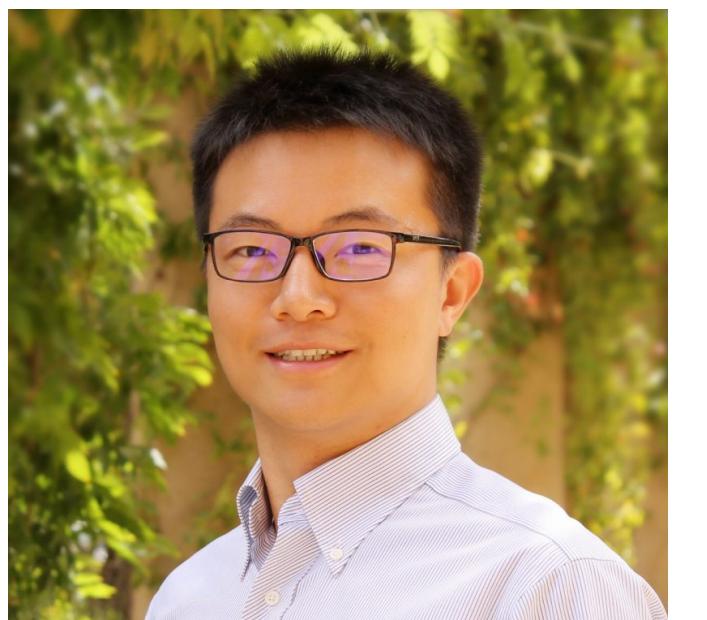


EfficientML.ai Lecture 04: Pruning and Sparsity

Part II



Song Han

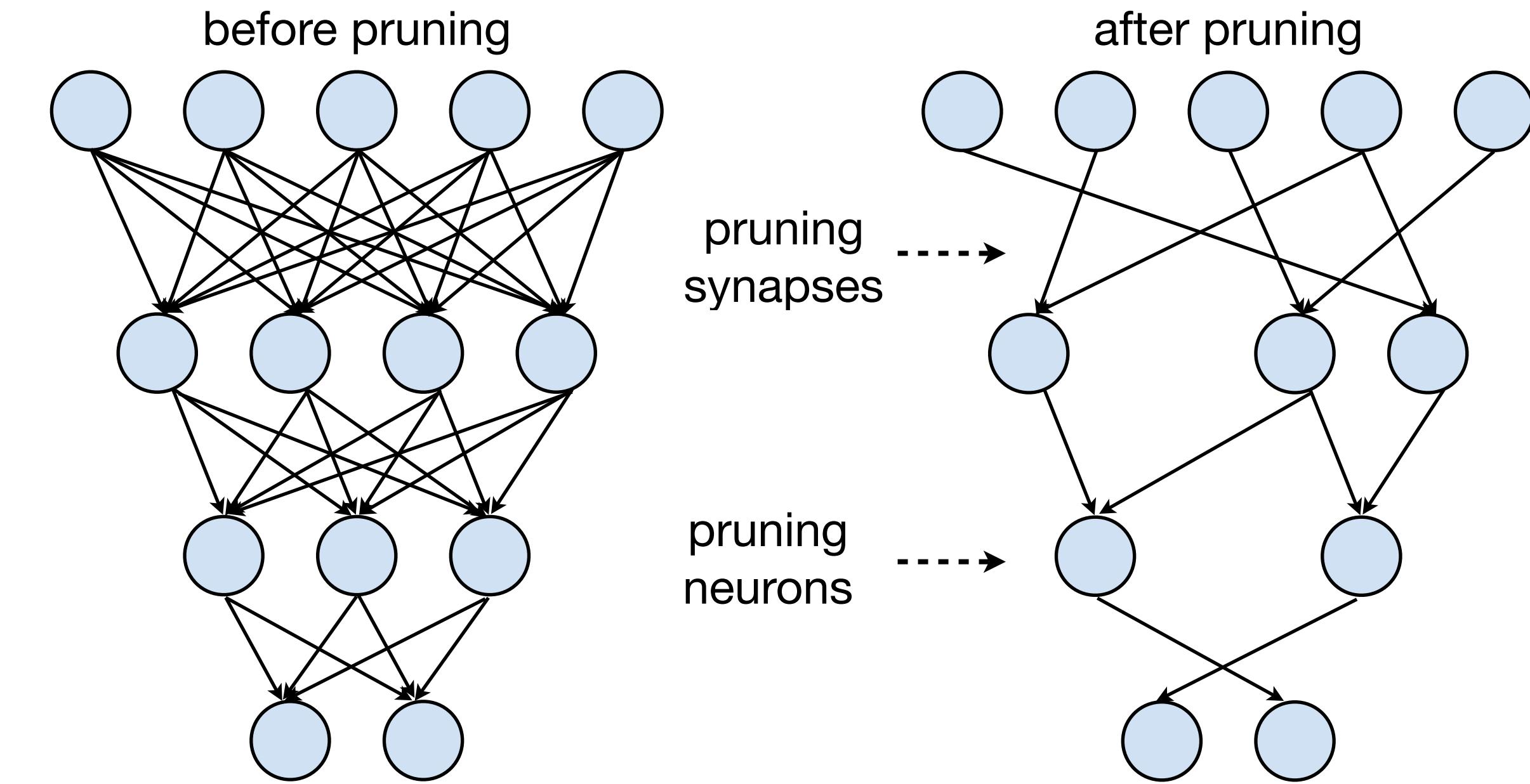
Associate Professor, MIT
Distinguished Scientist, NVIDIA

 @SongHan/MIT



Neural Network Pruning

- **Introduction to Pruning**
 - What is pruning?
 - How should we formulate pruning?
- **Determine the Pruning Granularity**
 - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
 - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
 - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
 - How should we improve performance of pruned models?



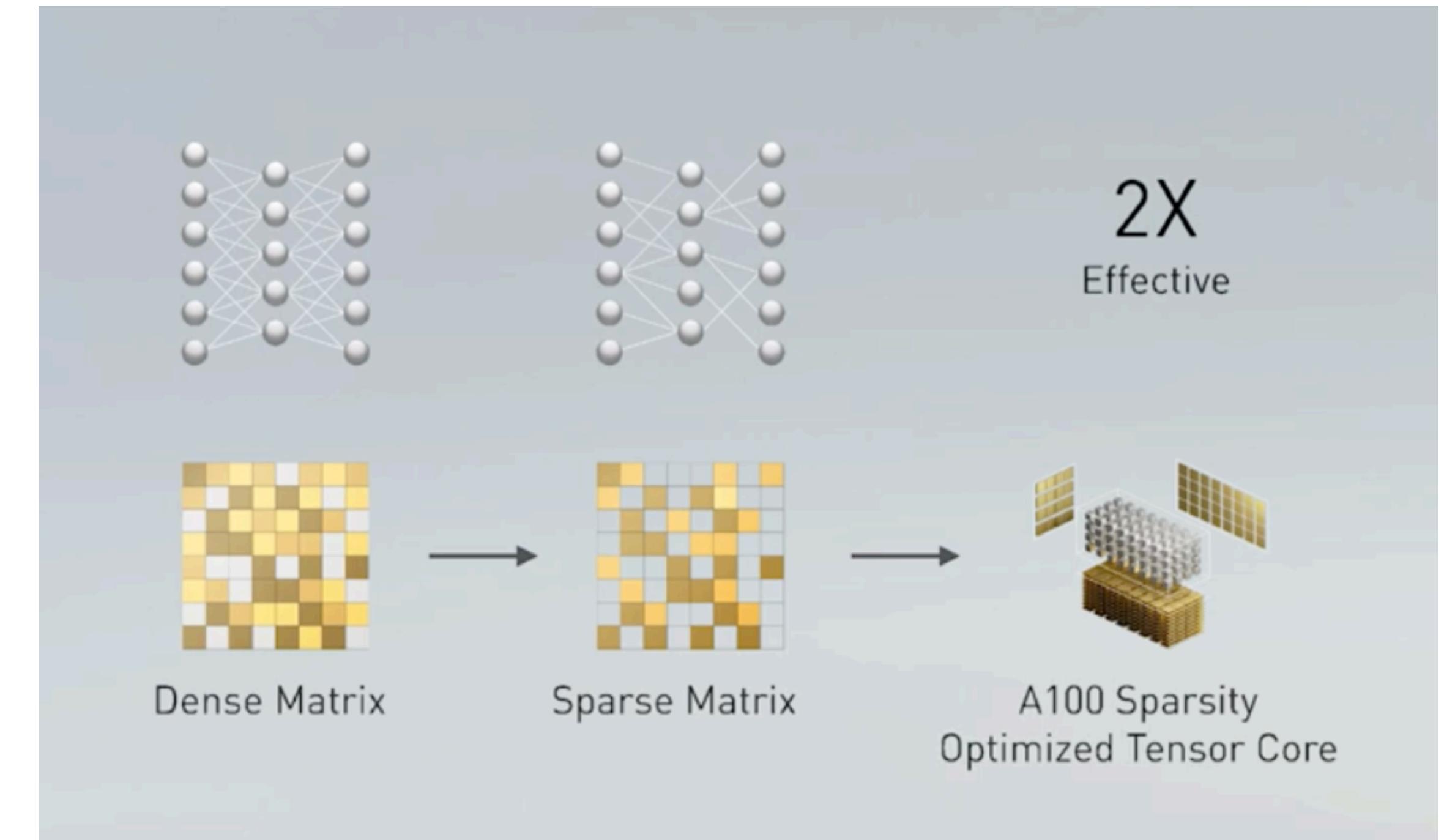
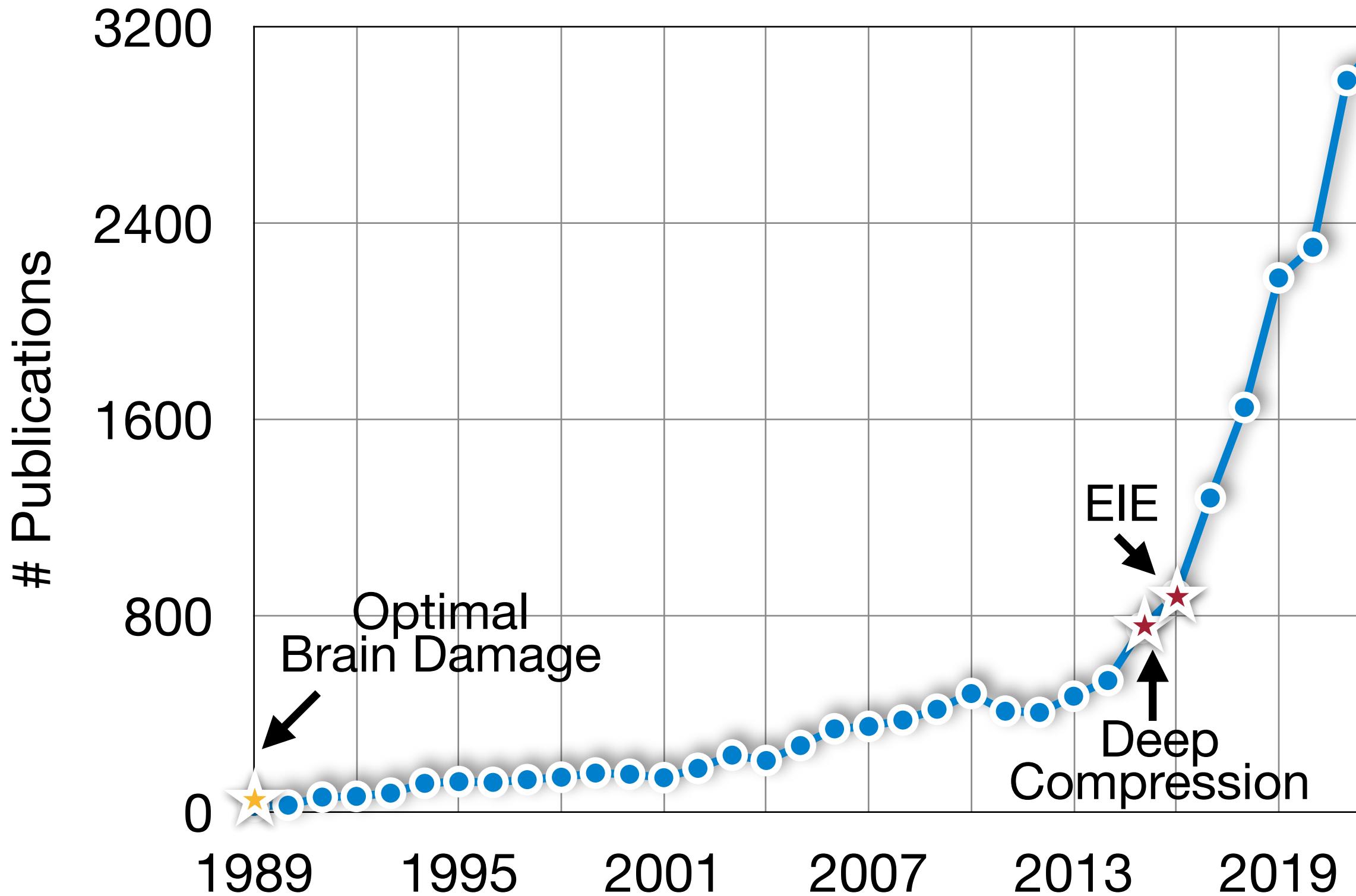
Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Neural Network Pruning



