ONNXPruner: ONNX-Based General Model Pruning Adapter

Dongdong Ren, Wenbin Li, Tianyu Ding, Lei Wang, *Senior Member, IEEE*, Qi Fan, Jing Huo, Hongbing Pan, and Yang Gao

***Abstract*—Recent advancements in model pruning have focused on developing new algorithms and improving upon benchmarks. However, the practical application of these algorithms across various models and platforms remains a significant challenge. To address this challenge, we propose ONNXPruner, a versatile pruning adapter designed for the ONNX format models. ONNXPruner streamlines the adaptation process across diverse deep learning frameworks and hardware platforms. A novel aspect of ONNXPruner is its use of node association trees, which automatically adapt to various model architectures. These trees clarify the structural relationships between nodes, guiding the pruning process, particularly highlighting the impact on interconnected nodes. Furthermore, we introduce a tree-level evaluation method. By leveraging node association trees, this method allows for a comprehensive analysis beyond traditional single-node evaluations, enhancing pruning performance without the need for extra operations. Experiments across multiple models and datasets confirm ONNXPruner's strong adaptability and increased efficacy. Our work aims to advance the practical application of model pruning.**

***Index Terms*—General model pruning, ONNX, tree-level evaluation, deep neural network.**

I. Introduction

In recent years, deep learning has achieved remarkable successes in many different fields. However, the deployment of deep neural network (DNN) models in practical applications is hindered by their substantial model size and intensive computational demands. To mitigate these challenges and accelerate inference speed while maintaining performance, researchers have paid much attention to DNN model compression, particularly for edge computing [1], [2], [3], [4], [5], [6], [7], [8].

Among the various model compression techniques, network pruning has been becoming a popular method, known for its efficiency and portability [9], [10]. Generally, network pruning methods fall into two categories [11]. One is unstructured pruning [2], [12], [13], [14], which creates sparse models by zeroing out weights of less important filters, but often requires specialized hardware or software to effectively handle sparse matrices. The other is structured pruning [15], [16], [17], [18], [19], [20], which removes unimportant or redundant filters, offering a more hardware-agnostic approach, and thus has become more widely used [21].

However, existing model pruning techniques face two main challenges in their application to device environments. First, existing pruning algorithms are typically tied to specific deep learning frameworks and it is difficult to transfer them across different platforms. For instance, developers often have to recreate pruning algorithms for each framework (like Keras, PyTorch, PaddlePaddle, *etc.*), convert them to ONNX (Open Neural Network Exchange) format [22], and then deploy on devices. This process is time-consuming and labor-intensive. Second, developers must tailor pruning procedures for different model structures. The complex internal connections in DNNs can lead to cascading effects when pruning nodes. Although some studies have attempted to address this issue by constructing chains of relationships between pruned and associated nodes [1], [3], [23], they typically require additional components and operations for evaluation, increasing computational complexity. Furthermore, these relational chains lack clear modeling of the associative structure, hindering the full utilization of associated nodes in assessing the importance of weights. This limitation can lead to suboptimal pruning decisions and potentially impact the overall effectiveness of the pruning process.

Aiming at solving the above challenges, we propose a novel *ONNX-based general model pruning adapter*, named ONNXPruner. Specifically, ONNXPruner leverages the ONNX framework to improve the interoperability of pruning algorithms across different application systems, as shown in Fig. 1. ONNX, being a cross-platform deep learning model exchange format, facilitates easy conversion and deployment between various deep learning frameworks and hardware platforms. It is widely supported by numerous deep learning frameworks and hardware acceleration platforms. Thus, we initiate pruning research on ONNX models to enhance the adaptability of pruning algorithms

Received 10 April 2024; revised 22 October 2024; accepted 13 March 2025. Date of publication 25 March 2025; date of current version 6 June 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62192783, Grant 62276128, and Grant 62406140, in part by the Young Elite Scientists Sponsorship Program by China Association for Science and Technology under Grant 2023QNRC001, in part by the Key Research and Development Program of Jiangsu Province under Grant BE2023019, and in part by Jiangsu Natural Science Foundation under Grant BK20221441 and Grant BK20241200. Recommended for acceptance by J. Wang. *(Corresponding author: Wenbin Li.)*

Dongdong Ren and Hongbing Pan are with the School of Electronic Science and Engineering, Nanjing University, Nanjing 210093, China (e-mail: rdd@smail.nju.edu.cn; phb@nju.edu.cn).

Wenbin Li, Qi Fan, Jing Huo, and Yang Gao are with the State Key Laboratory for Novel Software Technology and School of Intelligence Science and Technology, Nanjing University, Suzhou Campus, Suzhou 215163 , China (e-mail: liwenbin@nju.edu.cn; fanqi@nju.edu.cn; huojing@nju.edu.cn; gaoy@nju.edu.cn).

Tianyu Ding is with the Applied Sciences Group, Microsoft Corporation, Redmond, WA 98052 USA (e-mail: tianyuding@microsoft.com).

Lei Wang is with the School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia (e-mail: leiw@uow.edu.au).

Digital Object Identifier 10.1109/TPAMI.2025.3554560

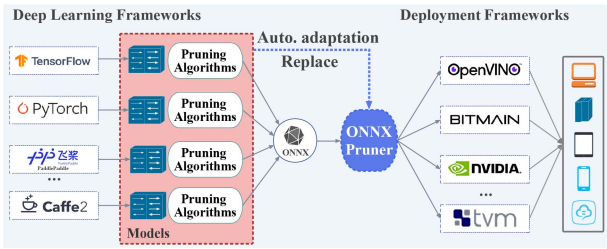


Fig. 1. Illustration of pruning algorithms in development and deployment. Existing pruning algorithms (red boxes) are tailored for specific development frameworks and necessitate manual adaptation for various model structures. Our work introduces ONNXPruner (blue box), a versatile model pruning adapter for ONNX format models, which provides automatic adaptation of pruning algorithms to models with diverse structures, in-depending on the development framework used.

in diverse applications. Crucially, to mitigate the cascading effects caused by pruning nodes, ONNXPruner constructs a node association tree for each pruned node within a model. This allows the pruning algorithm to automatically adjust to different model structures by clearly defining the relationships between a pruned node and its associated nodes. This feature enables the pruner to effectively track changes to associated nodes following the pruning of a node. Furthermore, while structured pruning typically involves the simultaneous removal of parameters from a pruned node and its associated nodes, existing pruning algorithms often only assess the pruned node, overlooking the impact of this action [16], [24], [25], as shown in Fig. 2(a). To address this limitation, we design a *tree-level pruning method* that utilizes node association trees to comprehensively evaluate channels, facilitating the removal of unimportant weights more effectively, as demonstrated in Fig. 2(b). This approach efficiently manages complex node associations without the need for additional components or operations.

To validate the performance of ONNXPruner, we tailor multiple popular pruning algorithms as baselines within ONNXPruner, including ℓ_1-norm [16], ℓ_2-norm [24], and Hrank [25]. ONNXPruner exhibits superior performance across a variety of model structures and tasks when compared to these baseline methods. To accommodate different model architectures, we establish a node attribute library for ONNXPruner. This library categorizes the model nodes' operator types into four attributes: pruned, next-no-process, next-process, and stop-process, which are essential for constructing node association trees. The library currently encompasses over 37 types of DNN operators and has been validated on both CNNs and Transformer models, including AlexNet [26], SqueezeNet [27], VGG [28], ResNet18 [29], FCN [30], PSPNet [31], ViT [32], and LLaMA-7B [33], among others.

