruction error of the layer output as defined in Eq. (12). The hessian matrix of each layer is used here to calculate the importance and to update the parameters after pruning, they can be derived by taking the outer product of the calibration data per layer as  $H^{(l)} = X^{(l)} X^{(l)T}$ .

$$S(\theta_j^{(l)}) = \frac{(\theta_j^{(l)})^2}{[(H^{(l)})^{-1}]_{j,j}} \quad (12)$$

## A.6 OBSPA and SparseGPT

**SparseGPT** Frantar & Alistarh (2023) is a large-scale extension of OBC that proposes a method to incrementally prune weights in each column of the weight matrix. Different from OBS that uses the whole Hessian of the layer to adjust the values of all available parameters to compensate for the removal, Frantar & Alistarh (2023) only updates the weight among the remaining unpruned weights with a smaller Hessian matrix. The update procedure of SparseGPT is illustrated in Fig. 7a.

**OBSPA** is extended to a structured pruning algorithm from SparseGPT by applying group-level importance estimation and directly masking entire columns and rows before structurally deleting them. In OBSPA, we determine the coupled channels to be pruned by applying the layer-OBS Frantar et al. (2022) criterion and then create masks for those channels. We then apply the masks on the weight matrix column by column and update the remaining columns. For a specific column  $i$  that needs to be pruned, we first calculate the error and then update the remaining parameters with the following equations.

$$err = \frac{\Theta_{:,i}}{H_{i,i}^{-1}} \quad (13)$$

$$\Theta_{:,i} = \Theta_{:,i} - err \cdot H_{i,i}^{-1} \quad (14)$$

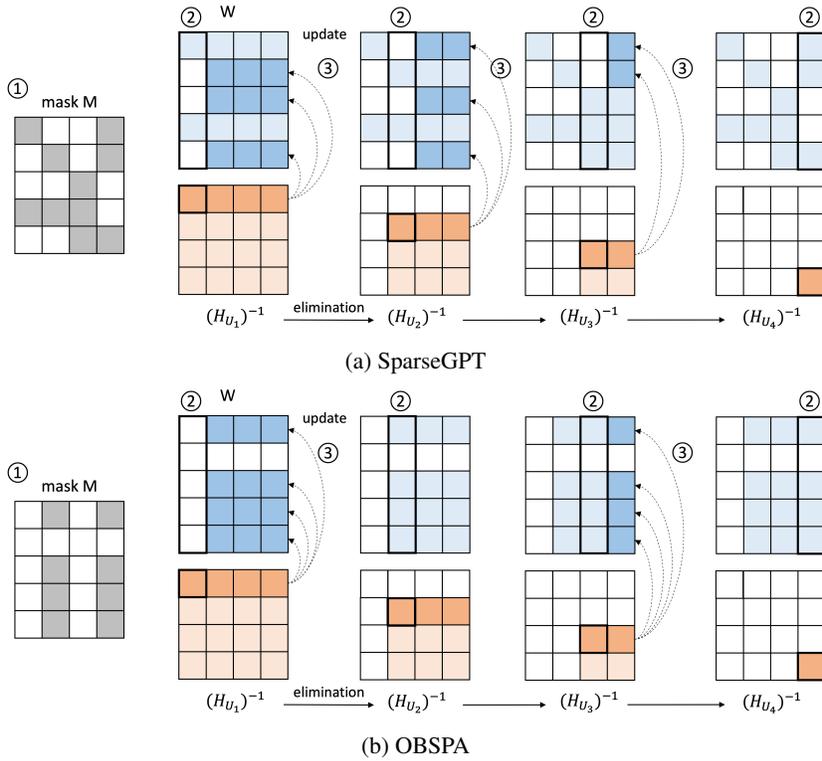


Figure 7: Visualization of reconstruction algorithm of Frantar & Alistarh (2023) and OBSPA. ① mask are derived according to layer-OBS score. for SparseGPT, zeros are scattered in the mask while for OBSPA, zeros span the whole channel. ② weights in the first column of the weight matrix are pruned. ③ Using Hessian inverses  $(H_{u_j})^{-1}$  to update the reminder of the weight (only in dark blue). Then repeat ②&③ for the next column until all columns are processed

## A.7 Implementation details

We provide Fig. 8 for a compact overview of the implementation of our method. As detailed in previous sections, we first obtain a  $gs\_graph$  and build our Computational Graph based on it. Then we apply mask propagation and importance estimation on the computational graph to derive the index of target channels for pruning. We can then very conveniently prune those channels on  $gs\_graph$  and