Make neural network smaller by removing synapses and neurons

- #publications on pruning and sparse neural networks



2:4 sparsity in A100 GPU

2X peak performance, 1.5X measured BERT speedup

Source: <https://github.com/mit-han-lab/pruning-sparsity-publications>

Neural Network Pruning

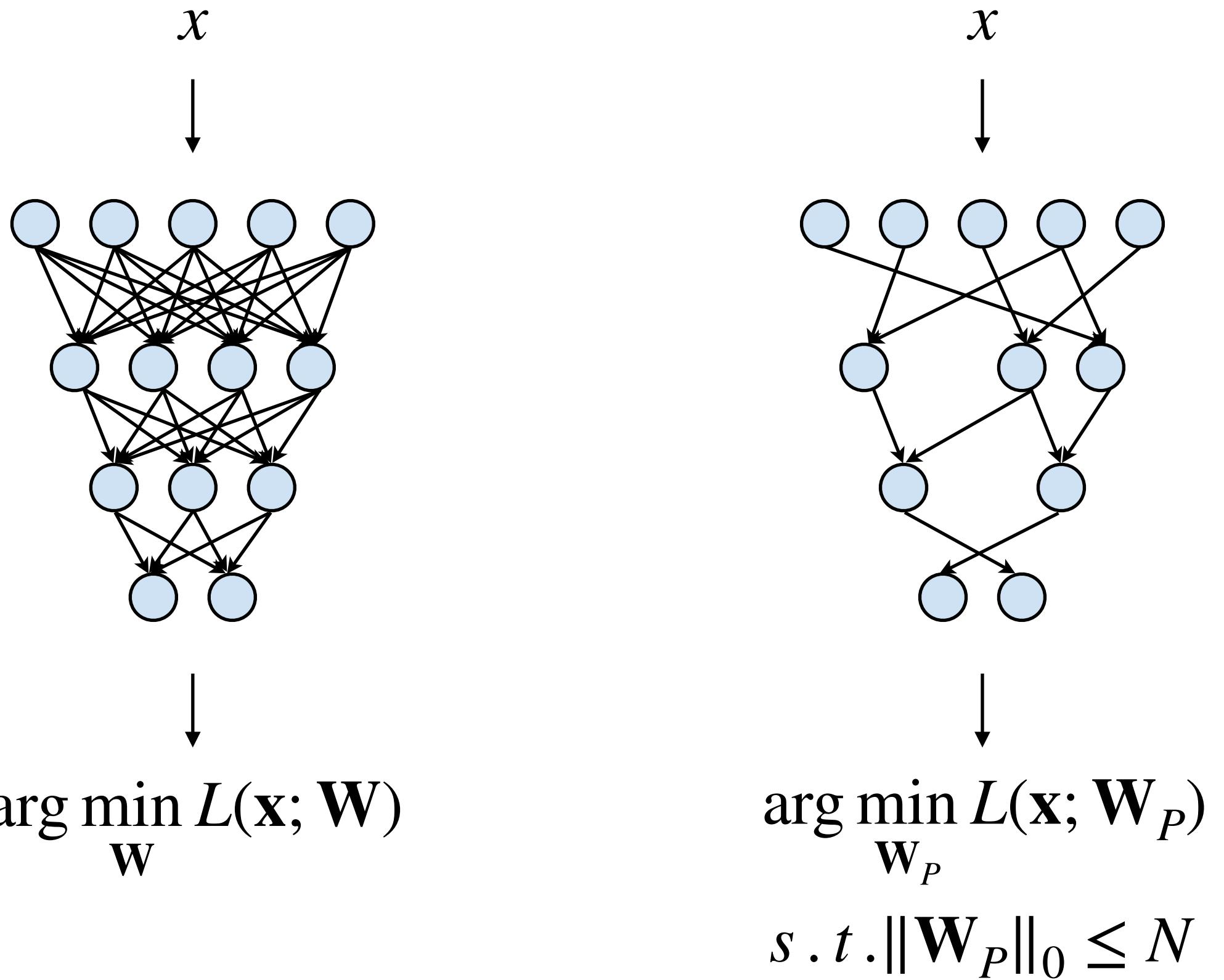
- In general, we could formulate the pruning as follows:

$$\arg \min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

subject to

$$\|\mathbf{W}_p\|_0 < N$$

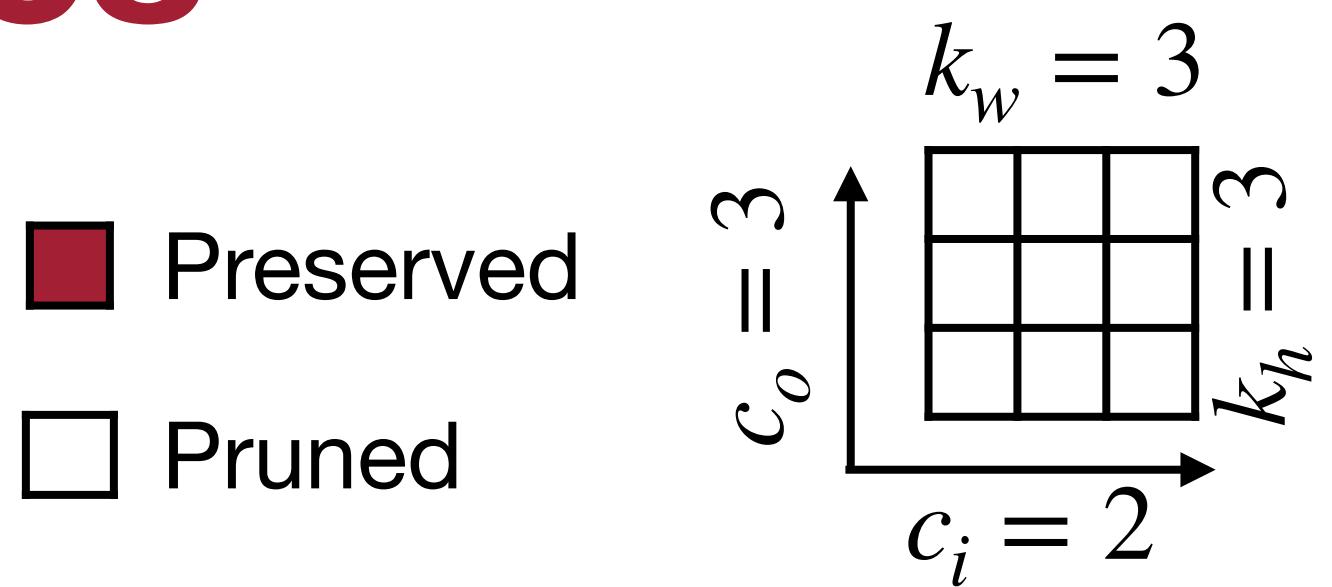
- L represents the objective function for neural network training;
- \mathbf{x} is input, \mathbf{W} is original weights, \mathbf{W}_P is pruned weights;
- $\|\mathbf{W}_p\|_0$ calculates the #nonzeros in W_P , and N is the target #nonzeros.



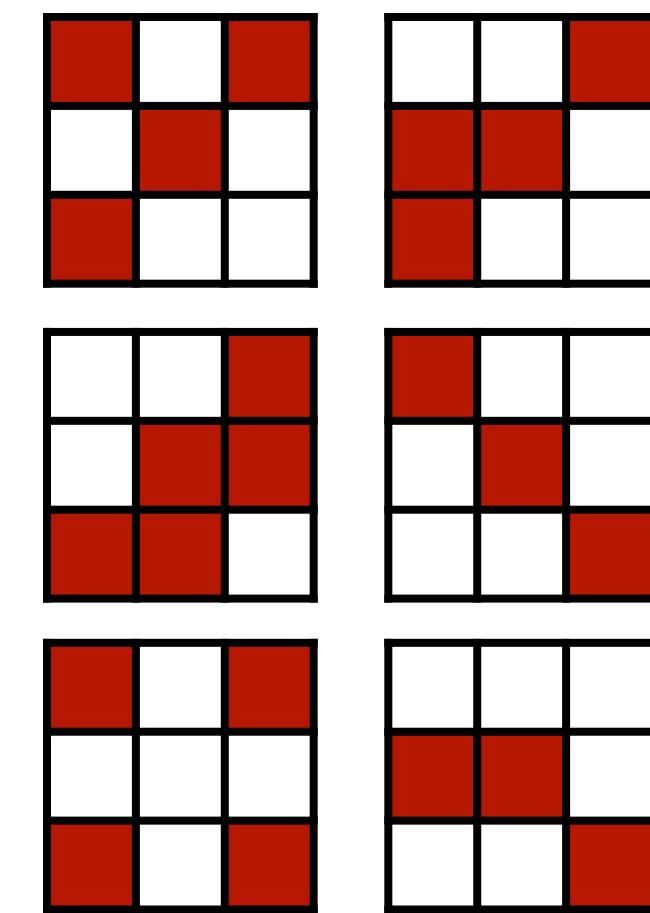
Pruning at Different Granularities

The case of convolutional layers

- Some of the commonly used pruning granularities

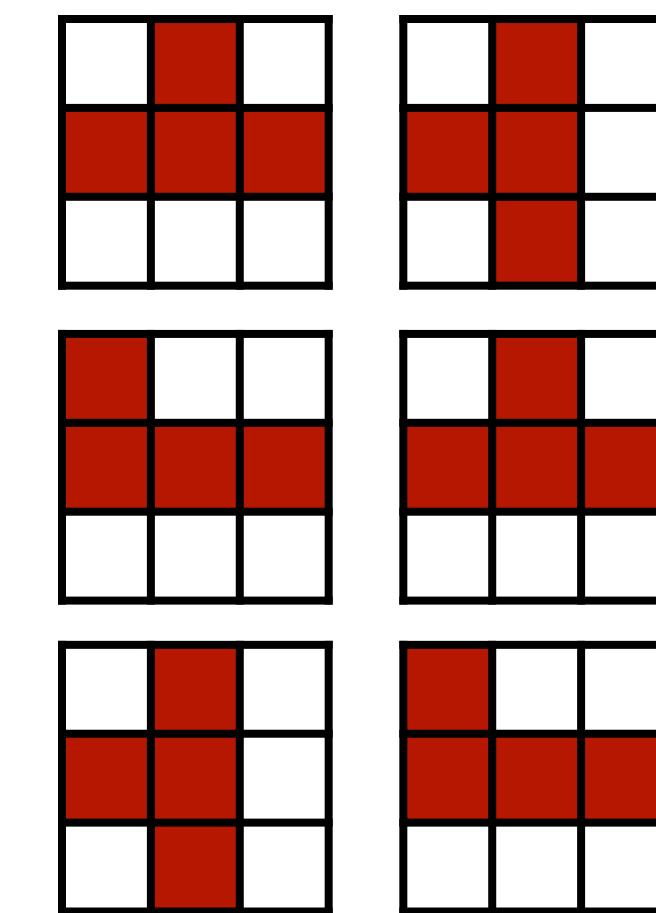


Irregular



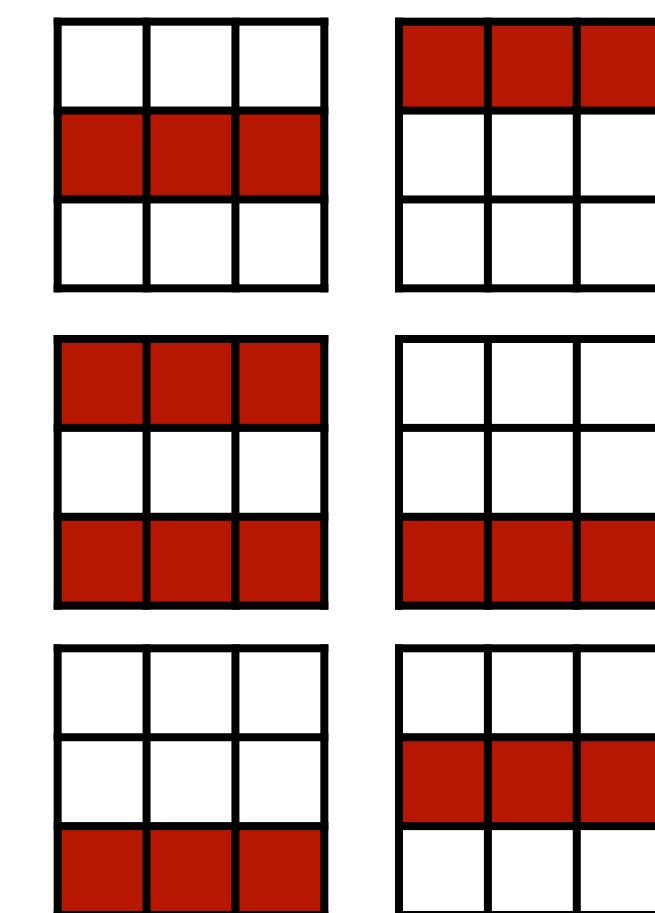
Fine-grained
Pruning

Irregular



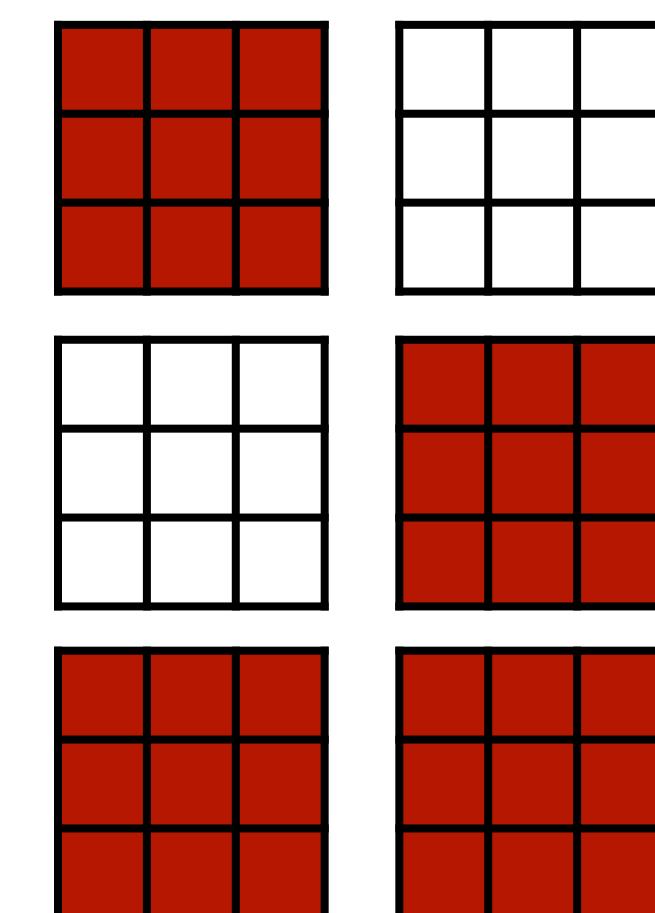
Pattern-based
Pruning

Irregular



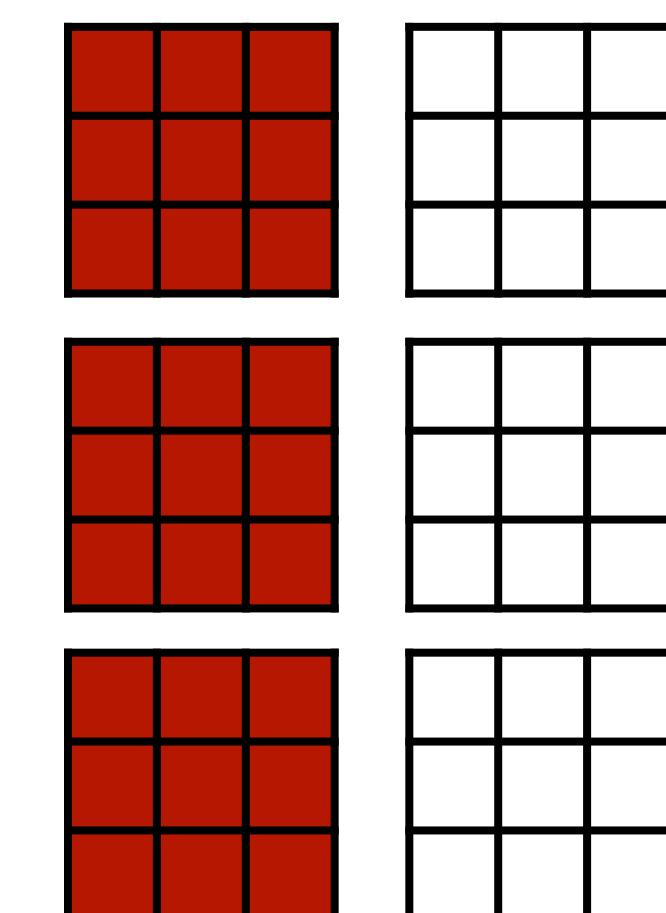
Vector-level
Pruning

Irregular



Kernel-level
Pruning

Regular



Channel-level
Pruning

like Tetris :)

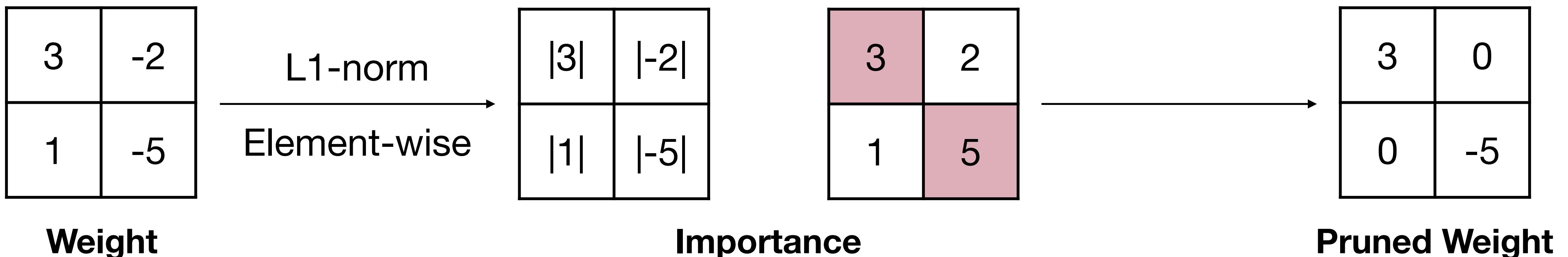
Exploring the granularity of sparsity in convolutional neural networks [Mao et al., CVPR-W]