In summary, we propose ONNXPruner, a versatile model pruning adapter designed to fit a wide range of model architectures through the interoperable ONNX format. The goal of ONNXPruner is not to invent new pruning algorithms but to provide a general-purpose pruning tool that allows application developers to effortlessly implement pruning algorithms. The main contributions of this work are as follows:

- To the best of our knowledge, this is the first attempt to develop a general pruning adapter for ONNX. This adapter significantly improves the interoperability of pruning algorithms across different device applications.
- We introduce node association trees to clearly define the relationships between pruned nodes and their associated nodes. This innovation enables pruning adapters to effectively manage various model structures.
- We develop a tree-level evaluation method utilizing node association trees. This approach allows for the assessment of diverse node connectivity structures without extra components and demonstrates superior performance.

II. Related Work

A. Model Conversion for Interoperability

In deep learning application systems, interoperability refers to the ability of software to exchange algorithms and models efficiently [34]. This software primarily encompasses development frameworks [35], [36], [37] and compilers [38], [39], [40] for deep learning. Given the vast array of development and deployment options, transferring algorithms between different frameworks is challenging. Moreover, compilers do not universally support models from all frameworks. Consequently, intermediaries like ONNX [22] have become crucial for enhancing the interoperability of deep learning software, as depicted in Fig. 1. ONNX uses the protocol buffers data structure and standardizes model representation through control flow in a graphical format, simplifying the conversion of models to the ONNX format. This facilitates the easy transfer of models across various frameworks, thus boosting interoperability.

In this context, we introduce a novel ONNX-based generic model pruning method. This approach, for the first time, circumvents the need for algorithmic porting between different frameworks. It allows for the seamless integration of pruned models with deep learning compilers, enhancing the interoperability of pruning algorithms from the development phase to deployment.

B. Model Pruning Method

Mainstream pruning methods fall into two categories: structural pruning [3], [16], [23], [24], [41] and unstructural pruning [42], [43], [44]. Unstructured pruning, which creates sparse matrices by zeroing out less important weights, often requires specialized hardware and software. Consequently, current research predominantly focuses on structured pruning. We now briefly review key studies in structured pruning, which is further divided based on the pruning timeline: before training, during training, and after training.

Pruning before training involves reducing the network size before the training process begins, utilizing concepts like the lottery hypothesis or sensitivity analysis to identify a lightweight sub-network early on [45], [46], [47]. This approach can expedite training by making the network sparse from the start [48], but its stability is generally low, potentially leading to significant performance variability in the pruned model.

Pruning during training employs a cyclical process of training, pruning, and evaluating to minimize performance loss. One common method [24] sets filters with low ℓ_n-norm values to zero at each training epoch, removing those zero-value filters upon

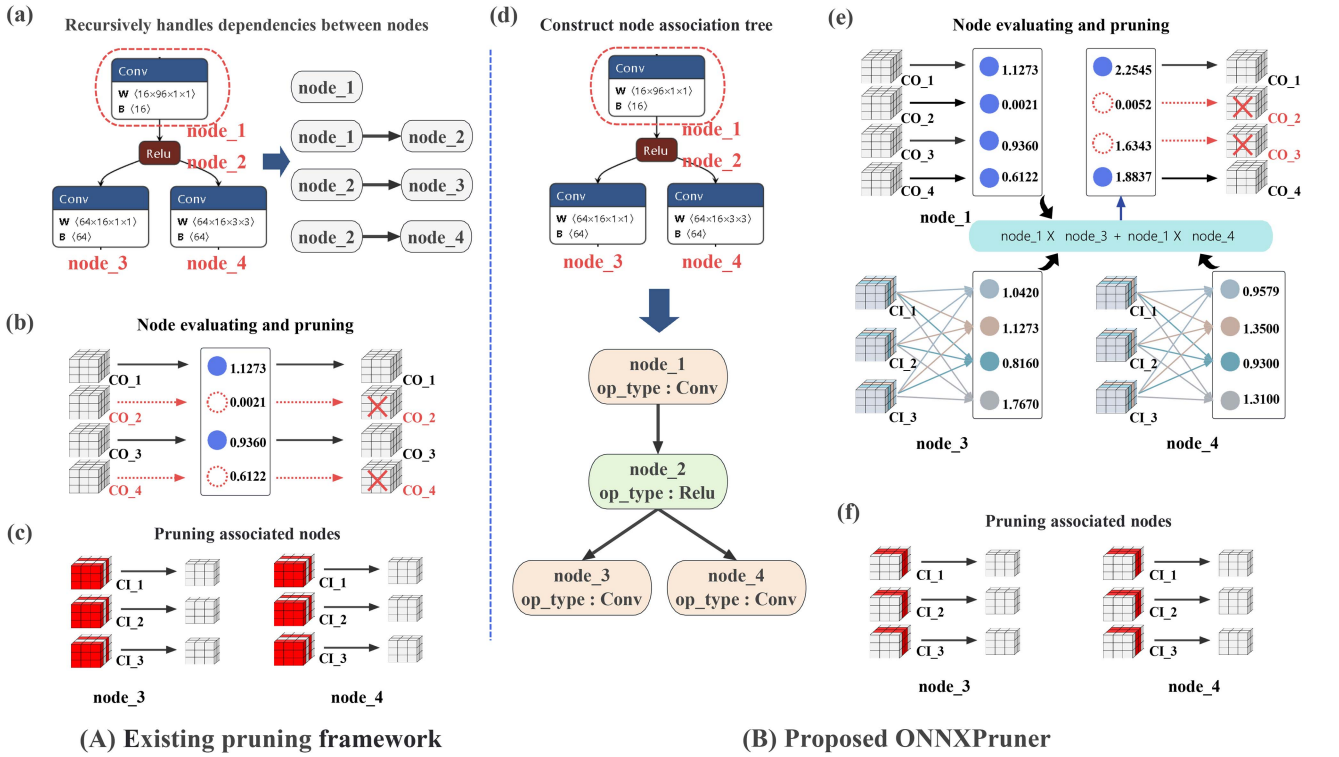


Fig. 2. Comparison between the existing pruning framework (A) and the proposed pruning framework (B). (a) The current method [1] recursively traverses each node to identify the associated nodes. (b) Evaluates filters based solely on the pruned node for pruning decisions. (c) Prunes associated nodes based on the evaluation in (b). (d) The proposed ONNXPruner constructs node association trees to represent the relationships between a pruned node and its associated nodes, presenting these relationships hierarchically. (e) Employs a tree-level pruning strategy, leveraging the node association tree to jointly evaluate and prune the filters of both the pruned node and associated nodes. (f) Prunes associated nodes following the evaluation in (e). CO and CI represent the output and input channels of the Conv kernel, respectively.

training completion. Other strategies incorporate reinforcement learning or neural architecture search to dynamically prune filters during the training process [49], [50]. These methods help mitigate the adverse effects of filter removal but extend the training duration and necessitate specific adjustments during training.

Pruning after training follows a train-prune-finetune approach, where a large model is first trimmed down based on certain criteria and then refined to recover performance. The crucial aspect here is defining the importance of filters, typically through the calculation of their ℓ_n-norm [16], [24]. This assumption holds that filters with smaller ℓ_n-norms are less crucial for the network. Such methods are straightforward, requiring no extra fine-tuning computations and are commonly employed in various applications [21], [51].