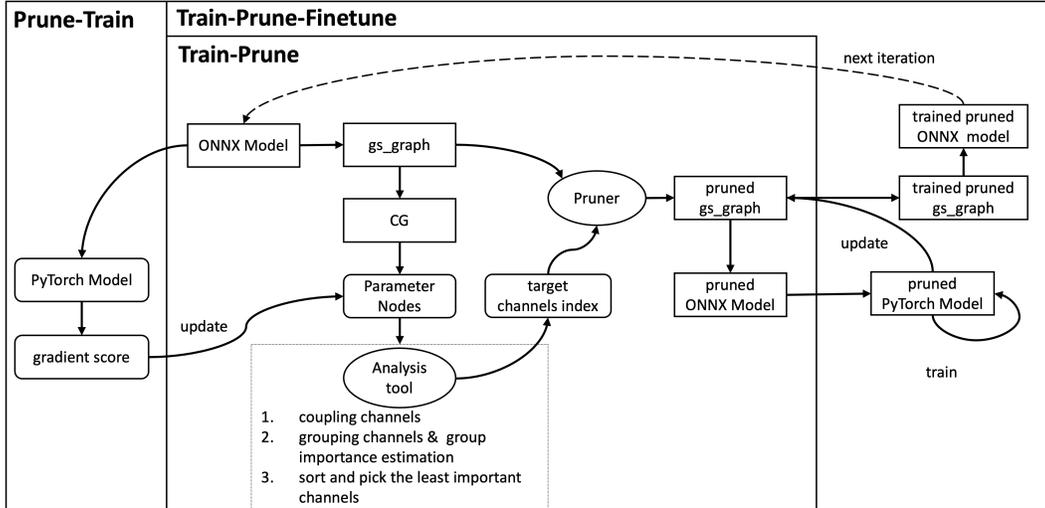


Figure 8: Detailed implementation of SPA

convert *gs\_graph* to ONNX model using tools provided by *onnx-graphsurgeon*. In this way, we can already support the Train-Prune framework. To further support Train-Prune-Finetune and Prune-Train settings, we add additional blocks to convert ONNX model to PyTorch model. This conversion grants our method the ability to apply sensitivity-based criteria and to train/fine-tune the pruned model.

## B Experiments Details

### B.1 Dataset Details

**CIFAR-10** Krizhevsky et al. (a) and **CIFAR-100** Krizhevsky et al. (b) datasets both serve as platforms for image classification tasks, diverging based on their class count and intricacy. CIFAR-10 comprises a collection of 60,000 32x32 color images, categorized into ten distinct classes, each containing 6,000 images. These classes encompass common objects like airplanes, automobiles, birds, cats, dogs, and more. In contrast, CIFAR-100, also consisting of 60,000 images, exhibits a finer granularity with 100 distinct classes, representing more nuanced categories. Notably, CIFAR-10 and CIFAR-100 are mutually exclusive, allowing for a reciprocal utilization wherein CIFAR-100 serves as an out-of-distribution dataset for CIFAR-10, and vice versa.

**ImageNet-1k** Deng et al. (2009), is a widely recognized and extensively used dataset in the field of computer vision and machine learning. This dataset consists of millions of labeled images, each categorized into one of the 1,000 predefined classes or object categories. The diversity and size of ImageNet make it a valuable resource for training and evaluating deep learning models. We also evaluate OBSPA’s performance on ImageNet-1k.

**ImageNet-O** Hendrycks et al. (2021) dataset is the natural adversarial example dataset for out-of-distribution detectors of ImageNet-1k. It consists of 2000 images from 200 classes that are not found in the ImageNet-1k dataset. We resize the images to 224x224. This dataset serves as an OOD dataset of ImageNet-1k in our experiment.

**Imagenette** Howard, derived from ImageNet, showcases 13394 images from a subset of 10 easily classifiable classes (e.g., tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute). We also preprocess the images to 224x224. Despite its modest size, Imagenette proves to be a suitable testbed to assess the functionality of SPA across models with divergent architectures.

**SST-2** Socher et al. (2013), the Stanford Sentiment Treebank 2 (SST-2) is a popular dataset for sentiment analysis in natural language processing. It consists of 215,154 unique phrases from movie reviews, where each review is labeled with its sentiment as either "positive" or "negative". The dataset is well-structured, and it has been widely used for training and evaluating sentiment analysis models.

In our work, we use pruning a DistilBERT model on this dataset to show SPA’s ability to prune self-attention-based NLP models.