Selection of Synapses to Prune

- When removing parameters from a neural network model,
 - ***the less important*** the parameters being removed are,
 - the better the performance of pruned neural network is.
- **Magnitude-based pruning** considers weights with ***larger absolute values*** are more important than other weights.
 - For element-wise pruning,

$$Importance = |W|$$

- **Example**

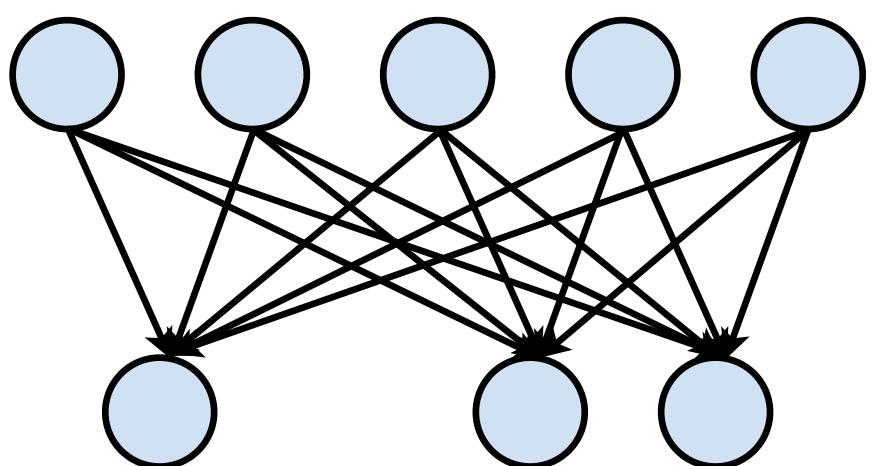


Selection of Neurons to Prune

- When removing neurons from a neural network model,
 - ***the less useful*** the neurons being removed are,
 - the better the performance of pruned neural network is.

Recall: Neuron pruning is coarse-grained pruning indeed.

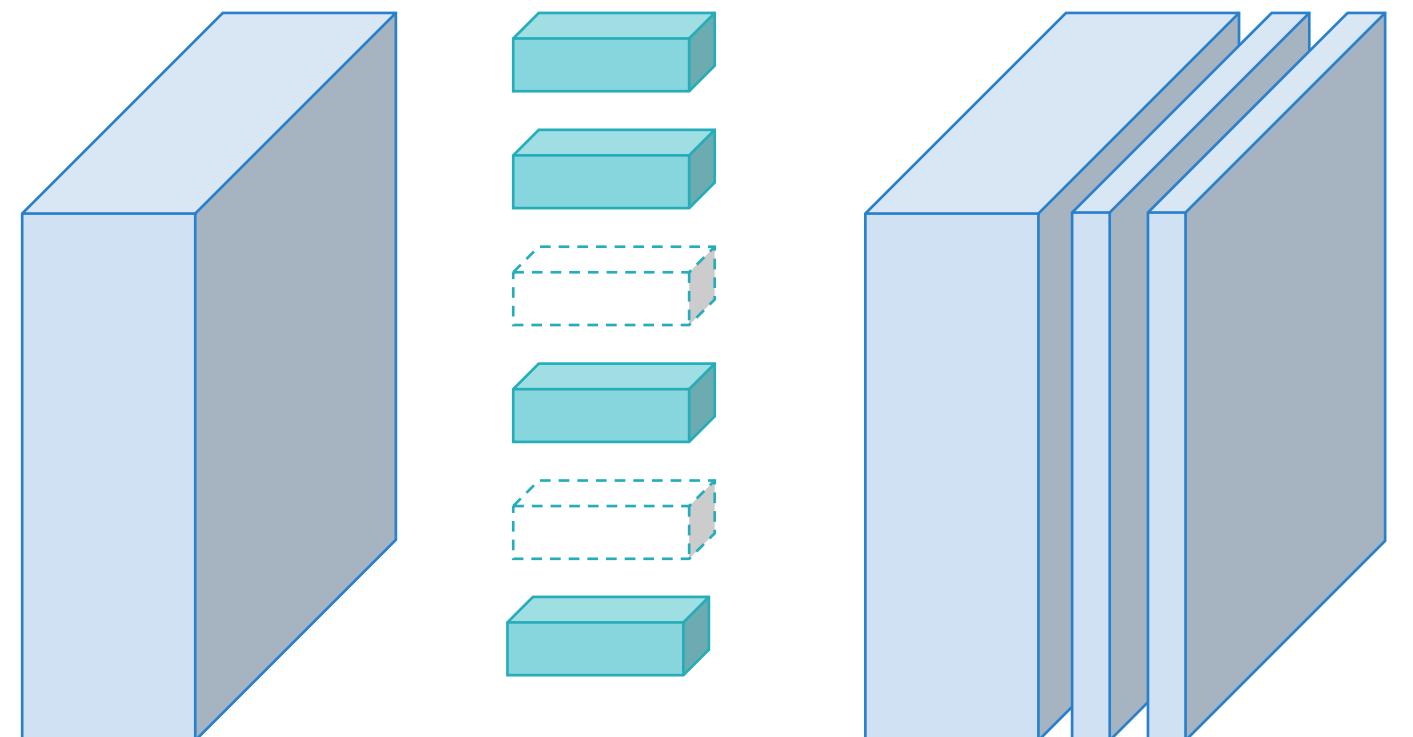
**Neuron Pruning
in Linear Layer**



Weight Matrix

■	■	■	■
■	■	■	■
■	■	■	■
■	■	■	■

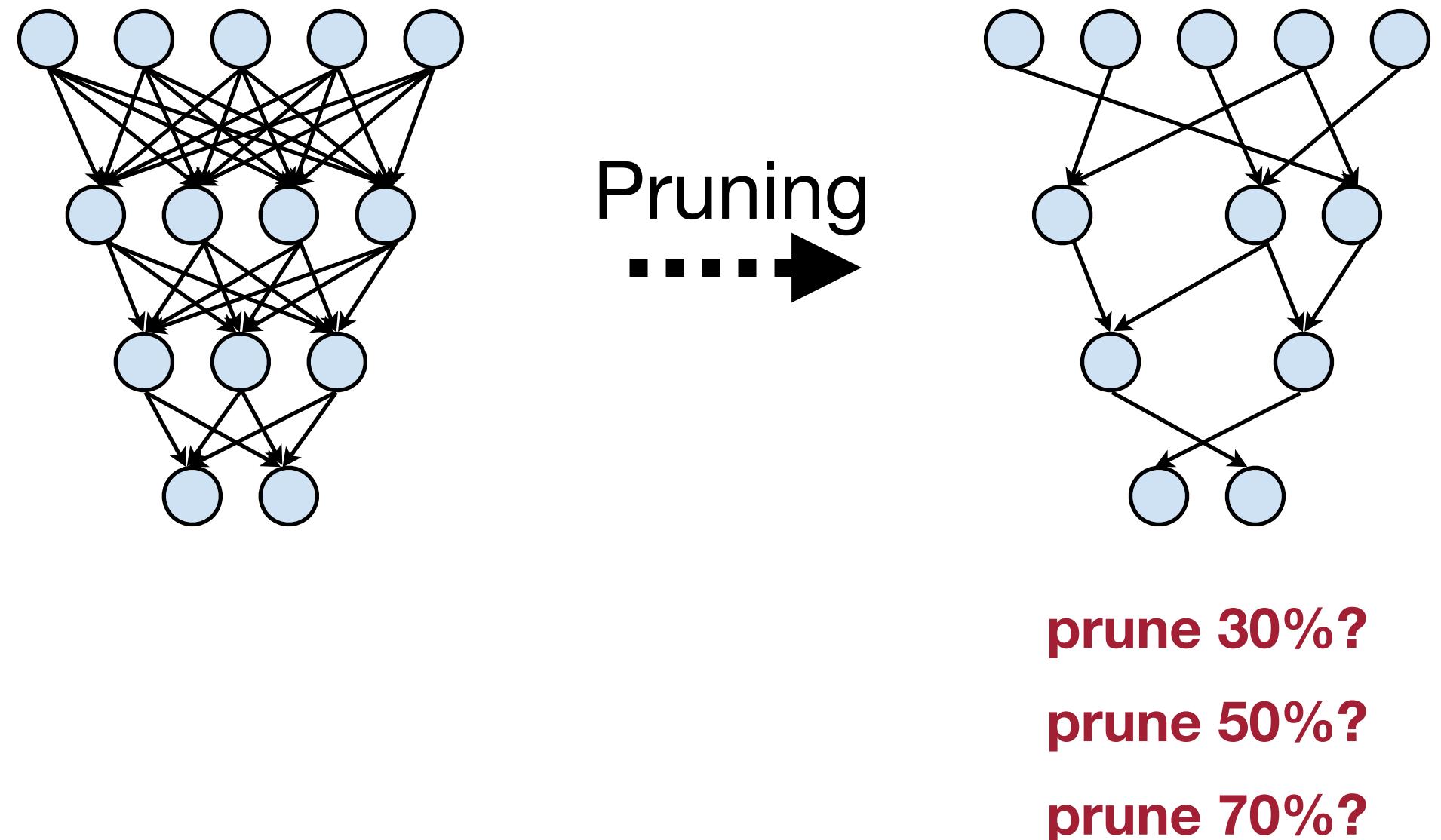
**Channel Pruning
in Convolution Layer**



■ ■ ■	■ ■ ■	■ ■ ■
■ ■ ■	■ ■ ■	■ ■ ■
■ ■ ■	■ ■ ■	■ ■ ■
■ ■ ■	■ ■ ■	■ ■ ■
■ ■ ■	■ ■ ■	■ ■ ■

Neural Network Pruning

- **Introduction to Pruning**
 - What is pruning?
 - How should we formulate pruning?
- **Determine the Pruning Granularity**
 - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
 - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
 - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
 - How should we improve performance of pruned models?



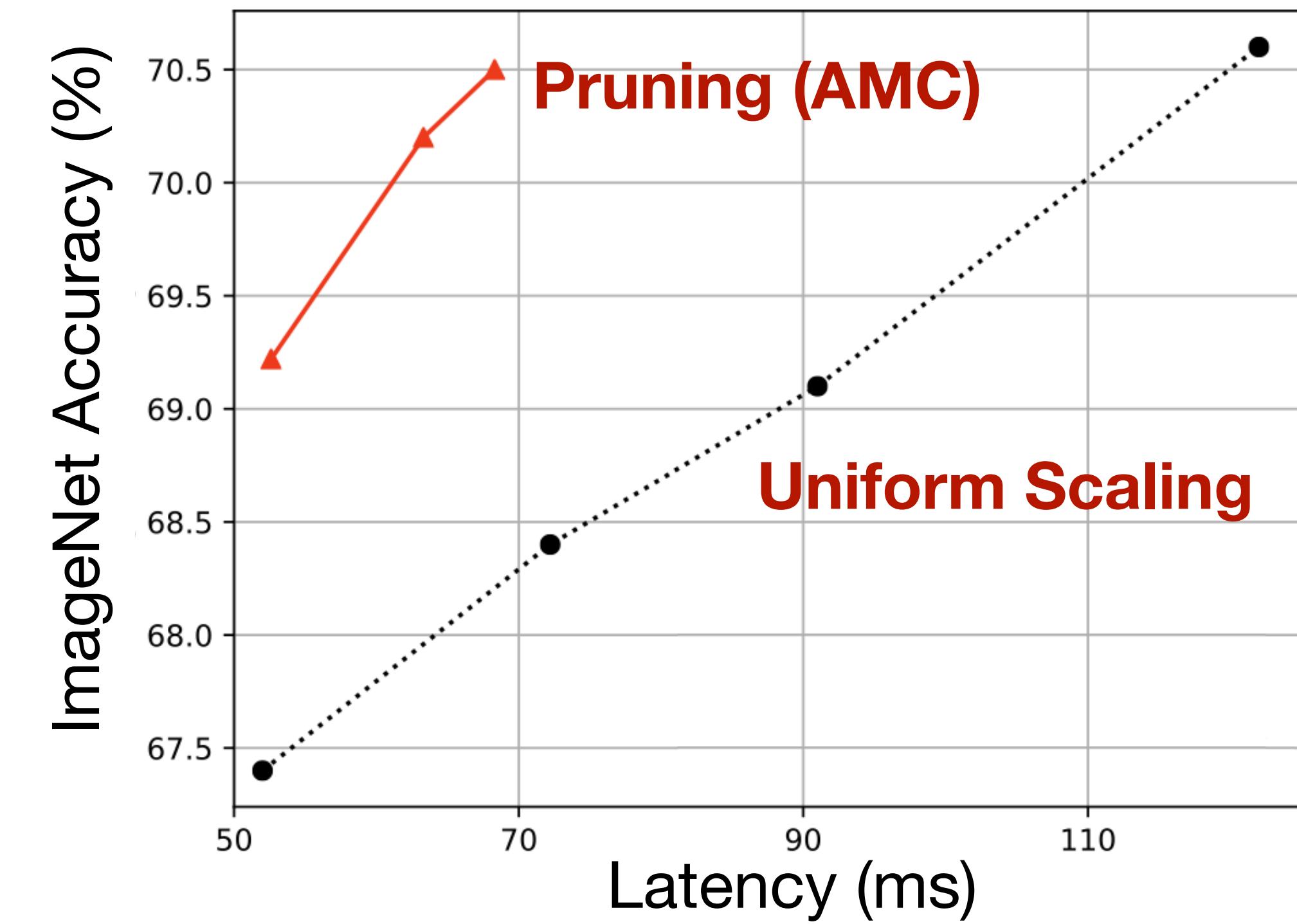
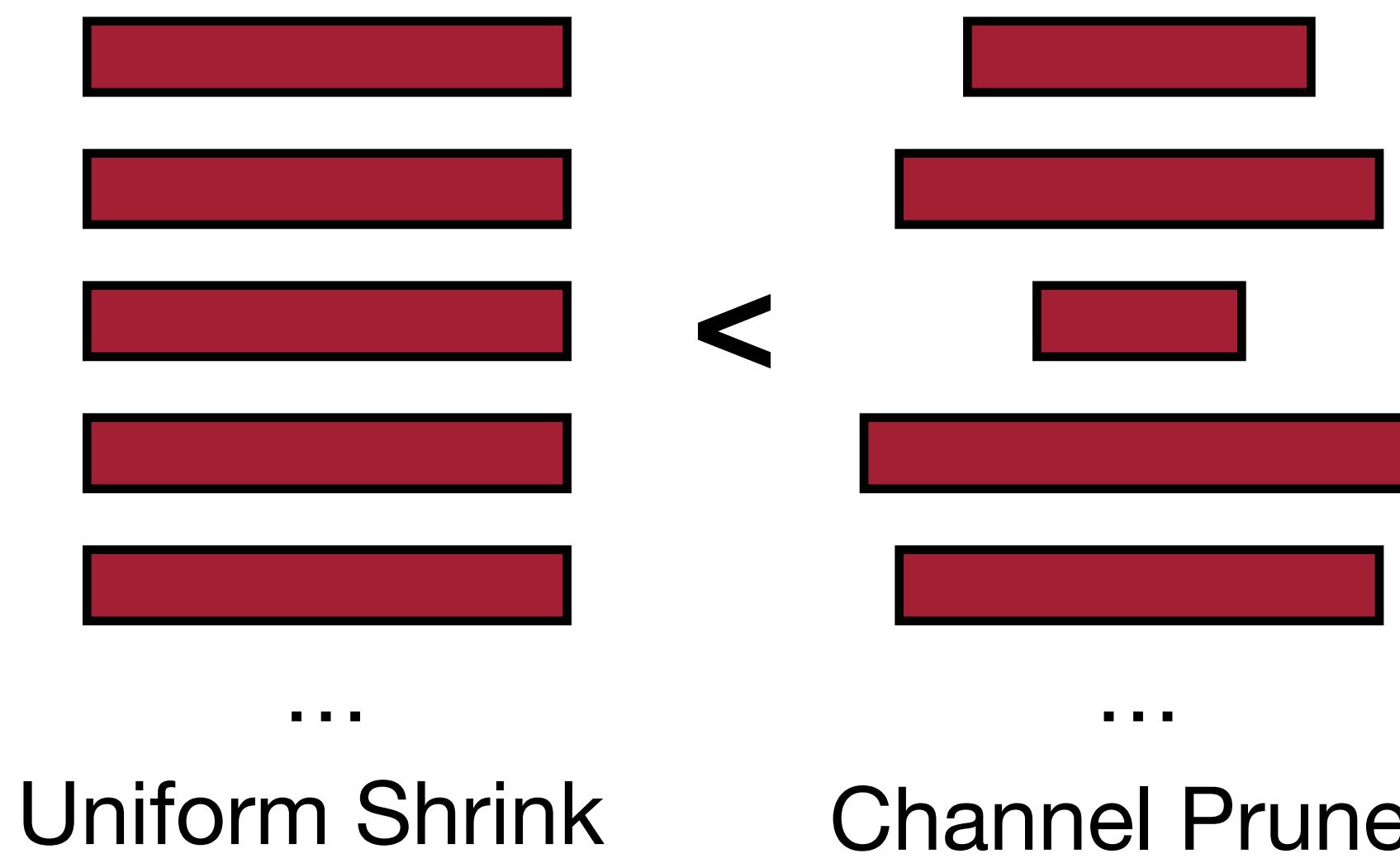
Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Section 1: Pruning Ratio