C. General Model Pruning Method

In deep neural networks, the interconnections between neural nodes are generally intricate [29], [52]. When a node is pruned or reduced, its associated nodes must be adjusted to maintain normal inference functionality. For instance, in structural pruning, pruning some filters in a node necessitates the removal of corresponding input channels in the filters of the subsequent layer. In simpler models, it's feasible to manually determine which layers to prune, a common approach in many existing methods [16], [53]. However, this becomes challenging with complex model architectures, especially those featuring intricate residual connections and layer operations.

Recent efforts have aimed to unravel these complex layer relationships. Liu et al. [3] and You et al. [23] propose pruning algorithms for residual structures, employing a uniform mask to prune the two input nodes of the add operator within residual blocks. Similarly, Fang et al. [1] and Chen et al. [20] introduce automatic pruning framework that models relationships between layers and groups dependent nodes. For residual structures, thanks to their identified groups, both of them enable the pruning of the two input nodes of the add operator. In addition, we noticed that a work around the same time as ours also pruned the ONNX model [54], but it also introduced masks to handle complex connections. Nonetheless, these methods have several limitations: (1) They require adding new structures to the model or retraining it to handle complex node couplings, like residual connections; (2) Their approach of recursively grouping associated nodes lacks a clear construction of relationships within the entire group; (3) These methods are developed primarily using PyTorch, resulting in limited interoperability across different deep learning frameworks.

To address these challenges, we propose an ONNX-based general model pruning framework in this paper. It clearly establishes a node correlation tree between a pruned node and its

TABLE I
THE ONNX CONVERSION PROCESS INCLUDES CONSTRUCTING NODE GRAPHS, TRANSFORMING NODES FOR EQUIVALENCE, OPTIMIZING NODES, SERIALIZING THE MODEL, CHECKING FOR CORRECTNESS, AND VERIFYING PERFORMANCE

Step	Operation	Definition
S1	Load Model	Loading models of native frameworks
S2	Representation	ONNX graph tracing dynamic graph
S3	Node conversion	Graph nodes replaced by ONNX equivalents
S4	Optimization	Elimination of redundant nodes
S5	Export	Model serialized into protocol buffer
S6	Check	Syntactic checks and semantic checks
S7	Validate	Test whether the inference result is correct

associated nodes, without adding extra components or necessitating re-training.

III. METHOD

A. ONNX Model Converter and Runtime

To enhance the interoperability of pruning algorithms, we propose a universal pruning adapter named ONNXPruner, leveraging the ONNX framework. A key aspect of ONNXPruner is its ability to standardize models from diverse frameworks into the ONNX format. For instance, using PyTorch, we first load a pre-trained deep learning model and then convert it to ONNX using PyTorch's built-in conversion tool. This conversion encompasses several steps: representing the ONNX graph, replacing node operators equivalently, optimizing nodes, storing the model, checking the model structure, and verifying performance, as detailed in Table I. Since the model requires fine-tuning after pruning, we avoid operator fusion and other optimizations that could alter the original structure during conversion.

After converting models from various frameworks to ONNX, the proposed ONNXPruner will automatically prune these ONNX models. For the evaluation and fine-tuning of pruned models, we use ONNX Runtime (ORT) [55], an engine that supports ONNX-based models for both inference and training through backpropagation. ORT offers efficient training and inference across different platforms and hardware, including CPUs and GPUs, by integrating with hardware-specific libraries through flexible interfaces. This capability of ORT facilitates the application and integration of diverse pruning algorithms within ONNXPruner.

B. Construct Node Association Tree

In this paper, a "pruned node" refers to a model layer requiring pruning, while "associated nodes" are those that must be adjusted following the pruning of a pruned node, including all nodes in the sequence. To automate model pruning, identifying the associated nodes of a pruned node and understanding their relationships is crucial. Our goal is to locate all associated nodes of a pruned node and group them. To accomplish this, we initially create a node attribute library categorizing nodes into four types based on their operator roles: pruned, next-no-process, next-process, and stop-process, as detailed in Table II. Subsequently, leveraging the node attribute library, we develop a clearer method for modeling node relationships, the *node association tree*.

Algorithm 1. Construct Node Association Tree.

Input: ONNX model, pruning nodels $N[0, 1..., n]$
Output: Node association Trees

```
Function Insertchild(fathernode, childnode)
    fathernode → child = childnode
    return fathernode → child
for i = {0, 1..., n} do
    node = Tree(N_i)
    while node not NULL do
        if The next node of N_i in the node graph then
            for Next node NN do
                if NN then
                    node = Insertchild(node, NN)
                end
                else
                    node = NULL
                end
            end
        end
    end
end
```

The construction of the node association tree begins by navigating the ONNX node graph, starting from the pruned node as the tree's root, and identifying all nodes receiving output from this root node as their input, considered the root's children. Each child node is then evaluated based on its attribute. For next-process or next-no-process nodes, this examination continues, treating each as a new parent node. The search concludes for stop-process nodes, which are deemed leaf nodes. This approach, as summarized in Algorithm 1, facilitates the creation of node association trees for pruned nodes in any model, provided that the model's operator types are represented in the node attribute library. Fig. 3 illustrates the construction of a node association tree for a pruned node in SqueezeNet [27].

C. Tree-Level Pruning

The standard approach to model pruning involves pruning a pre-trained model and then fine-tuning the pruned structure to mitigate the performance loss caused by pruning. While some studies incorporate pruning at the beginning or during the training phase, the practical application of these methods is often complicated by the large dataset sizes and extended training durations. The most commonly adopted criterion for pruning in applications is the ℓ_n-norm of the model weights [16], [24], a practice we adhere to develop our pruning adapter and evaluation method.

The ℓ_n-norm technique prunes filters by comparing the ℓ_n-norm of all channel weights in a single node, subsequently removing the corresponding input channels of the associated

TABLE II
NODE ATTRIBUTE LIBRARY

Node attribute	Operator type	Tree
pruned	Conv, ConvTranspose, Gemm, MatMul, Mul	root
next-no-process	Relu, Sigmoid, Softmax, Tanh, MaxPool, AveragePoo, Flatten GAP, Pad, Reshape, Transpose, ReduceMean, Pow, ReduceMax Sqrt, Erf, Unsqueeze, Resize, Slice, Cast, ReduceMean, Neg	child
next-process	Add, Concat, BatchNormalization, Sub, Div, Gather Split, Slice, Tile, GatherElements	child
stop-process	Conv, ConvTranspose, Gemm, MatMul, Mul	leaf

We categorize node attributes according to their operator types to facilitate the construction of node association trees. We distinguish operators into four node attributes: pruned refers to a model node requiring pruning, next-no-process for associated nodes that do not need further processing, next-process for those that do, and stop-process for nodes where the search for further children ends post-processing.