## B.2 Metric Details

Reduction in Floating Points Operations and Reduction in Parameters are widely used in many papers Narshana et al. (2023); Fang et al. (2023) to demonstrate the effectiveness of pruning methods, we provide definition of these two evaluation metrics

1. Reduction in Floating Point Operations, represented as RF, quantifies the acceleration in FLOP execution speed achieved through pruning.

$$RF = \frac{FLOP_{before}}{FLOP_{after}} \quad (15)$$

2. Reduction in Parameters, denoted as RP, evaluates the parameter reduction achieved through the pruning process.

$$RP = \frac{\#params_{before}}{\#params_{after}} \quad (16)$$

## B.3 Setting Details

For the experiment that follows the Train-Prune-Finetune and Prune-Train schemes on CIFAR and ImageNet datasets, we use a 12GB NVIDIA GeForce GTX 1080 Ti GPU, for the experiments that follow the Train-Prune setting and the experiment on ImageNet, we use a 40G NVIDIA A100 GPU.

**Prune any framework:** We test the framework-agnostic ability of SPA on the ImageNet dataset. We first define random initialized ResNet-18 from PyTorch, TensorFlow, JAX, and MXNet respectively, they are then trained for 100 epochs on their original frameworks before being converted to ONNX. While PyTorch, TensorFlow, and MXNet offer direct conversion functionalities, Jax models necessitate an additional intermediary step, involving a conversion to TensorFlow before arriving at the ONNX representation. Then we prune and finetune the model based on SPA-L1. In addition to the pruning outcomes, we also test the computational overhead incurred during the framework conversion process. We quantify this overhead by reporting the average model conversion time, derived from 10 separate conversion instances as shown in Tab. 6.

**Prune any architecture:** The functional test of architecture-agnostic property of SPA is done on both CIFAR10 and SST-2. Here we conducted pruning experiments on DenseNet-121 Huang et al. (2016), EfficientNet-b0 Tan & Le (2019) MobileNet-v2 Howard et al. (2017), RegNet\_x\_16gf Radosavovic et al. (2020), ResNet-18 He et al. (2015), Resnext-50\_32x4d Xie et al. (2016), VGG-16 Simonyan & Zisserman (2015), Wide-ResNet-101\_2 Zagoruyko & Komodakis (2016) sourced from TorchVision, ViT Dosovitskiy et al. (2021) and DistilBERT Devlin et al. (2019) sourced from HuggingFace. The setting of those experiments are same as framework-agnostic experiments.

**Prune with fine-tuning:** This set of experiments is first done on ResNet-18 and VGG-16 to perform image classification on CIFAR-10 and CIFAR-100. We compare the L1-based method SPA-L1 to its ungrouped counterpart for the Train-Prune-Finetune setting and compare SPA-SNIP, SPA-CroP and SPA-GraSP to their structured algorithms, SNAP, Structured-CroP and Structured-GraSP for Prune-Train setting. To ensure equitable comparisons, we maintain uniformity in total epochs across all configurations. When pruning is executed after training, the model undergoes 100 epochs of training followed by 100 epochs of pruning and fine-tuning. Conversely, for pruning before training, a total of 200 epochs is allocated for the combined pruning and fine-tuning procedure. Besides, building upon the findings in Verdenius et al. (2020), which advocate for the efficacy of iterative pruning, we conduct iterative experiments for each criterion. In this iterative version, we employ 5 steps, with 5 training epochs between each step. The optimization procedure involves the use of the SGD optimizer and CosineAnnealingLR as the learning rate scheduler.

For the experiments on ImageNet-1k, we first pruned pre-trained ResNet-50, DenseNet-121, and ViT\_b\_16 using SPA-L1 and OBSPA, we then fine-tune the models following Fang et al. (2023)’s setting, with 90 epochs of fine-tuning on both pruned ResNet-50 and DenseNet-121. However different from Fang et al. (2023) that fine-tunes ViT for 300 epochs, we only fine-tune ViT for 30 epochs. We also follow Leclerc et al. (2023) to perform fast training.

**Prune without fine-tuning:** We follow the setting from DFPC to evaluate the performance of ResNet-50, ResNet-101, and VGG-19 models on both CIFAR-10 and CIFAR-100 datasets under the pruning without fine-tuning scheme. Models are pre-trained before pruning and no further fine-tuning is allowed after pruning. We also conduct experiments on the ImageNet-1k dataset. We use calibration data to calculate the Hessian per layer for importance estimation and parameter update. for the CIFAR dataset in which samples are in low resolution, 2048 data samples are used, for the ImageNet dataset, 896 data points are used. For image classification tasks, SPA-OBC encompasses two distinct settings: data-driven setting and data-free setting. Data points are directly sampled from the training set in the data-driven setting, but in the data-free setting, calibration data are either drawn from the OOD dataset or generated following a uniform distribution between 0 and 1. CIFAR-10 and CIFAR-100 are mutually exclusive, they can serve as OOD datasets for each other, we also use ImageNet-O as the OOD dataset of ImageNet-1k. However, In NLP tasks, where different sentences can be easily accessed, choosing random sentences is not rational. Consequently, we exclusively utilize out-of-distribution (OOD) datasets. Specifically, we employ the ax dataset as an example of an OOD dataset of SST-2.