How should we find per-layer pruning ratios?

Recap

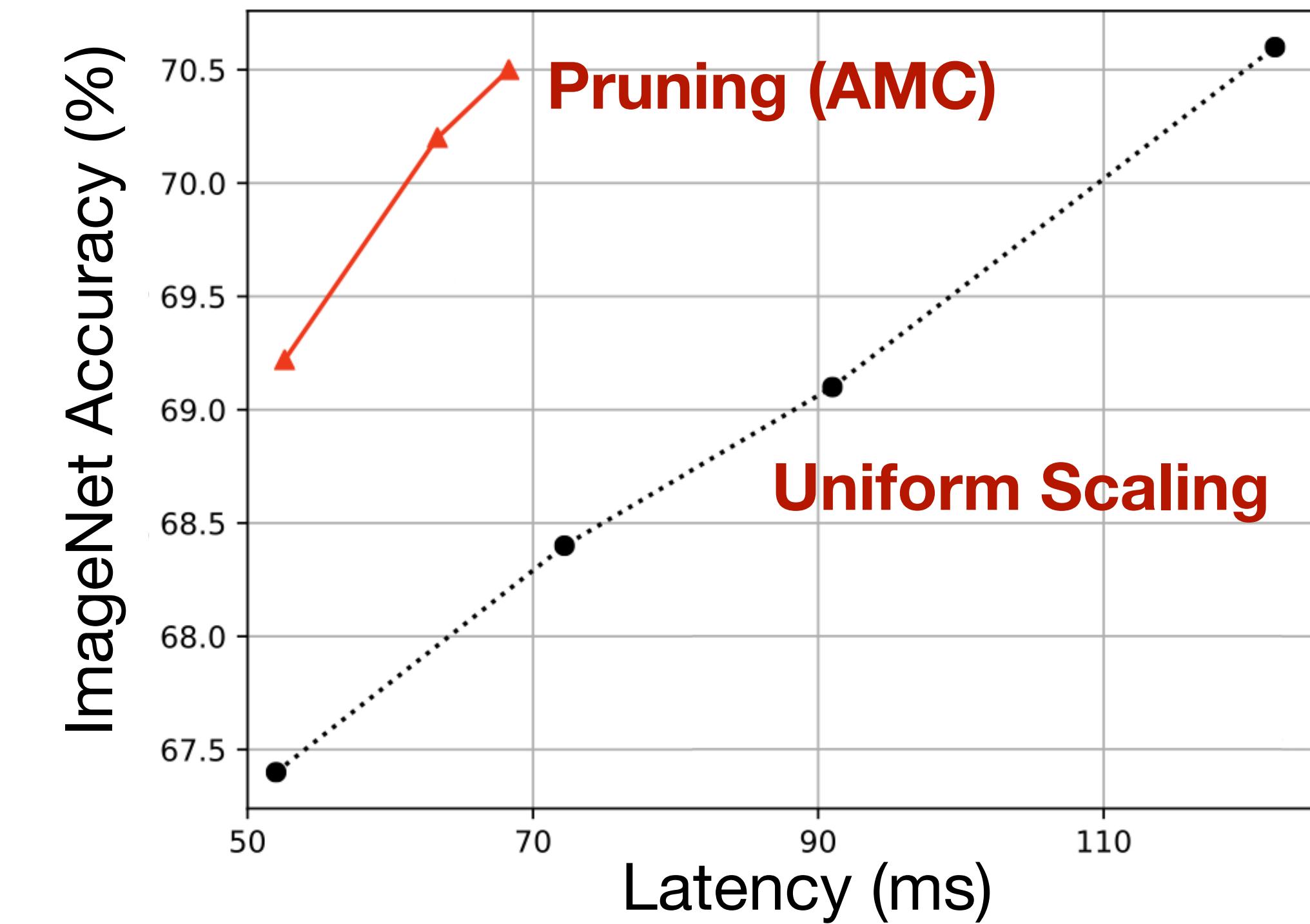
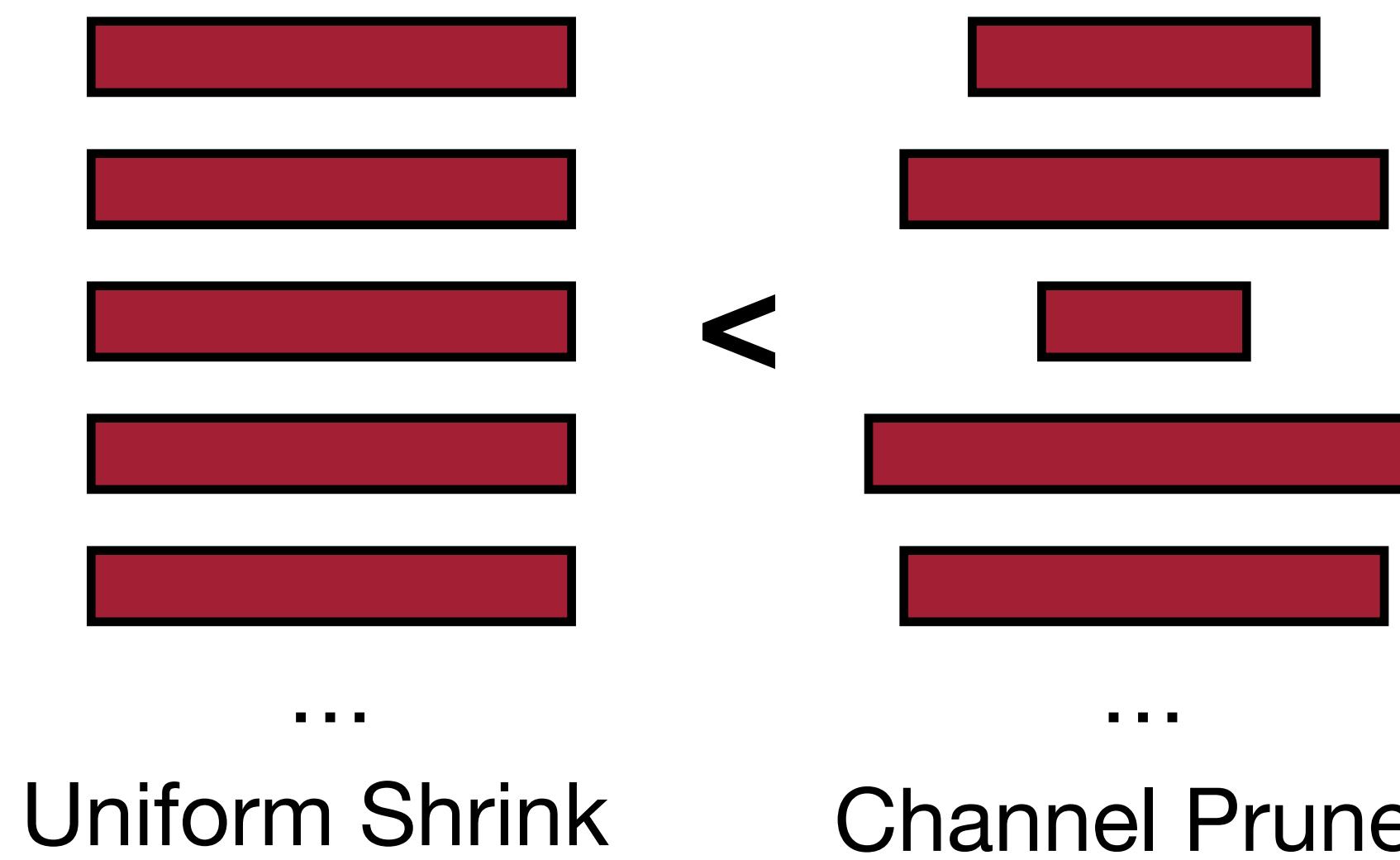
Non-uniform pruning is better than uniform shrinking



AMC: Automl for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]

Recap

Non-uniform pruning is better than uniform shrinking



Question: how should we find ratios for each layer?

AMC: Automl for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]

Finding Pruning Ratios

Analyze the sensitivity of each layer

- We need different pruning ratios for each layer since different layers have different **sensitivity**
 - Some layers are more sensitive (e.g., first layer)
 - Some layers are more redundant

Finding Pruning Ratios

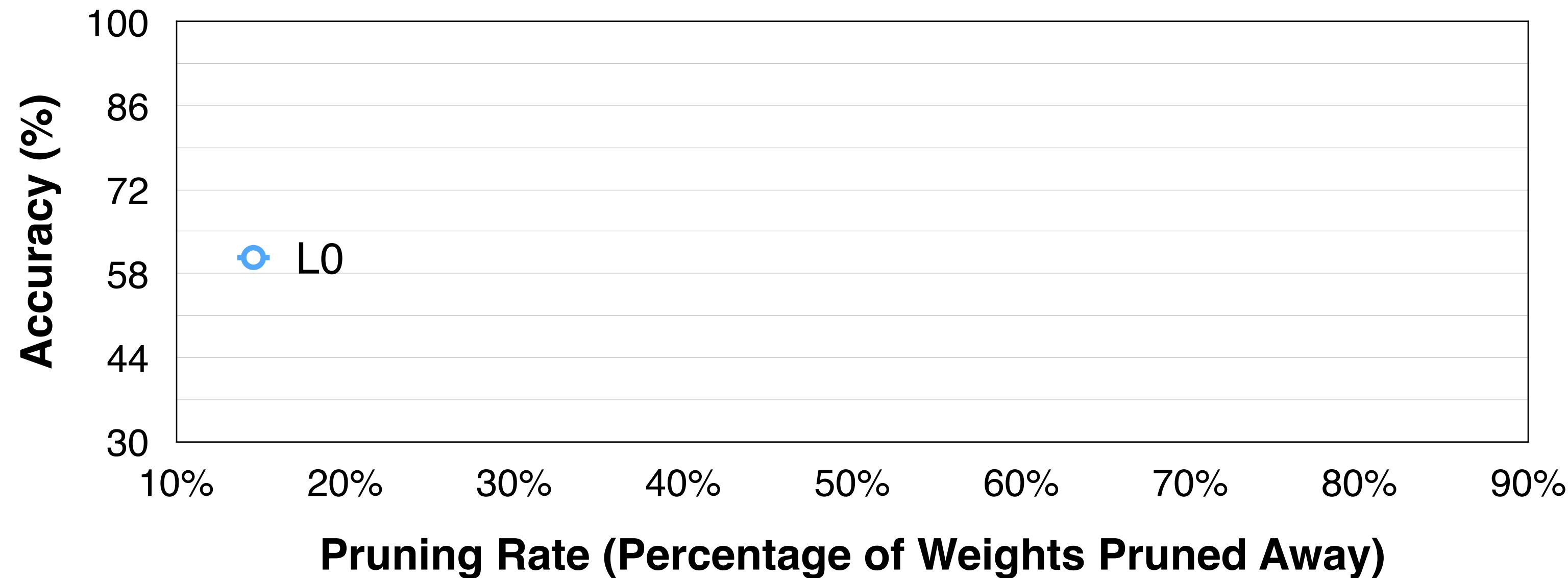
Analyze the sensitivity of each layer

- We need different pruning ratios for each layer since different layers have different **sensitivity**
 - Some layers are more sensitive (e.g., first layer)
 - Some layers are more redundant
- We can perform **sensitivity analysis** to determine the per-layer pruning ratio

Finding Pruning Ratios

Analyze the sensitivity of each layer

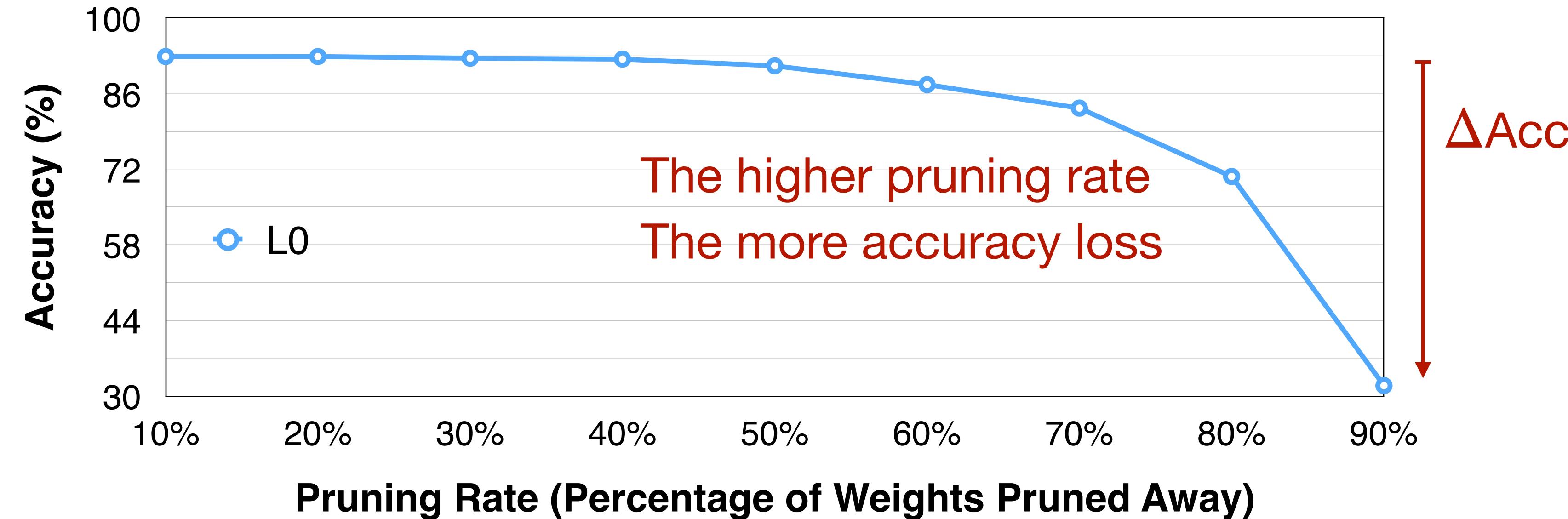
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model