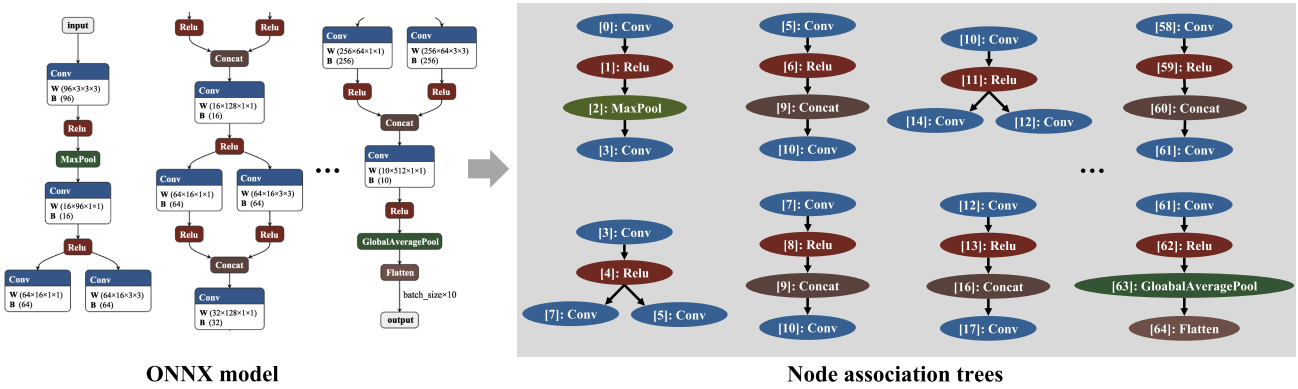


Fig. 3. An example for constructing the node association trees. We use the node graph of the ONNX model to build a node association tree for each pruned node. The attributes within these trees are assigned based on the operator type: the pruned node serves as the root, child nodes are tagged as 'next' indicating further exploration, and leaf nodes are marked as 'stop' indicating the end of the branch.

nodes in the following layer. However, assessing weight importance based solely on a single node's weights may not accurately reflect the true significance, especially since the parameters of the pruned node and its associated nodes are removed simultaneously. To address this issue, we put forward a *tree-level pruning* method utilizing node association trees, which allows for a more accurate evaluation of which channels to prune, thus enabling the removal of less important channels more securely without adding extra components.

We identify four types of tree-level pruning structures based on node association trees: one-to-one, one-to-many, many-to-one, and many-to-many, as shown in Fig. 4.

One-to-one connections are fundamental in DNN models, where the output of a layer serves as the input for the next. Specifically, for input feature F_n, the weight W_n^i convolves with it, producing the ith feature map for the subsequent layer. The corresponding filter in the next layer, $W_{n+1}^{k,i}$, is directly linked to W_n^i's output channel, as shown in Fig. 4(a). The evaluation of a pruned node in a one-to-one structure is thus defined as:

$$\left\|W_n^i\right\|_1 \times \sum_{k=1}^{j} \left\|W_{n+1}^{k,i}\right\|_1, \tag{1}$$

where j represents the dimensional index of W_{n+1}.

One-to-many connections indicate a single pruned node connects to multiple associated nodes. For input feature F_n, W_n^i convolves with it and outputs the ith feature map affecting multiple subsequent filters, such as $W_{n+1}^{k,i}$ and $W_{n+2}^{k,i}$, shown in Fig. 4(b). Therefore, the evaluation of the pruned node in a one-to-many structure is given by:

$$\left\|W_n^i\right\|_1 \times \left(\sum_{k=1}^{j_1} \left\|W_{n+1}^{k,i}\right\|_1 + \sum_{k=1}^{j_2} \left\|W_{n+2}^{k,i}\right\|_1\right), \tag{2}$$

where j_1 represents the dimensional index of W_{n+1} and j_2 represents the dimensional index of W_{n+2}.

Many-to-one connections, such as in residual structures, involve combining outputs from multiple nodes into a single subsequent node. This is shown in Fig. 4(c), where W_n^i and W_{n-1}^i convolve with their respective inputs (F_n and F_{n-1}) and their outputs are summed for the next layer's computation. The evaluation formula for a pruned node in a many-to-one structure is:

$$\left(\left\|W_{n-1}^i\right\|_1 + \left\|W_n^i\right\|_1\right) \times \sum_{k=1}^{j} \left\|W_{n+1}^{k,i}\right\|_1, \tag{3}$$

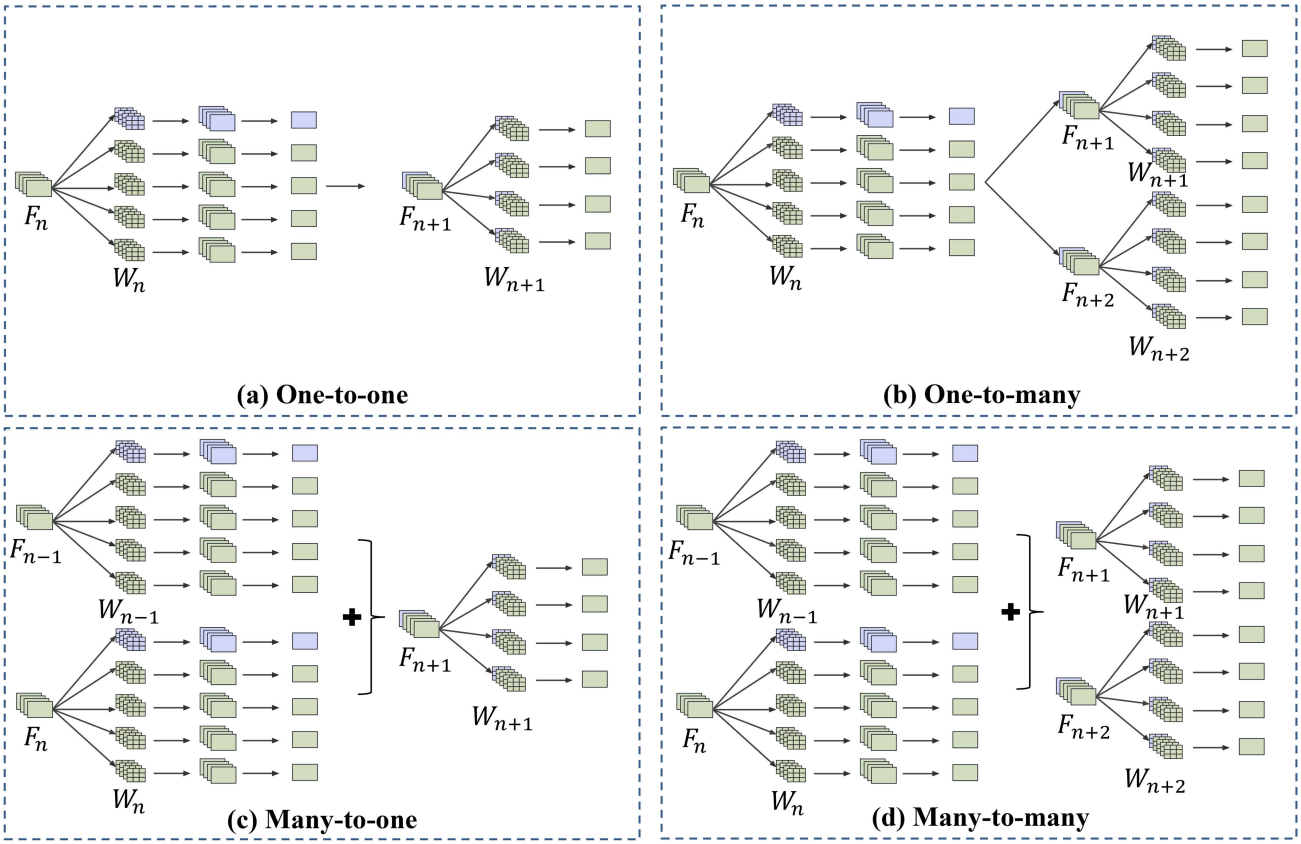


Fig. 4. We illustrate four basic configurations of pruned and associated nodes, using convolution as a representative example for simplicity. We omit associated nodes like ReLU and pooling, which do not require processing in this context. (a) One-to-one represents the standard structure in DNNs, where the output channel of one layer feeds directly into the input channel of each filter in the next layer. (b) One-to-many is prevalent in models that incorporate feature reuse, such as SqueezeNet and Inception. (c) Many-to-one typifies the conventional residual structure found in networks. (d) Many-to-many combines the characteristics of (b) and (c), representing a hybrid structure that integrates feature reuse with residual connections.