We also need to mention an additional noteworthy observation pertaining to the performance enhancement achieved through the resetting of batch normalization statistics following pruning, a phenomenon previously elucidated in OBC Frantar et al. (2022). In our study, we adopt a straightforward approach of forwarding the calibration data twice to facilitate the updating of running mean and running variance in the batch normalization layers. However, it is important to highlight that this performance gain is exclusively relevant to the ID and OOD settings. The presence of informative calibration data in these scenarios enables effective updates of batch normalization statistics. In contrast, when employing randomly generated calibration data, the batch normalization statistics can become distorted, leading to potential performance degradation. Therefore, in this experimental context, we implement batch normalization statistic re-calibration exclusively for the ID and OOD scenarios, while refraining from its utilization in the data-free setting.

## C Additional Experiment

### C.1 Model Framework Conversion Time

We test the conversion time from different frameworks to ONNX. The results, as detailed in Tab. 6, reveal that even for the Jax models requiring dual conversions, the process completes within seconds. This indicates that the computational overhead incurred during the model conversion process is trivial compared to the time in pruning and training.

Table 6: Model Conversion time from different frameworks (Pytorch, TensorFlow, MXNet, Jax) to ONNX.

Model	conversion time (s)			
	PyTorch	TensorFlow	MXNet	Jax
ResNet-18	0.51s	2.47s	2.28s	5.47s
ResNet-50	2.01s	7.35s	7.36s	12.52s

### C.2 SPA with fine-tuning

In this section, we report the additional experiment results of performing pruning with SPA. Fig. 9 compares the SPA grouped versions of L1, SNIP, CroP, and GraSP to their original structured counterparts of L1, SNAP, Structured-CroP, Structured-GrasP on ResNet18 on CIFAR18. We observed that SPA versions of these pruning criteria always matches or outperforms their structured versions. Further, Tab. 7 and Tab. 8 shows additional results on DenseNet/ImageNet, Vit/ImageNet. Note the in the Vit Experiment, we only fine-tuned 30 epochs after pruning while DepGraph fine-tuned 300 epochs. We observe that SPA matches or outperform other previous methods.

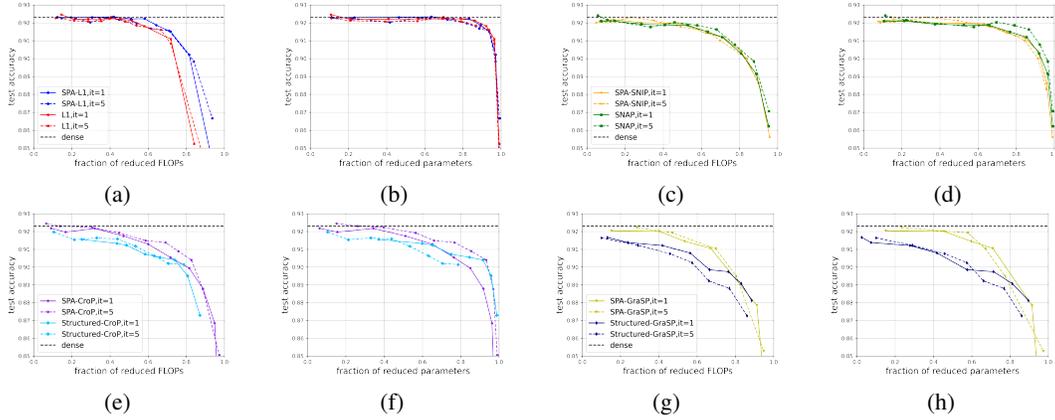


Figure 9: Trade off between accuracy and FLOPs/parameters with ResNet-18 on CIFAR-10 (see Figs. 9a to 9h). SPA efficiently implements both the structured and grouped versions of train-prune-finetune criteria like L1 and prune-train criteria like SNAP, CroP and GraSP

Table 7: Structured pruning of DenseNet-121 on ImageNet with fine-tuning. "N/R" indicate non-reported results in original papers.