Finding Pruning Ratios

Analyze the sensitivity of each layer

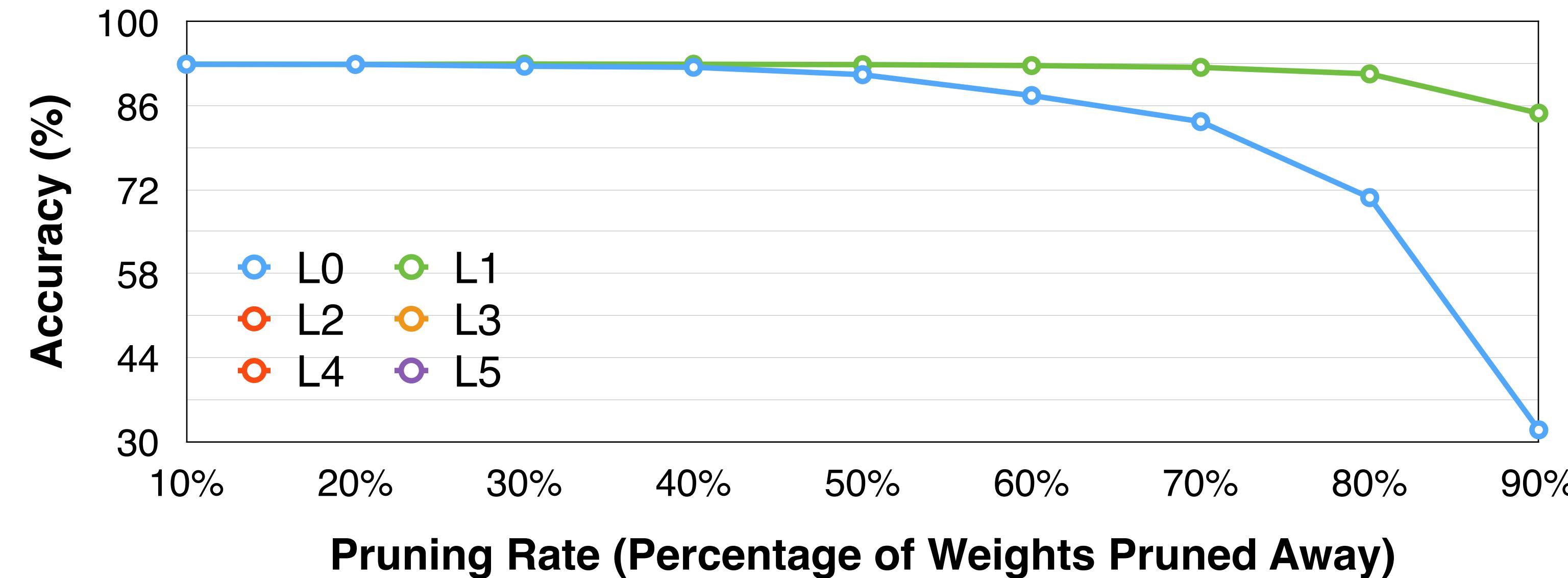
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio



Finding Pruning Ratios

Analyze the sensitivity of each layer

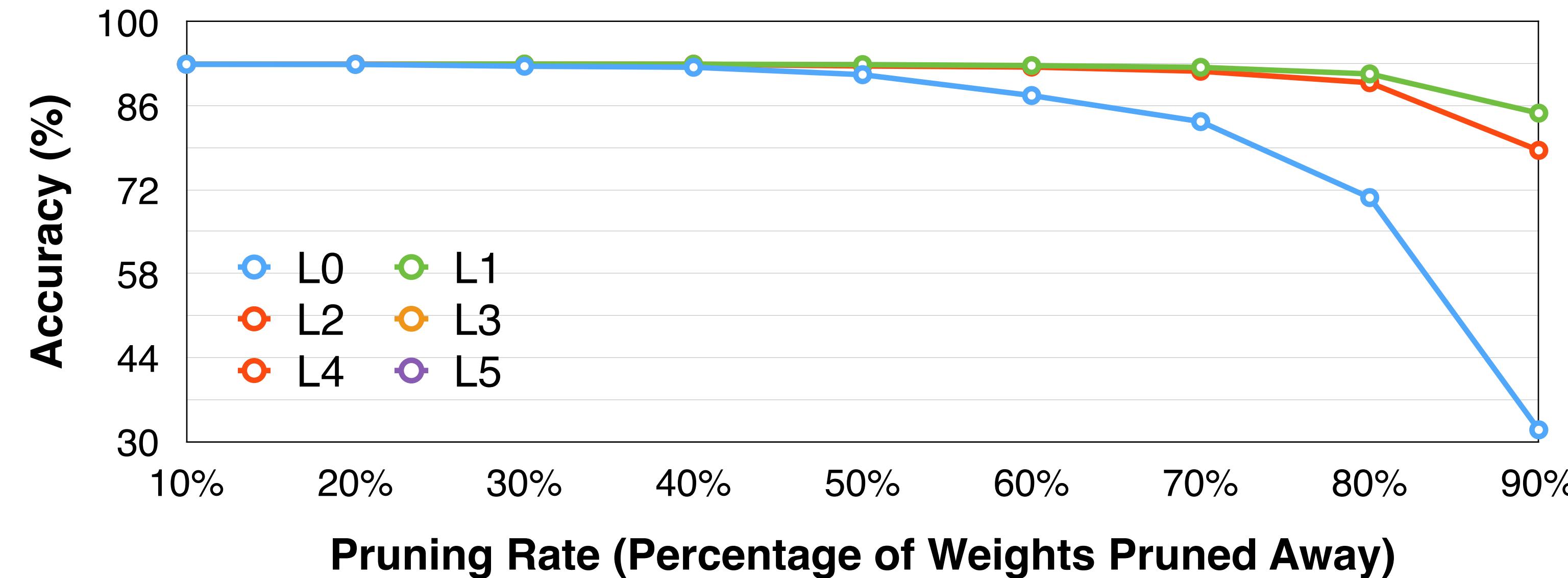
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

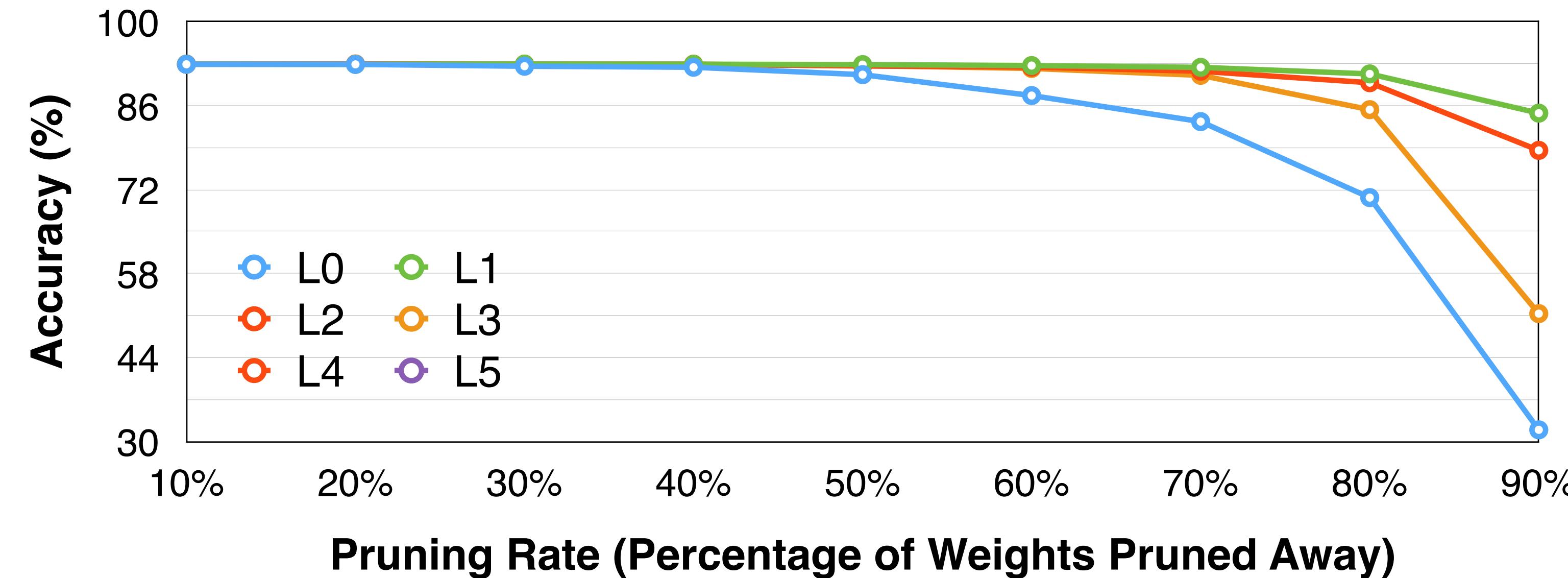
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

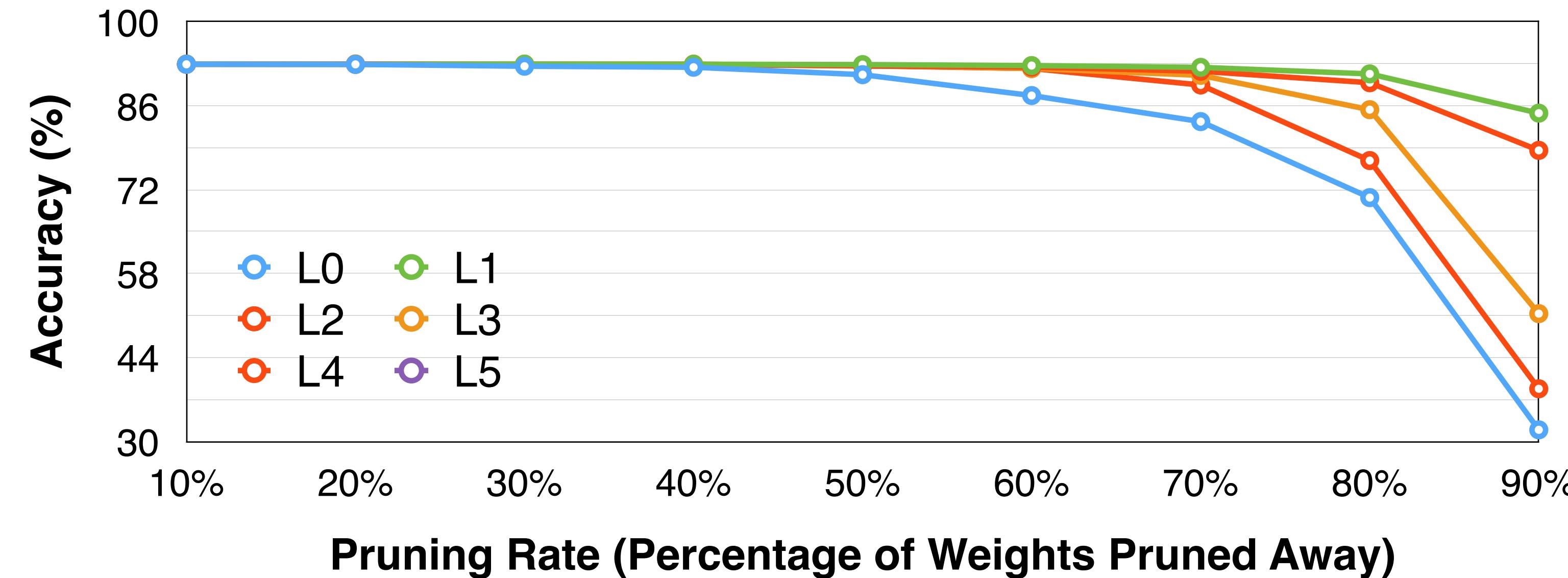
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

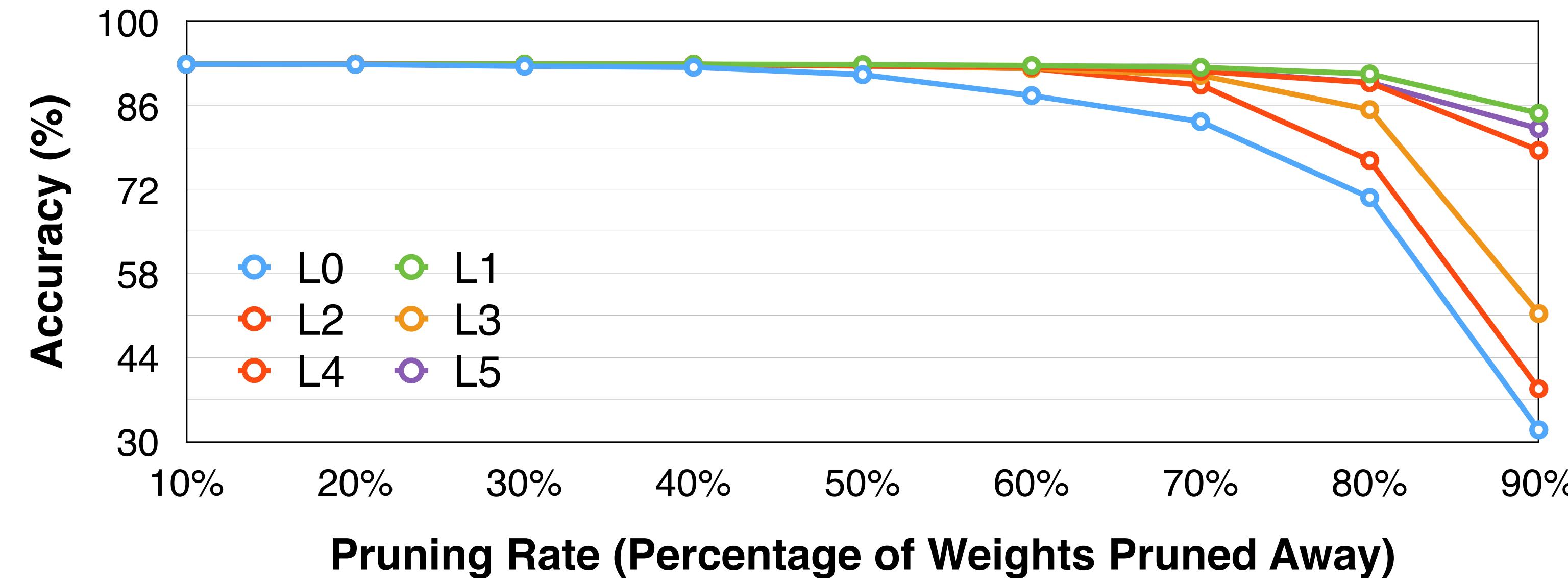
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