Many-to-many connections combine aspects of one-to-many and many-to-one structures. The evaluation for a pruned node in such a structure is calculated as:

$$\left(\left\|W_{n-1}^{i}\right\|_1+\left\|W_{n}^{i}\right\|_1\right)\times\left(\sum_{k=1}^{j_1}\left\|W_{n+1}^{k,i}\right\|_1+\sum_{k=1}^{j_2}\left\|W_{n+2}^{k,i}\right\|_1\right). \tag{4}$$

In the following experimental section, we demonstrate that incorporating tree-level pruning with certain methods can yield performance on par with contemporary approaches.

IV. Experiments

A. Settings

Dataset. To evaluate the performance of ONNXPruner, we utilize three well-established benchmark datasets: CIFAR-10, CIFAR-100 [56], ImageNet [57] and PASCAL VOC 2012 [58]. Both CIFAR-10 and CIFAR-100 comprise 60,000 images each, split into 50,000 for training and 10,000 for testing, across 10 and 100 classes, respectively. The ImageNet dataset contains 1.2 million training images and 50,000 validation images across 1,000 classes. PASCAL VOC 2012, a benchmark for semantic segmentation, includes 20 object classes plus one background class, with 1,464 training images, 1,449 validation images, and 1,456 testing images. The training set is augmented by the Semantic Boundaries Dataset [59], totaling 10,582 training images.

Model. We follow the widely-used models in existing pruning studies to verify for ONNXPruner's efficacy. For CIFAR datasets, ONNXPruner is tested on various architectures, including AlexNet [26], VGG16 [28], VGG19 [28], SqueezeNet [27], ResNet18 [29], and ViT-B_16 [32], with each model trained independently. For the ImageNet dataset, we evaluate ONNXPruner on ResNet50 [29]. For the PASCAL VOC 2012 dataset, we evaluate ONNXPruner on FCN [30] and PSPNet [31].

Large Language Model. To demonstrate the effectiveness and generalization ability of ONNXPruner, we tested it on the open-source large language model LLaMA-7B [33]. We followed the LLaMA evaluation protocol [60] and performed classification on seven diverse datasets: BoolQ [61], PIQA [62], HellaSwag [63], WinoGrande [64], ARC-easy [65], ARC-challenge [65], and OpenbookQA [66].

B. Effectiveness of Tree-Level Pruning

The proposed tree-level pruning method evaluates the importance of filters by considering both pruned and associated nodes, which distinguishes it from evaluations that focus on single

TABLE III
PRUNING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR VGG16 ON CIFAR-10 WITH DIFFERENT PRUNING RATES

Unpruned acc. 92.64, Params: 14.16M, FLOPs: 0.58G				
Layerwise PR	0.3	0.5	0.7	0.9
Sparsity	49.89%	74.02%	90.19%	98.62%
Speedup	1.89×	3.37×	7.57×	30.11×
ℓ_1-norm [16]	91.34	90.11	88.77	84.58
ONNXPruner (ℓ_1)	92.29 (↑0.95)	91.07 (↑0.96)	90.16 (↑1.39)	85.26 (↑0.68)
ℓ_2-norm [24]	91.34	90.16	88.75	84.58
ONNXPruner (ℓ_2)	92.26 (↑0.92)	91.00 (↑0.84)	90.15 (↑1.40)	85.17 (↑0.59)
Hrank* [25]	91.52	90.47	88.91	84.51
ONNXPruner (Hrank*)	92.33 (↑0.81)	91.50 (↑1.03)	90.19 (↑1.28)	85.20 (↑0.69)

Layerwise PR stands for layerwise pruning ratio. Speedup stands for the ratio of the unpruned FLOPs compared to the pruned model.

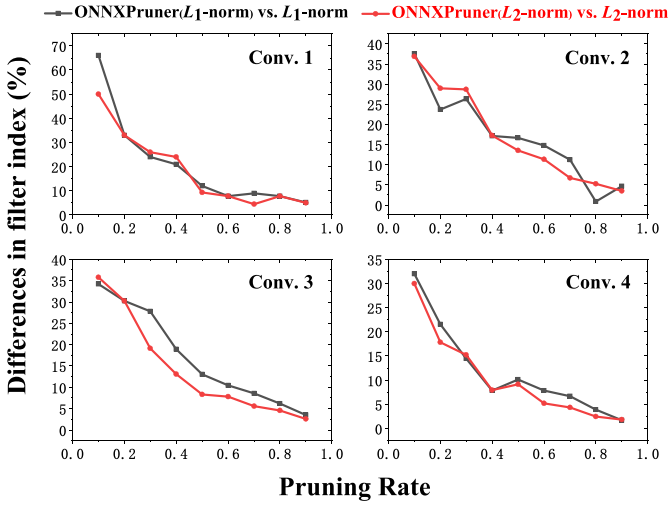


Fig. 5. Differences in filter index of ONNXPruner (ℓ_n-norm) vs. ℓ_n-norm.

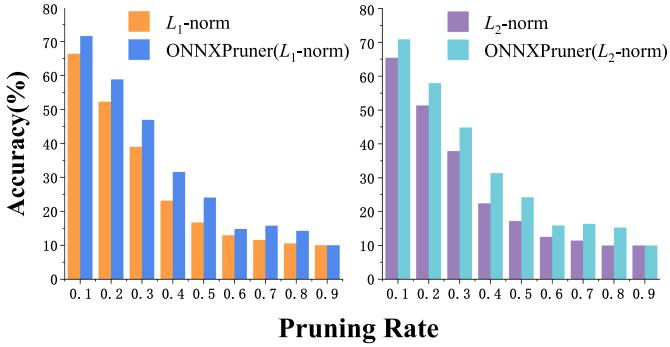


Fig. 6. Pruning of ONNXPruner (ℓ_n-norm) and ℓ_n-norm without fine-tuning.

nodes [16], [24], [25]. This approach is illustrated in Fig. 5, showing the difference in filter selection between ONNXPruner (using ℓ_1-norm and ℓ_2-norm) and the traditional ℓ_1-norm or ℓ_2-norm methods, with AlexNet [26] as the example. The difference in filter indices between the methods is calculated as follows:

$$\frac{\text{index}(\text{ONNXPruner}(\ell_n\text{-norm})) \cap \text{index}(\ell_n\text{-norm})}{\text{Count}(\text{index}(\ell_n\text{-norm}))}. \quad (5)$$

Results in Fig. 5 indicate a notable discrepancy in filter indices between ONNXPruner (ℓ_n-norm) and traditional methods across various layers, especially at lower pruning rates (below 0.3), where differences exceed 20%. This gap diminishes as the pruning rate increases, mainly because of the broader inclusion of pruned channels.

To assess the efficacy of tree-level pruning without the influence of finetuning, we test the accuracy of models pruned with ONNXPruner, as shown in Fig. 6. Using CIFAR-10 as the dataset and pruning rates ranging from 0.1 to 0.9, we observe that ONNXPruner generally enhances accuracy across various pruning levels. Notably, at a pruning rate of 0.9, accuracy approaches that of random guessing, reflecting the extensive loss of model capacity due to excessive pruning.