method	top1 acc.	top5 acc.	RF	RP
Base Model	74.43%	91.97%	1×	1×
DepGraph Fang et al. (2023)	73.98%	N/R	2.09×	N/R
SPA-L1	74.39%	92.19%	2.09×	1.80×
OBSPA	74.62%	92.19%	1.78×	1.84×

Table 8: Structured pruning of ViT\_b\_16 on ImageNet with fine-tuning. "N/R" indicate non-reported results in original papers.

method	top1 acc.	top5 acc.	RF	RP
Base Model	81.43%	96.02%	1×	1×
DepGraph +EMA Fang et al. (2023)	79.58%	N/R	1.69×	N/R
DepGraph Fang et al. (2023)	79.17%	N/R	1.69×	N/R
SPA-L1	78.81%	94.20%	2.03×	2.05×
OBSPA	78.90%	94.30%	1.95×	1.98×

Table 9: Structured pruning of ResNet-101 on CIFAR-10 without finetuning

method	CIFAR-10		
	acc. drop	RF	RP
DFPC	4.95%	1.64x	2.22x
OBSPA (ID)	0.93%	1.59x	1.49x
OBSPA (OOD)	1.08%	1.59x	1.49x
OBSPA (DataFree)	1.51%	1.58x	1.49x

Table 10: Structured pruning of ResNet-101 on CIFAR-100 without finetuning

method	CIFAR-100		
	acc. drop	RF	RP
DFPC	9.40%	1.72x	1.53x
OBSPA (ID)	7.31%	1.68x	1.51x
OBSPA (OOD)	6.68%	1.68x	1.51x
OBSPA (DataFree)	9.95%	1.61x	1.47x

Table 11: Accuracy of Base Models of OBSPA experiment

Model	CIFAR-10		CIFAR-100	
	DFPC	ours	DFPC	ours
ResNet-50	94.99%	94.70%	78.85%	78.10%
ResNet-101	95.09%	94.48%	79.43%	81.05%
VGG-19	93.50%	96.04%	72.02%	81.05%

### C.3 SPA without fine-tuning

**OBSPA with ResNet-101 and Based Models.** In this section, we first report the additional experiment result of performing pruning after training with OBSPA on ResNet-101. These results are detailed in Tab. 9 and Tab. 10. We then provide the test accuracy of the base models used in our OBSPA and DFPC in Tabs. 4, 9 and 10 as Tab. 11.

**OBSPA on ImageNet-1k.** We also conduct pruning experiments without fine-tuning on the harder ImageNet-1k. DFPC does not present results for ImageNet without fine-tuning. We observed that, while using only less than 1000 calibration data samples or no calibration data, SPA presents non-trivial compression capabilities being able to maintain accuracy above 70% accuracy.

Table 12: Structured pruning of ResNet-50 on ImageNet without fine-tuning

method	accuracy	RF	RP
Base Model	76.15%	1x	1x
OBSPA (ID) - Low compression	74.27%	1.22×	1.16×
OBSPA (ID) - High compression	70.57%	1.43×	1.20×
OBSPA (OOD) - Low compression	71.60%	1.25×	1.18×
OBSPA (DataFree) - Low compression	70.13%	1.21×	1.19×

### C.4 Pruning Time of OBSPA

We compare the pruning time of our OBSPA algorithm to DFPC. The total pruning time of OBSPA includes all the necessary steps including building the computational graph, analyzing groups and applying OBSPA to prune and update parameters. For pruning a ResNet-50 on CIFAR-10 or CIFAR-100, DFPC takes 12 minutes, but our algorithm only takes 1.5 to 2 minutes. Pruning larger networks such as ResNet-101 and VGG-19 could also be completed within 6 minutes. For ImageNet-1k, a higher resolution dataset, DFPC also takes 6× more time than ours OBSPA.

The calibration data is processed batch by batch, so the batch size and batch number could also influence the pruning time. In our experiment, we use 2 batches of calibration data with batch size equal to 1024 in the CIFAR experiment and 7 batches of 128 data in the ImageNet-1k experiment.

Table 13: Pruning time for OBSPA and DFPC

Method	Model	Dataset	Pruning time
DFPC	ResNet-50	CIFAR-10/100	12 min
DFPC	ResNet-50	ImageNet-1k	38 min
OBSPA	ResNet-50	CIFAR-10/100	1.5-2 min
OBSPA	ResNet-101	CIFAR-10/100	3-6 min
OBSPA	VGG-19	CIFAR-10/100	3.5-4.5 min
OBSPA	ResNet-50	ImageNet-1k	5-6 min