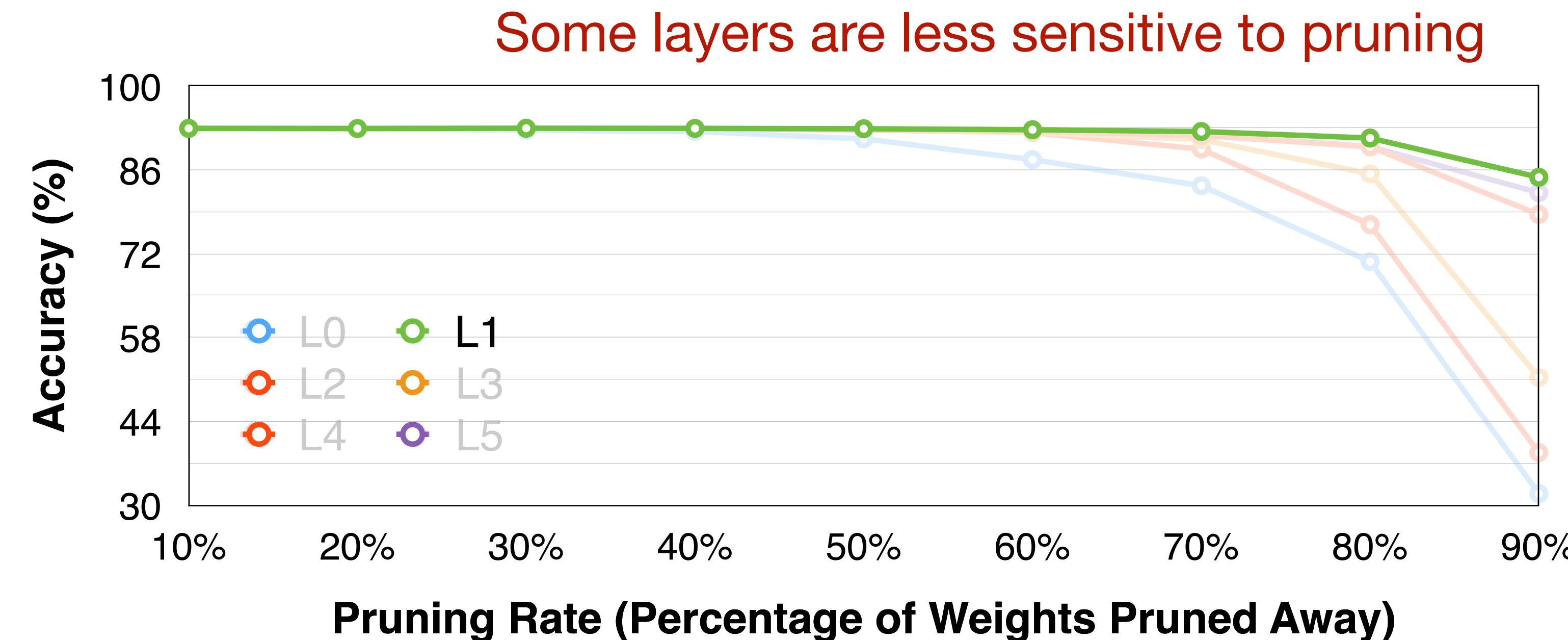
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

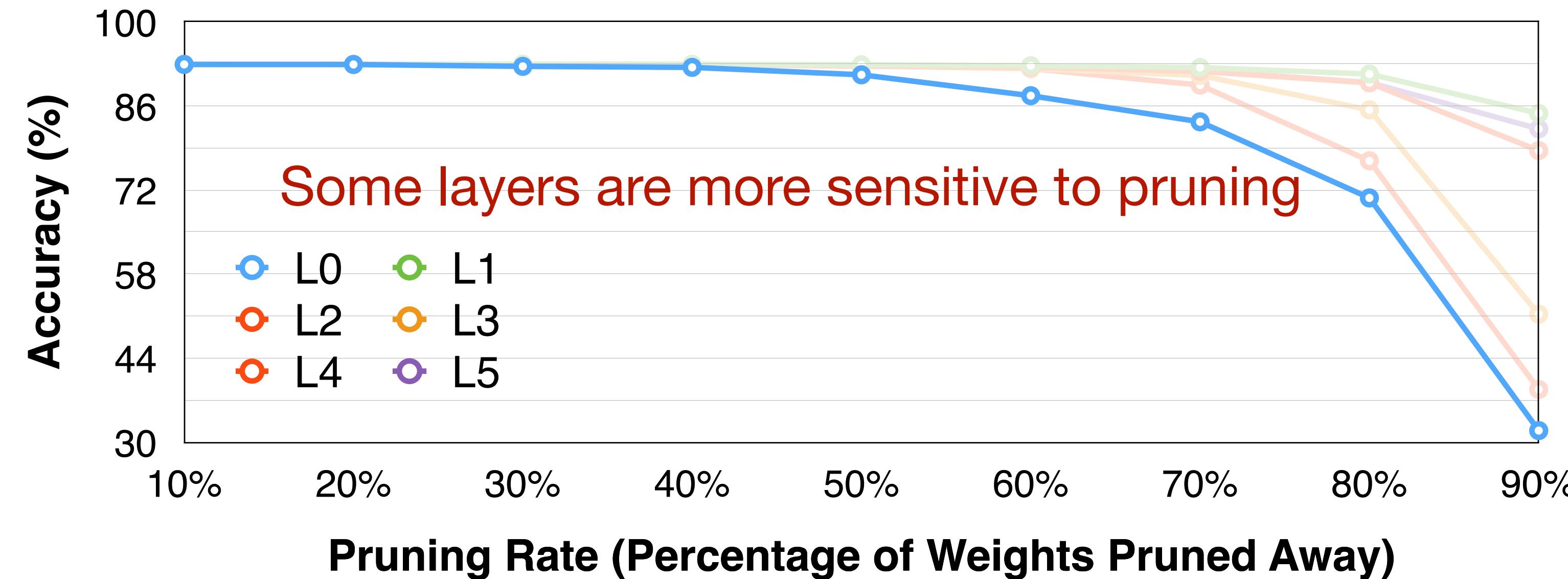
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

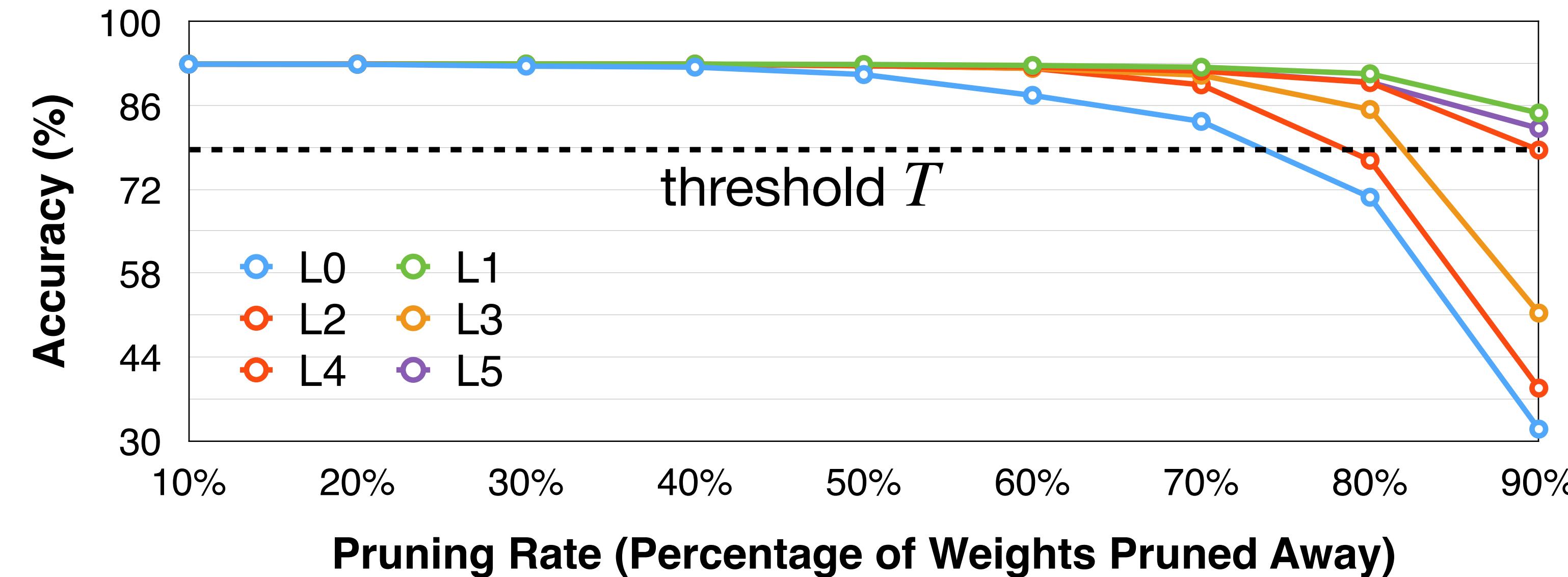
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

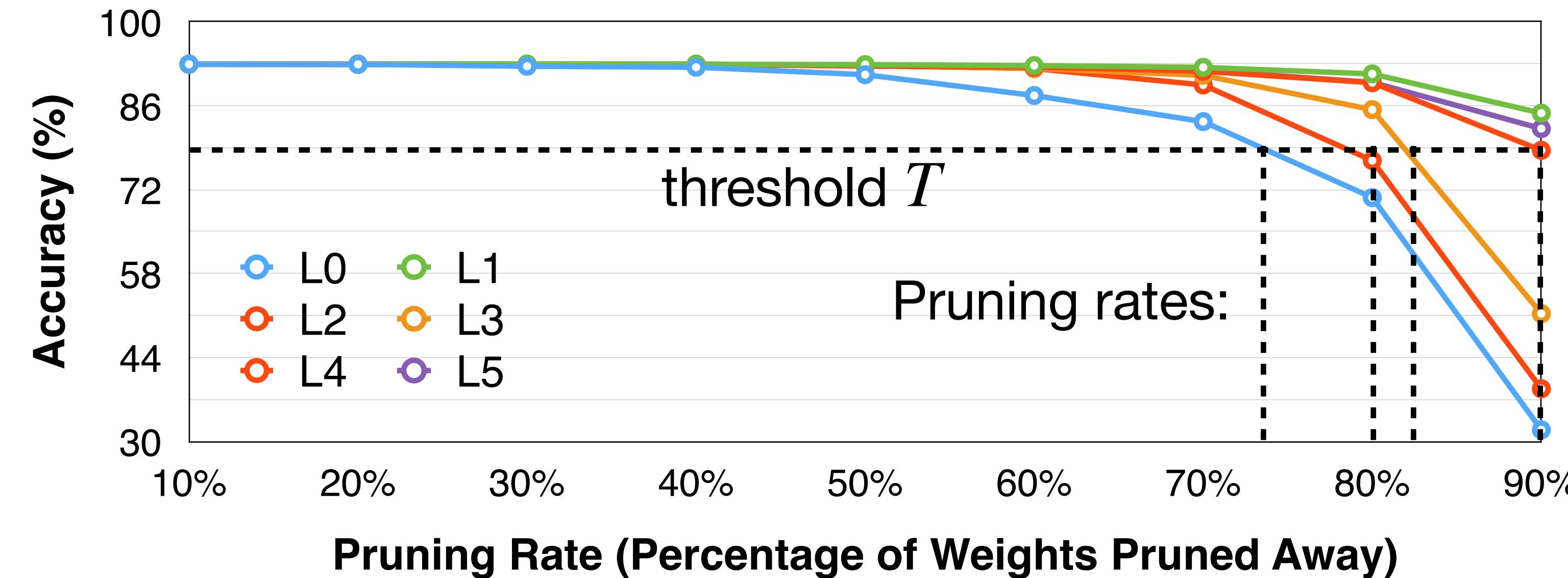
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers
- Pick a degradation threshold T such that the overall pruning rate is desired



Finding Pruning Ratios

Analyze the sensitivity of each layer

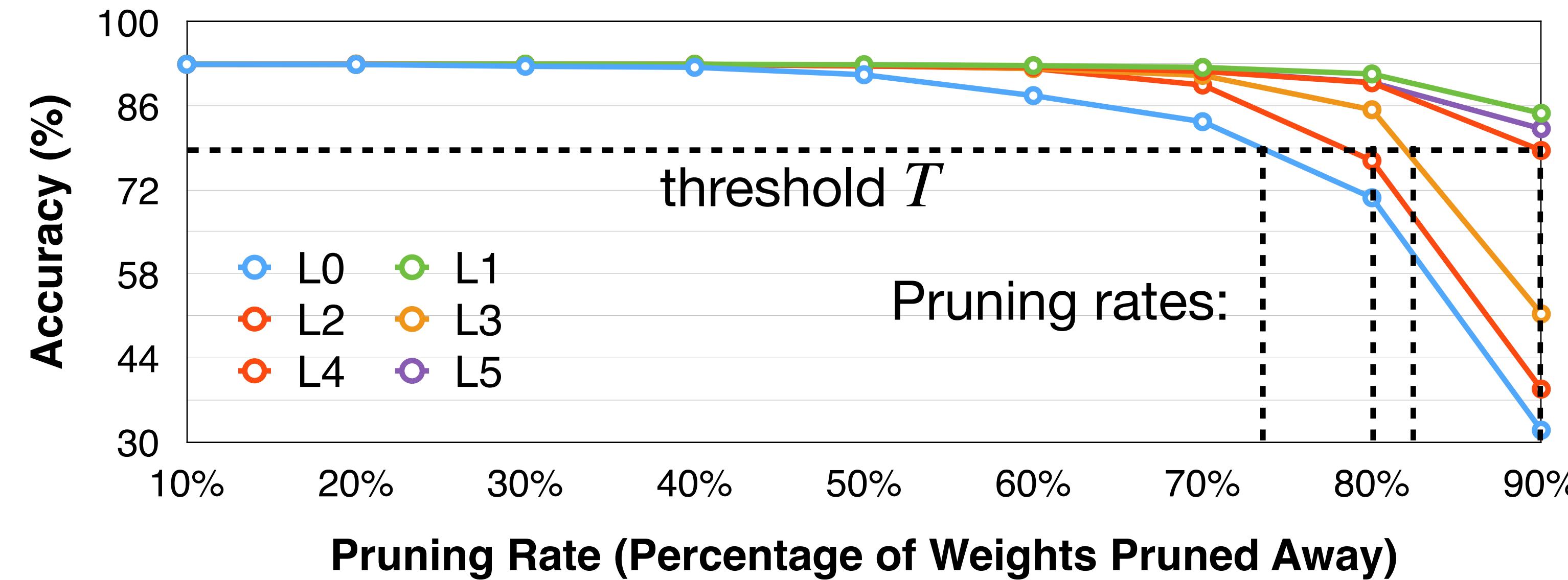
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
 - Pick a layer L_i in the model
 - Prune the layer L_i with pruning ratio $r \in \{0,0.1,0.2,\dots,0.9\}$ (or other strides)
 - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
- Repeat the process for all layers
- Pick a degradation threshold T such that the overall pruning rate is desired



Finding Pruning Ratios

Analyze the sensitivity of each layer

- Is this optimal?
 - Maybe not. We do not consider the interaction between layers.
- Can we go beyond the heuristics?
 - Yes!