C. Results on CIFAR

First, we validate the performance of ONNXPruner across various pruning rates on the VGG16 model using the CIFAR-10 dataset, as shown in Table III. We set specific layerwise pruning rates for all convolutional and fully connected layers, where a rate of 0.3 implies removing 30% of the parameters in these layers. The pruned models are then fine-tuned for 10 epochs using ONNXRuntime with an initial learning rate of 10^{-3}. We incorporate baseline algorithms such as ℓ_1-norm [16], ℓ_2-norm [24], and Hrank [25] into ONNXPruner for tree-level filter importance evaluation, with Hrank* indicating a combined approach of using feature map rank for convolutional layers and ℓ_1-norm for fully connected layers. The results demonstrate that the proposed ONNXPruner, without any additional constraints, can integrate various weight evaluation algorithms and outperform the baselines at various pruning rates.

To further evaluate the performance of ONNXPruner on different models, we conduct additional tests with a fixed pruning rate of 0.5, detailed in Table IV. Comparing ONNXPruner against other methods like Taylor-FO [67], GReg-2 [68],

TABLE IV
PRUNING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR VARIOUS MODELS ON CIFAR-10 AND CIFAR-100

Data	Method	AlexNet	SqueezeNet	VGG16	VGG19	ResNet18	ViT-B_16
CIFAR-10	Unpruned	79.27	83.15	92.64	92.58	93.02	98.67
	Taylor-FO [67]	77.65	80.01	90.70	90.63	90.91	96.02
	GReg-2 [68]	77.98	80.69	91.01	91.03	91.13	–
	reimpl. (ℓ_1-norm) [69]	78.09	80.71	91.08	91.24	91.35	96.17
	TPP [70]	78.19	81.93	91.47	**91.89**	**91.72**	–
	DepGraph (ℓ_1-norm) [1]	78.01	81.86	91.29	91.64	91.38	96.62
	ℓ_1-norm [16]	77.08	79.32	90.11	90.21	90.38	95.94
	ONNXPruner (ℓ_1)	78.13 (↑1.05)	81.76 (↑2.44)	91.07 (↑0.96)	91.77 (↑1.56)	91.68 (↑1.30)	**96.82** (↑0.88)
	ℓ_2-norm [24]	77.01	79.29	90.16	90.51	90.44	95.31
	ONNXPruner (ℓ_2)	78.06 (↑1.05)	81.78 (↑2.49)	91.00 (↑0.84)	91.91 (↑1.40)	91.73 (↑1.29)	96.14 (↑0.83)
	Hrank* [25]	77.33	79.57	90.47	90.59	90.40	–
	ONNXPruner (Hrank*)	**78.20** (↑0.87)	**82.00** (↑2.43)	**91.50** (↑1.03)	91.87 (↑1.28)	91.61 (↑1.21)	–
CIFAR-100	Unpruned	55.41	69.41	72.93	72.94	75.21	90.97
	Taylor-FO [67]	52.34	66.32	69.52	70.24	72.04	85.80
	GReg-2 [68]	52.47	66.51	69.70	70.49	72.17	–
	reimpl. (ℓ_1-norm) [69]	52.89	66.79	70.29	70.88	72.69	86.02
	TPP [70]	53.20	**67.00**	70.44	71.33	**73.78**	–
	DepGraph (ℓ_1-norm) [1]	53.14	66.83	70.37	71.22	73.16	86.47
	ℓ_1-norm [16]	52.03	66.01	69.35	70.01	71.80	85.41
	ONNXPruner (ℓ_1)	53.17 (↑1.14)	66.70 (↑0.69)	70.43 (↑1.08)	71.23 (↑1.22)	73.01 (↑1.21)	**86.90** (↑1.49)
	ℓ_2-norm [24]	52.09	66.08	69.31	68.98	71.91	85.02
	ONNXPruner (ℓ_2)	53.16 (↑1.07)	66.74 (↑0.66)	70.35 (↑1.04)	71.07 (↑1.09)	73.15 (↑1.24)	86.77 (↑1.75)
	Hrank* [25]	52.19	66.35	69.78	70.26	72.37	–
	ONNXPruner (Hrank*)	**53.29** (↑1.10)	66.91 (↑0.56)	**70.50** (↑0.72)	**71.66** (↑1.40)	73.70 (↑1.33)	–

We pruning 50% of the parameters in all convolutional layers and fully connected layers on models AlexNet [26], SqueezeNet [27], VGG16 [28], VGG19 [28], ResNet18 [29], and ViT-B_16 [32]. Taylor-FO [67], GReg-2 [68], reimpl. (ℓ_1-norm) [69], TPP [70], DepGraph [1] are existing methods that perform well in optimizing the evaluation strategies. Since these methods require additional components or operations, we implemented these on Pytorch and adapted them to different model structures, including residual connections. We implemented the ℓ_1-norm [16], ℓ_2-norm [24], and Hrank [25] methods on the ONNX format and embedded them into ONNXPruner. The blue color indicates the performance change after ONNXPruner embeds and optimizes these methods.

TABLE V
PRUNING-DURING-TRAINING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR VARIOUS MODELS ON CIFAR-10

Method	AlexNet	SqueezeNet	VGG16	ResNet18
FPGM [17]	77.08	79.32	90.11	90.38
ONNXPruner (FPGM)	**78.13**	**81.76**	**91.07**	**91.68**
SFP [24]	77.01	79.29	90.16	90.44
ONNXPruner (SFP)	**78.06**	**81.78**	**91.00**	**91.73**

reimpl. (ℓ_1-norm) [69], TPP [70] and DepGraph (ℓ_1-norm) [1], which focus on optimizing filter importance evaluation but require extra components or constraints, ONNXPruner shows superior or comparable performance. The "-" indicates models incompatible with certain algorithms, thus not yielding results.

We also explore the adaptability of ONNXPruner by integrating it with during-training pruning methods like FPGM [17] and SFP [24]. As shown in Table V, ONNXPruner not only accommodates these methods but enhances their optimization performance. ONNXPruner's design facilitates easy integration with existing and new pruning methods by simply updating the criteria for importance evaluation. Additionally, the overhead for constructing the node association tree and performing tree-level model evaluation on a model like ResNet is approximately 1.93 seconds on an RTX 3090 GPU, a minimal delay compared to the overall training duration of about 2.2 hours.

D. Results on ImageNet

To evaluate ONNXPruner on a larger dataset, we test its performance and efficiency on ImageNet for ResNet50. We show several pruning methods, including ℓ_1-norm [16], Hrank [25], FPGM [17], and SFP [24], and their performance when integrated into ONNXPruner. The results, as presented in Table VII, show that ONNXPruner outperforms each pruning method by achieving a higher Pruned Top-1 accuracy with

TABLE VI
PRUNING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR LLAMA-7B WITH 20% OF THE PARAMETERS PRUNED

Method	BoolQ	PIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Average
Unpruned	73.18	78.35	72.99	67.01	67.45	41.38	42.40	63.25
ℓ_1-norm [16]	58.93	**76.33**	50.82	63.77	58.92	35.58	26.00	52.91
ONNXPruner (ℓ_1)	**62.15**	76.01	**59.72**	**65.81**	**61.33**	**36.71**	**36.02**	**56.82**
ℓ_2-norm [24]	56.64	**76.01**	51.20	60.46	60.48	35.01	28.40	52.60
ONNXPruner (ℓ_2)	**61.00**	75.84	**59.83**	**66.37**	**62.22**	**36.03**	**37.15**	**56.92**