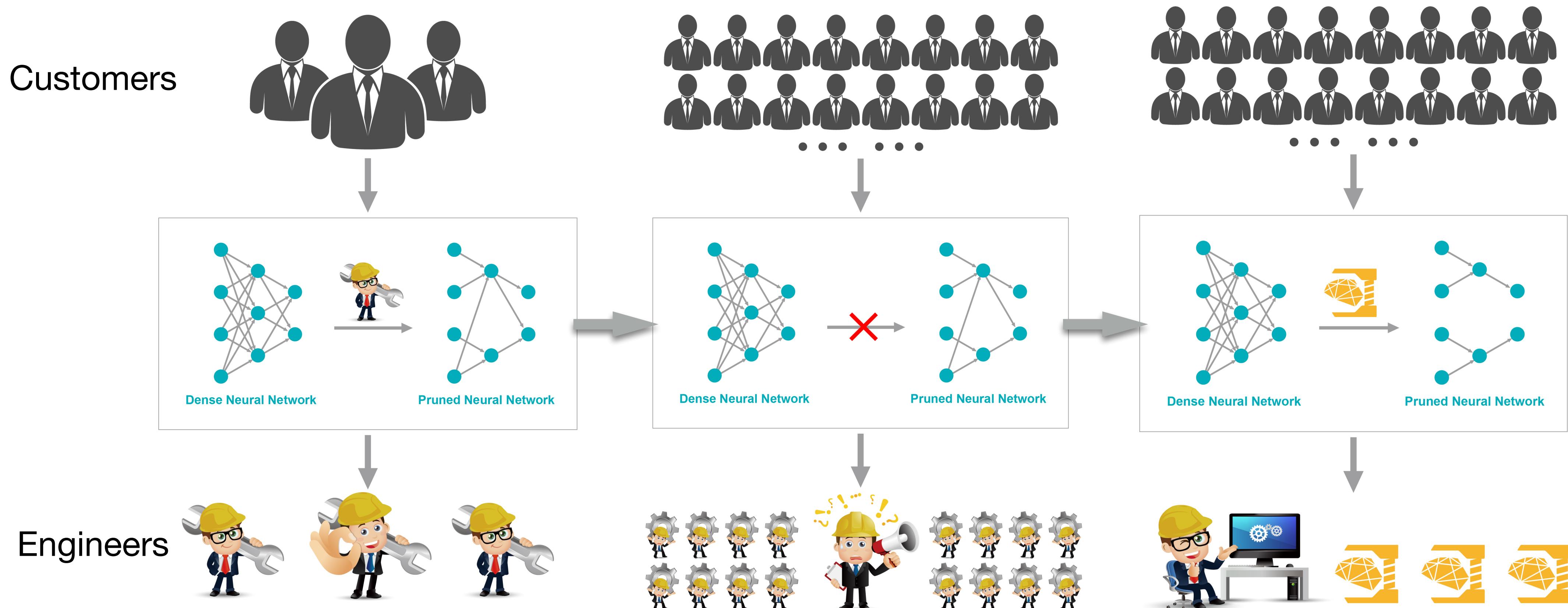


Automatic Pruning

- Given an **overall** compression ratio, how do we choose **per-layer** pruning ratios?
 - Sensitivity analysis ignores the interaction between layers -> sub-optimal

Automatic Pruning

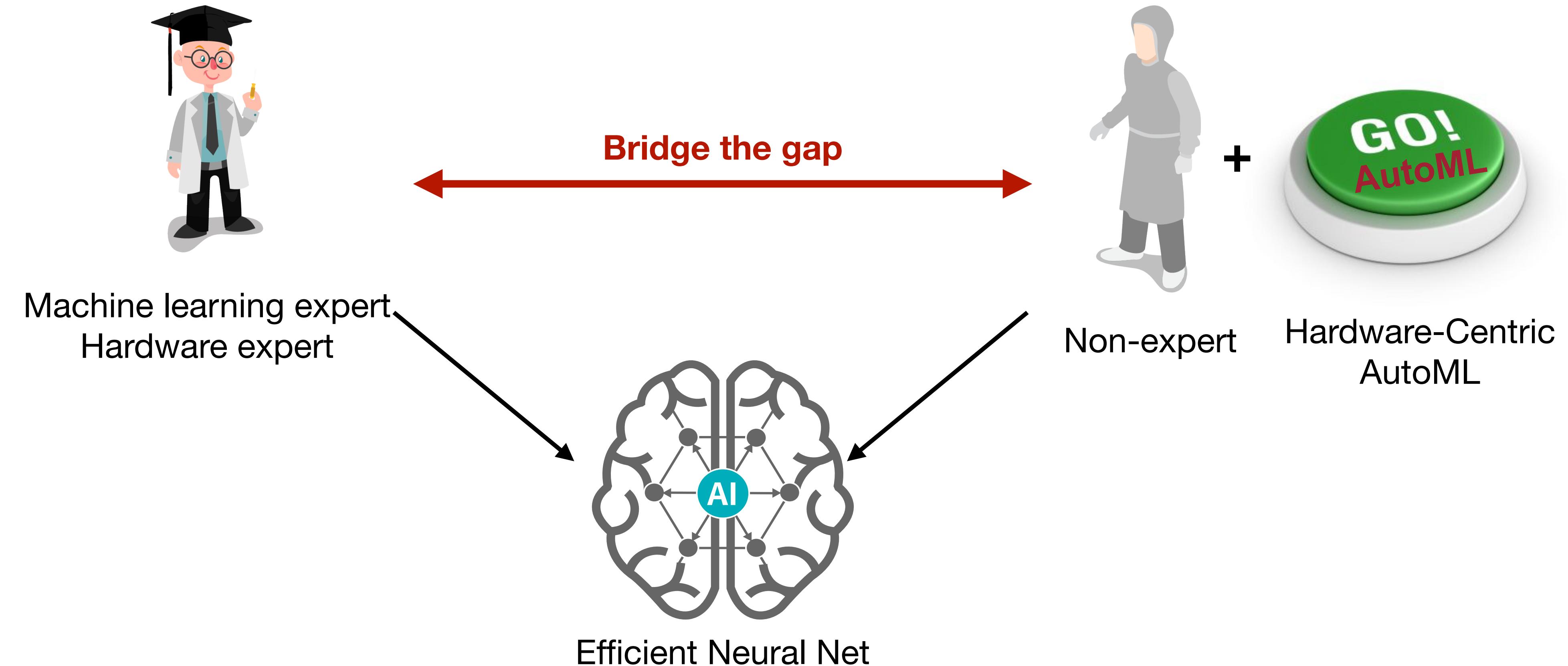
- Given an **overall** compression ratio, how do we choose **per-layer** pruning ratios?
 - Sensitivity analysis ignores the interaction between layers -> sub-optimal
- Conventionally, such process relies on human expertise and trials and errors



AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]

Automatic Pruning

- Can we develop a push-the-button solution?

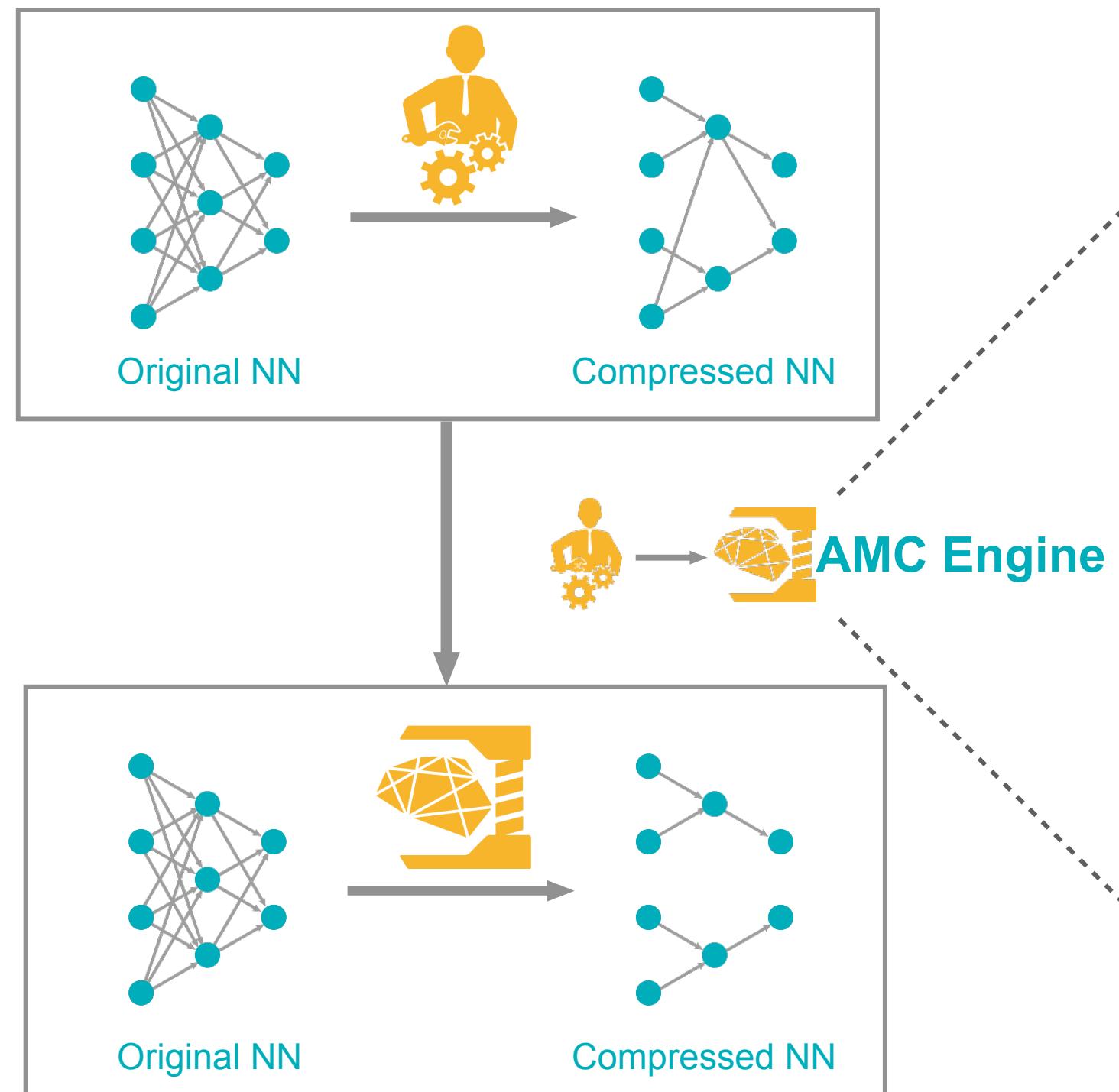


AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]