TABLE VII
PRUNING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR RESNET50 ON IMAGENET

Method	Pruned top-1	Top-1 drop	Speedup
ℓ_1-norm [16]	74.23	1.90	1.78×
ONNXPruner (ℓ_1)	**74.89**	**1.24**	**1.78**×
Hrank [25]	74.98	1.15	1.78×
ONNXPruner (Hrank)	**75.44**	**0.69**	**1.78**×
FPGM [17]	75.01	1.12	1.59×
ONNXPruner (FPGM)	**75.43**	**0.70**	**1.59**×
SFP [24]	74.61	1.52	1.76×
ONNXPruner (SFP)	**75.22**	**0.91**	**1.76**×

TABLE VIII
PRUNING RESULTS (MEAN IOU) FOR VARIOUS MODELS ON PASCAL VOC 2012

Method	FCN [30]	PSPNet [31]
Unpruned	62.2	82.6
ℓ_1-norm [16]	60.3	78.7
ONNXPruner (ℓ_1)	61.0 (↑0.7)	80.1 (↑1.4)
ℓ_2-norm [24]	60.3	78.6
ONNXPruner (ℓ_2)	61.1 (↑0.8)	79.8 (↑1.2)
Hrank* [25]	60.7	78.7
ONNXPruner (Hrank*)	61.6 (↑0.9)	80.1 (↑1.4)

a smaller decrease in Top-1 accuracy, all while maintaining consistent speed improvements.

E. Results on PASCAL VOC 2012

Beyond image classification, we extend the application of ONNXPruner to the task of image segmentation, testing it on two widely-used models: FCN [30] and PSPNet [31]. The evaluation, using mean Intersection over Union (mean IoU) as the metric and presented in Table VIII, involves baseline methods such as ℓ_1-norm [16], ℓ_2-norm [24], and Hrank [25] integrated within ONNXPruner. Adapting these methods to FCN and PSPNet manually is a time-consuming process, but the proposed ONNXPruner simplifies this, allowing for quick adaptation to model structures for pruning. The use of a tree-level pruning strategy enables more accurate filter importance assessment. The mean IoU results in Table VIII indicate that ONNXPruner delivers superior performance compared to baseline methods.

TABLE IX
PRUNING RESULTS (CLASSIFICATION ACCURACY IN PERCENTAGE POINT) FOR VGG16 ON CIFAR-10

Framework	Unpruned	Ori.(ℓ_1-norm [16])	ONNXPruner(ℓ_1)
PyTorch	92.64	90.11	91.07
Keras	92.35	90.08	91.09
PaddlePaddle	93.17	90.34	91.15
MXNet	93.20	90.26	91.18

We pruning 50% of the parameters in all convolutional layers and fully connected layers.

F. Results on Different Frameworks

To verify the effectiveness of ONNXPruner on ONNX models converted from different frameworks, we examined the performance of VGG16 trained with PyTorch, Keras, PaddlePaddle, and MXNet. We first pruned and fine-tuned the models in their four original frameworks. Then, we converted the original models to ONNX format and applied ONNXPruner for pruning and fine-tuning. The results, as shown in Table IX, indicate that ONNXPruner can be successfully applied to ONNX models from different frameworks, and there are corresponding performance improvements compared to the original frameworks.

G. Results on LLaMA

Extending beyond traditional deep neural networks, we applied ONNXPruner to the large language model domain. We successfully pruned 20% of the parameters from LLaMA-7B [33], reducing its size from 6.74 billion to 5.43 billion parameters. For fine-tuning, we utilized a clean version of the Alpaca dataset [71], consisting of 50,000 samples, and trained for three epochs. The results presented in Table VI demonstrate ONNXPruner's ability to effectively identify the LLaMA-7B's coupling structure, resulting in classification accuracy improvements of 3.91% and 4.32% compared to the ℓ_1-norm [16] and ℓ_2-norm [24] pruning methods, respectively. These findings underscore ONNXPruner's versatility and efficacy across diverse model architectures, including state-of-the-art large language models.

V. Conclusion

This paper proposes ONNXPruner, a general model pruning adapter for ONNX models. It automates the application of pruning algorithms across different model structures, aiding developers in enhancing practical applications. Future developments for ONNXPruner will focus on expanding support for more operator types, accommodating more model architectures like Bidirectional LSTM/GRU, and further enhancing the efficiency of tree-level pruning.

References

[1] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 16091–16101.

[2] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[3] L. Liu et al., "Group fisher pruning for practical network compression," in *Proc. 38th Int. Conf. Mach. Learn.*, 2021, pp. 7021–7032.

[4] Y. Jing, Y. Yang, X. Wang, M. Song, and D. Tao, "Meta-aggregator: Learning to aggregate for 1-bit graph neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 5281–5290.

[5] O. Kuchaiev et al., "NEMO: A toolkit for building AI applications using neural modules," 2019, *arXiv: 1909.09577*.

[6] Z. Yao et al., "WeNet: Production oriented streaming and non-streaming end-to-end speech recognition toolkit," 2021, *arXiv:2102.01547*.

[7] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019.

[8] W. Niu et al., "GRIM: A general, real-time deep learning inference framework for mobile devices based on fine-grained structured weight sparsity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6224–6239, Oct. 2022.

[9] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 14913–14922.

[10] S. Chao, Z. Wang, Y. Xing, and G. Cheng, "Directional pruning of deep neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2020, pp. 13986–13998.

[11] Y. He and L. Xiao, "Structured pruning for deep convolutional neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, pp. 2900–2919, May 2024.

[12] S. J. Kwon, D. Lee, B. Kim, P. Kapoor, B. Park, and G. Wei, "Structured compression by weight encryption for unstructured pruning and quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1906–1915.

[13] T. Chen et al., "Orthant based proximal stochastic gradient method for l_1-regularized optimization," in *Proc. Mach. Learn. Knowl. Discov. Databases - Eur. Conf.*, 2020, pp. 57–73.

[14] T. Chen et al., "Neural network compression via sparse optimization," 2020, *arXiv: 2011.04868*.

[15] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2755–2763.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. 5th Int. Conf. Learn. Representations*, 2017, pp. 1–13.

[17] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4340–4349.

[18] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 6682–6691.

[19] T. Chen et al., "Only train once: A one-shot neural network training and pruning framework," in *Proc. Conf. Neural Inf. Process. Syst.*, 2021, pp. 19637–19651.

[20] T. Chen, L. Liang, T. Ding, Z. Zhu, and I. Zharkov, "OTOV2: Automatic, generic, user-friendly," in *Proc. 11th Int. Conf. Learn. Representations*, 2023, pp. 1–26.

[21] Y. Liu, Z. Shu, Y. Li, Z. Lin, F. Perazzi, and S. Kung, "Content-aware GAN compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 12156–12166.

[22] "Onnx," 2019. [Online]. Available: https://onnx.ai/

[23] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 2130–2141.

[24] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2234–2240.

[25] M. Lin et al., "Hrank: Filter pruning using high-rank feature map," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1526–1535.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.

[27] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: Alexnet-level accuracy with 50× fewer parameters and! 0.5 mb model size," 2016, *arXiv:1602.07360*.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[30] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.

[31] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 6230–6239.

[32] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv: 2010.11929*.

[33] H. Touvron et al., "Llama: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[34] P. Jajal et al., "Analysis of failures and risks in deep learning model converters: A case study in the onnx ecosystem," 2023, *arXiv:2303.17708*.

[35] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[36] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 265–283.

[37] T. Chen et al., "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*.

[38] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 578–594.

[39] E. Jeong, J. Kim, S. Tan, J. Lee, and S. Ha, "Deep learning inference parallelization on heterogeneous processors with tensorrt," *IEEE Embed. Syst. Lett.*, vol. 14, no. 1, pp. 15–18, 2022.

[40] "Openvino documentation," 2021. [Online]. Available: https://docs.openvino.ai/latest/home.html

[41] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 3048–3056, Dec. 2019.

[42] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4857–4867.

[43] N. Lee, T. Ajanthan, S. Gould, and P. H. Torr, "A signal propagation perspective for pruning neural networks at initialization," 2019, *arXiv:1906.06307*.

[44] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A far-sighted alternative of magnitude-based pruning," 2020, *arXiv:2002.04809*.

[45] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *Proc. 7th Int. Conf. Learn. Representations*, 2019, pp. 1–15.

[46] C. Wang, G. Zhang, and R. B. Grosse, "Picking winning tickets before training by preserving gradient flow," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–11.

[47] P. de Jorge, A. Sanyal, H. S. Behl, P. H. S. Torr, G. Rogez, and P. K. Dokania, "Progressive skeletonization: Trimming more fat from a network at initialization," in *Proc. 9th Int. Conf. Learn. Representations*, 2021, pp. 1–21.

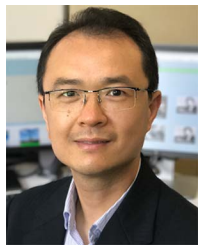
[48] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Linear mode connectivity and the lottery ticket hypothesis," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 3259–3269.

[49] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi, "Dynamic model pruning with feedback," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–22.
[50] J. Mu, M. Wang, F. Zhu, J. Yang, W. Lin, and W. Zhang, "Boosting the convergence of reinforcement learning-based auto-pruning using historical data," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 43, no. 2, pp. 548–561, Feb. 2024.
[51] L. Yao, R. Pi, H. Xu, W. Zhang, Z. Li, and T. Zhang, "Joint-detnas: Upgrade your detector with NAS, pruning and dynamic distillation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10175–10184.
[52] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.
[53] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1398–1406.
[54] X. Wang, J. Rachwan, S. Günnemann, and B. Charpentier, "Structurally prune anything: Any architecture, any framework, any time," 2024, *arXiv:2403.18955*.
[55] "Onnx-runtime environment," 2021. [Online]. Available: https://onnxruntime.ai/
[56] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, Canada, Tech. Rep., 2009.
[57] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[58] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The Pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.
[59] B. Hariharan, P. Arbeláez, L. D. Bourdev, S. Maji, and J. Malik, "Semantic contours from inverse detectors," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 991–998.
[60] L. Gao et al., "A framework for few-shot language model evaluation," *Version V0. 0.1. Sep.*, vol. 10, pp. 8–9, 2021.
[61] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "Boolq: Exploring the surprising difficulty of natural yes/no questions," 2019, *arXiv: 1905.10044*.
[62] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, "PIQA: Reasoning about physical commonsense in natural language," in *Proc. 32nd Innov. Appl. Artif. Intell. Conf.*, 2020, pp. 7432–7439.
[63] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?," 2019, *arXiv: 1905.07830*.
[64] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Commun. ACM*, vol. 64, no. 9, pp. 99–106, 2021.
[65] P. Clark et al., "Think you have solved question answering? Try arc, the AI2 reasoning challenge," 2018, *arXiv: 1803.05457*.
[66] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? A new dataset for open book question answering," 2018, *arXiv: 1809.02789*.
[67] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11264–11272.
[68] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Neural pruning via growing regularization," 2020, *arXiv: 2012.09243*.
[69] H. Wang, C. Qin, Y. Bai, and Y. Fu, "Why is the state of neural network pruning so confusing? On the fairness, comparison setup, and trainability in network pruning," 2023, *arXiv:2301.05219*.
[70] H. Wang and Y. Fu, "Trainability preserving neural structured pruning," 2022, *arXiv:2207.12534*.
[71] R. Taori et al., "Stanford alpaca: An instruction-following LLaMA model," 2023. [Online]. Available: https://github.com/tatsu-lab/stanford_alpaca

Dongdong Ren is currently working toward the PhD degree in the School of Electronic Science and Engineering, Nanjing University, China. His research interests include computer vision, deep learning, and HArdware and Software CO-design. He is working on model compression problems.

Wenbin Li received the PhD degree from the Department of Computer Science and Technology, Nanjing University, China, in 2019. He is currently an associate professor in the School of Intelligence Science and Technology with Nanjing University, Suzhou Campus. His research interests include brain-inspired artificial intelligence, advanced machine learning, computer vision, and collaborative optimization of software and hardware.

Tianyu Ding received the bachelor's degree in mathematics from Sun Yat-sen University, the two master's degrees in computer science and financial mathematics from JHU, and the PhD degree in applied mathematics and statistics from Johns Hopkins University (JHU). He is currently a principal researcher with Microsoft, Redmond, USA. His research interests focus on improving efficiency in machine learning and artificial intelligence, especially in areas like computer vision and generative models.



Lei Wang (Senior Member, IEEE) received the PhD degree from Nanyang Technological University, Singapore. He is now an associate professor with the School of Computing and Information Technology of University of Wollongong, Australia. His research interests include machine learning, pattern recognition, and computer vision. He has published more than 160 peer-reviewed papers, including those in highly regarded journals and conferences, such as the *IEEE Transactions on Pattern Analysis and Machine Intelligence*, the *International Journal of Computer Vision*, CVPR, ICCV, and ECCV, etc. He was awarded the Early Career Researcher Award by Australian Academy of Science and Australian Research Council. He served as the general cochair of DICTA 2014 and on the Technical Program Committees of more than 20 international conferences and workshops.

Qi Fan received the BEng degree from the China University of Mining and Technology, in 2015, the master's degree from Tsinghua University, in 2018, and the PhD degree from the Hong Kong University of Science and Technology. He is a tenure-track assistant professor with Nanjing University. His research interests lie in the area of computer vision and the data-efficient generalization of deep learning solutions, especially for vision foundation models.

Jing Huo received the PhD degree from the Department of Computer Science and Technology, Nanjing University, Nanjing, China, in 2017. She is currently an associate professor with the Department of Computer Science and Technology, Nanjing University. Her current research interests include machine learning and computer vision, with a focus on subspace learning, adversarial learning and their applications to heterogeneous face recognition and cross-modal face generation.

Hongbing Pan received the BS degree in applied physics from Nanjing University, China, in 1994 and the PhD degree in microelectronics and solid state electronics from Nanjing University, Nanjing, China, in 2005. From 2006 to 2012, he was an associate professor of the Institute of VLSI Design, Nanjing University, Nanjing, China. Since 2013, he has been a professor of the School of Electronic Science and Engineering, Nanjing University. He is the author of more than 40 articles. His research interests include VLSI Design, CMOS Sensors, Reconfigurable computing, and Artificial intelligence.

Yang Gao received the PhD degree from the Department of Computer Science and Technology, Nanjing University, China, in 2000. He is currently a professor and also the deputy director of the Department of Computer Science and Technology, Nanjing University, where he is also directing the Reasoning and Learning Research Group. He has published more than 100 papers in top-tier conferences and journals. His current research interests include artificial intelligence and machine learning. He also serves as the program chair and area chair for many international conferences.