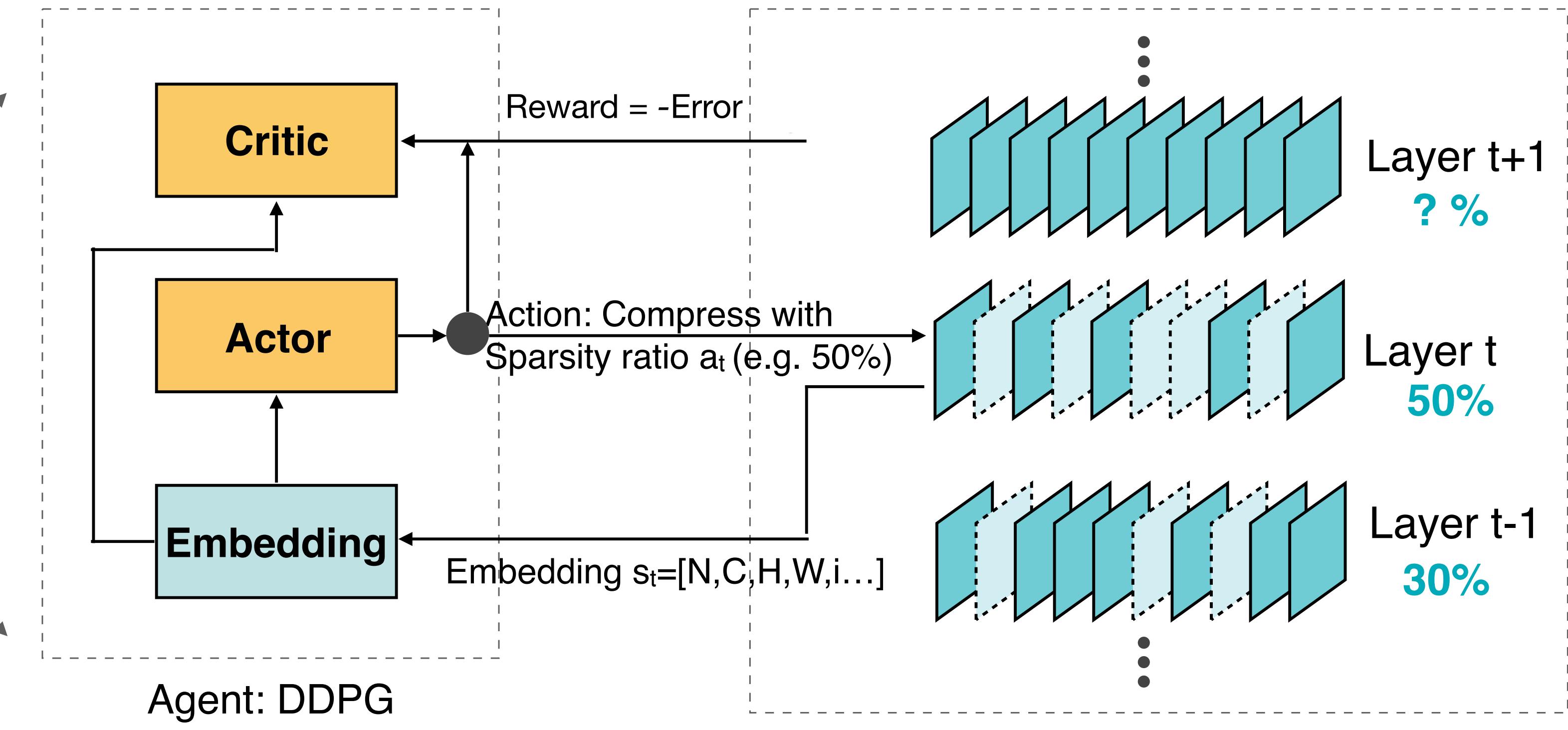
AMC: AutoML for Model Compression

Pruning as a reinforcement learning problem

Model Compression by Human:
Labor Consuming, Sub-optimal



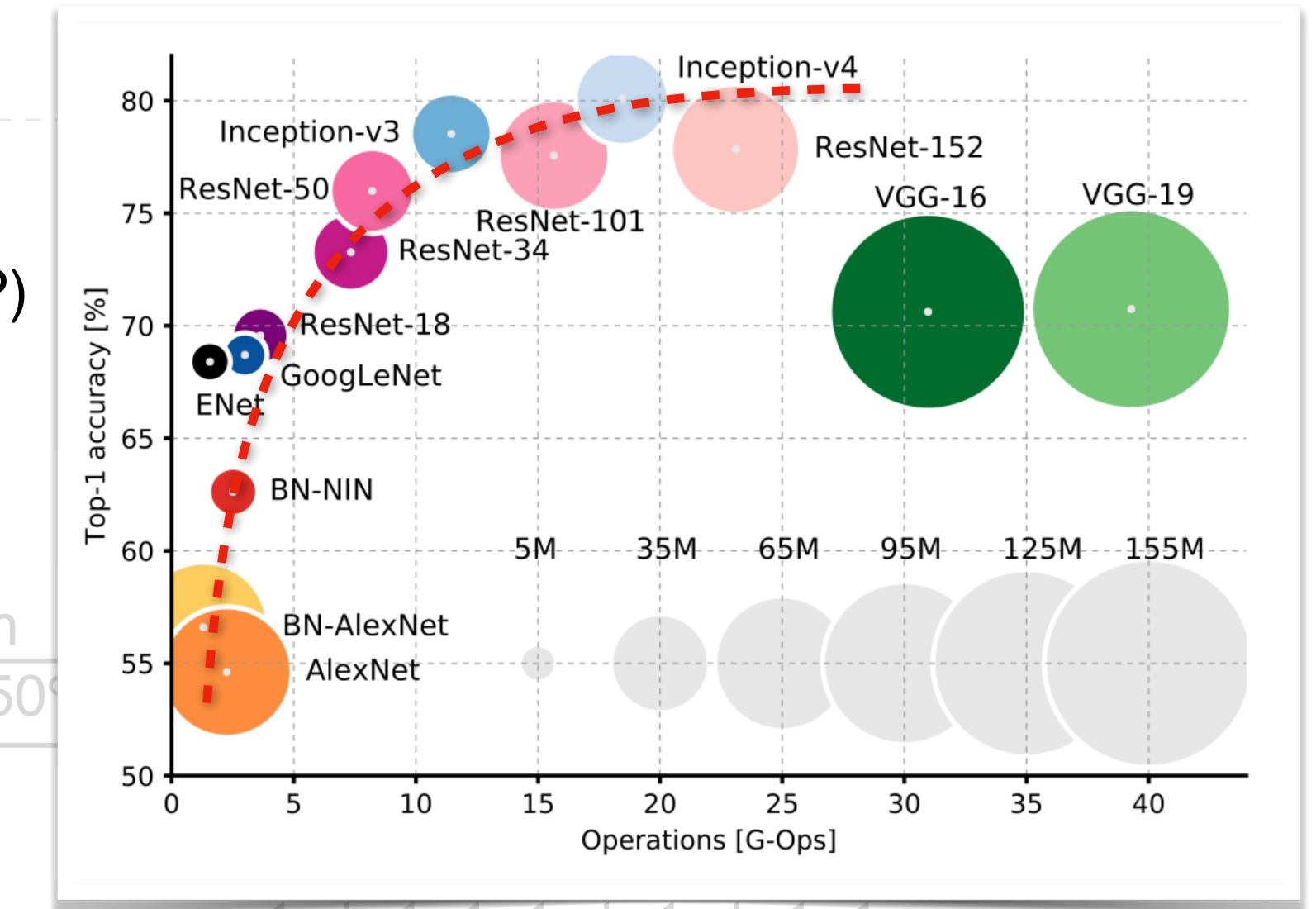
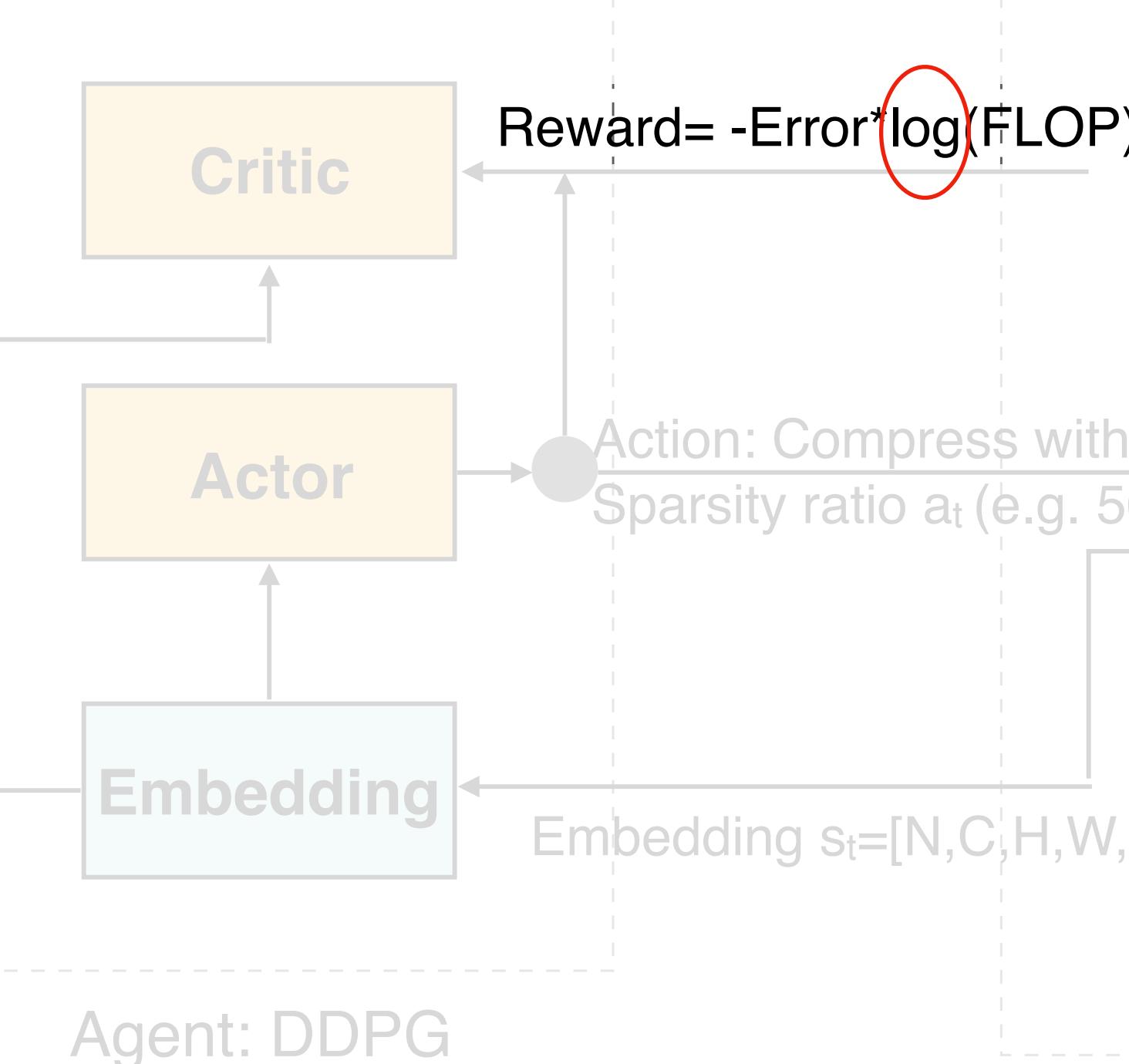
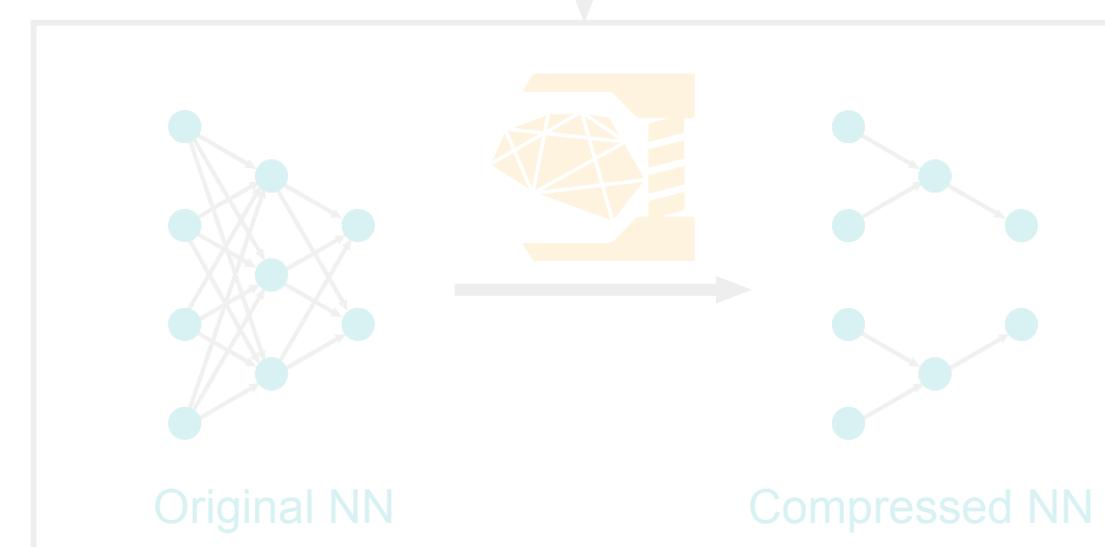
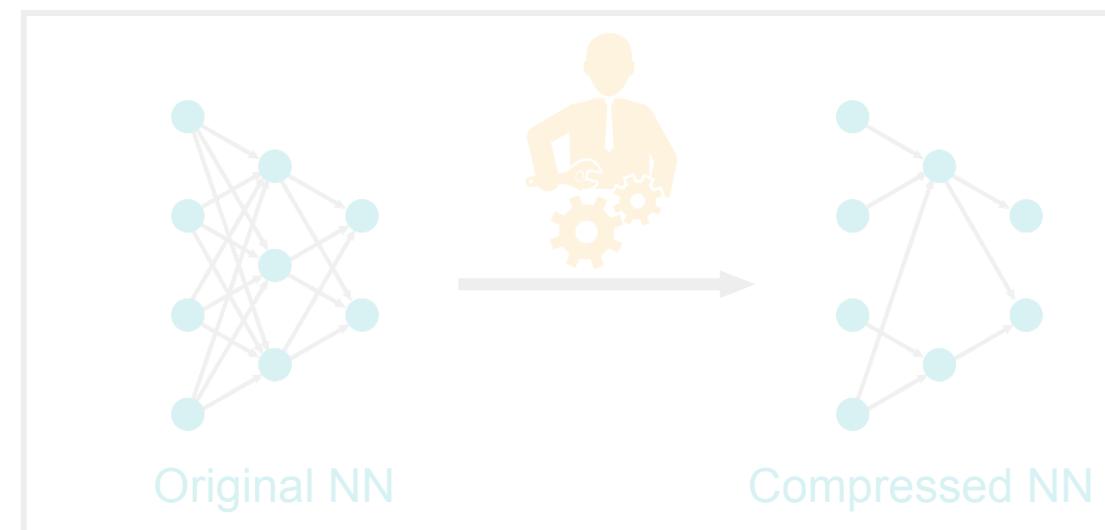
Model Compression by AI:
Automated, Higher Compression Rate, Faster



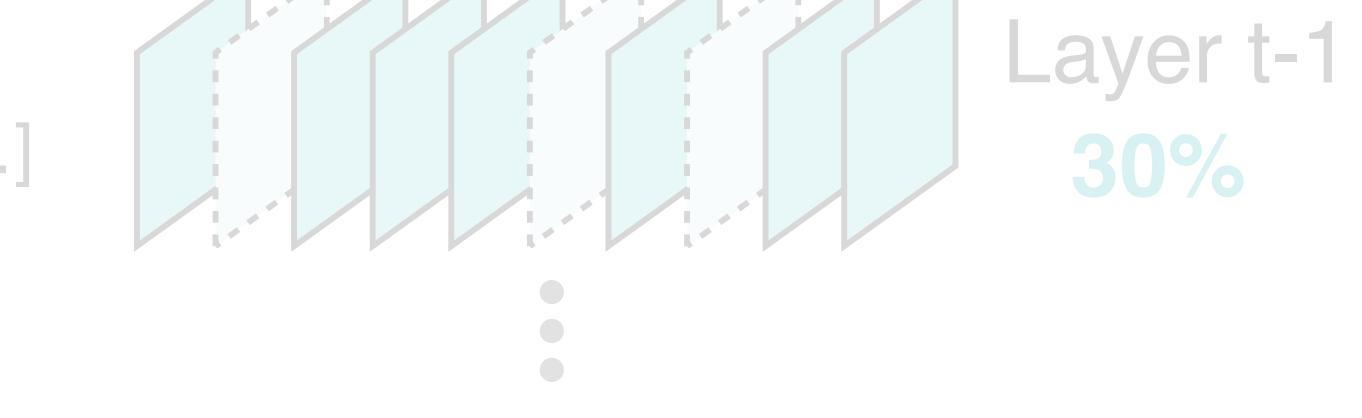
AMC: AutoML for Model Compression

Pruning as a reinforcement learning problem

Model Compression by Human:
Labor Consuming, Sub-optimal



Model Compression by AI:
Automated, Higher Compression Rate, Faster



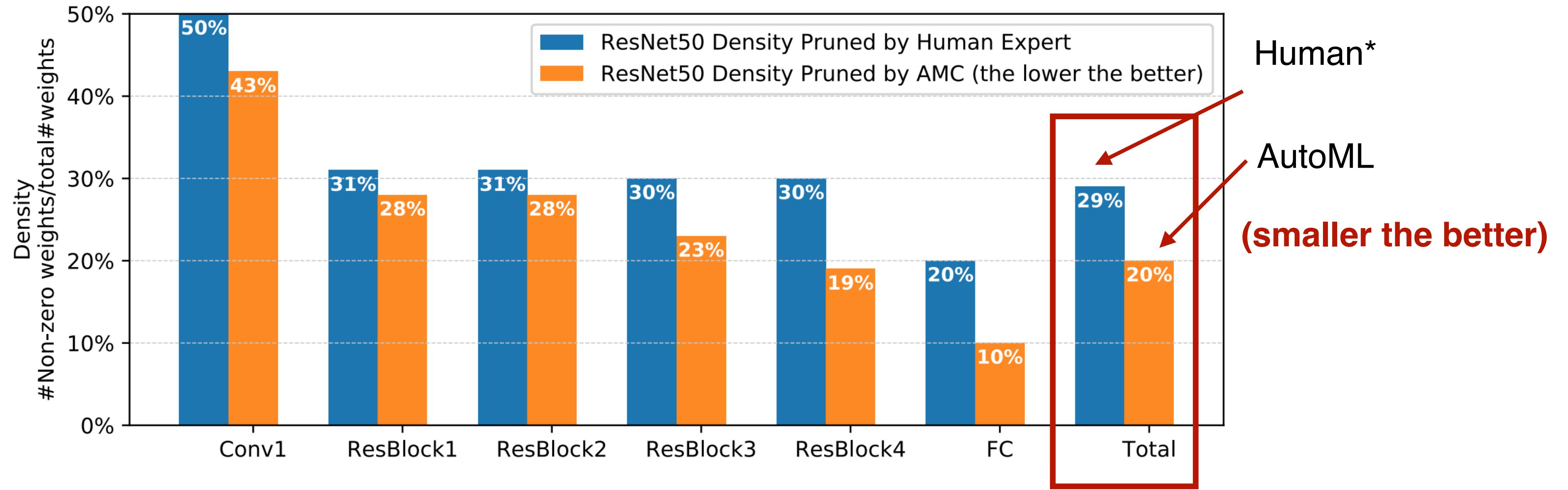
Environment: Channel Pruning

An Analysis of Deep Neural Network Models for Practical Applications [Canziani et al., 2016]

AMC: AutoML for Model Compression

- AMC uses the following setups for the reinforcement learning problem
 - **State:** features including layer indices, channel numbers, kernel sizes, FLOPs, ...
 - **Action:** A continuous number (pruning ratio) $a \in [0,1)$
 - **Agent:** DDPG agent, since it supports continuous action output
 - **Reward:** $R = \begin{cases} -\text{Error}, & \text{if satisfies constraints} \\ -\infty, & \text{if not} \end{cases}$
 - We can also optimize **latency** constraints with a pre-built lookup table (LUT)

AMC: AutoML for Model Compression



* Efficient Methods and Hardware for Deep Learning [Han, *thesis*]

AMC: AutoML for Model Compression

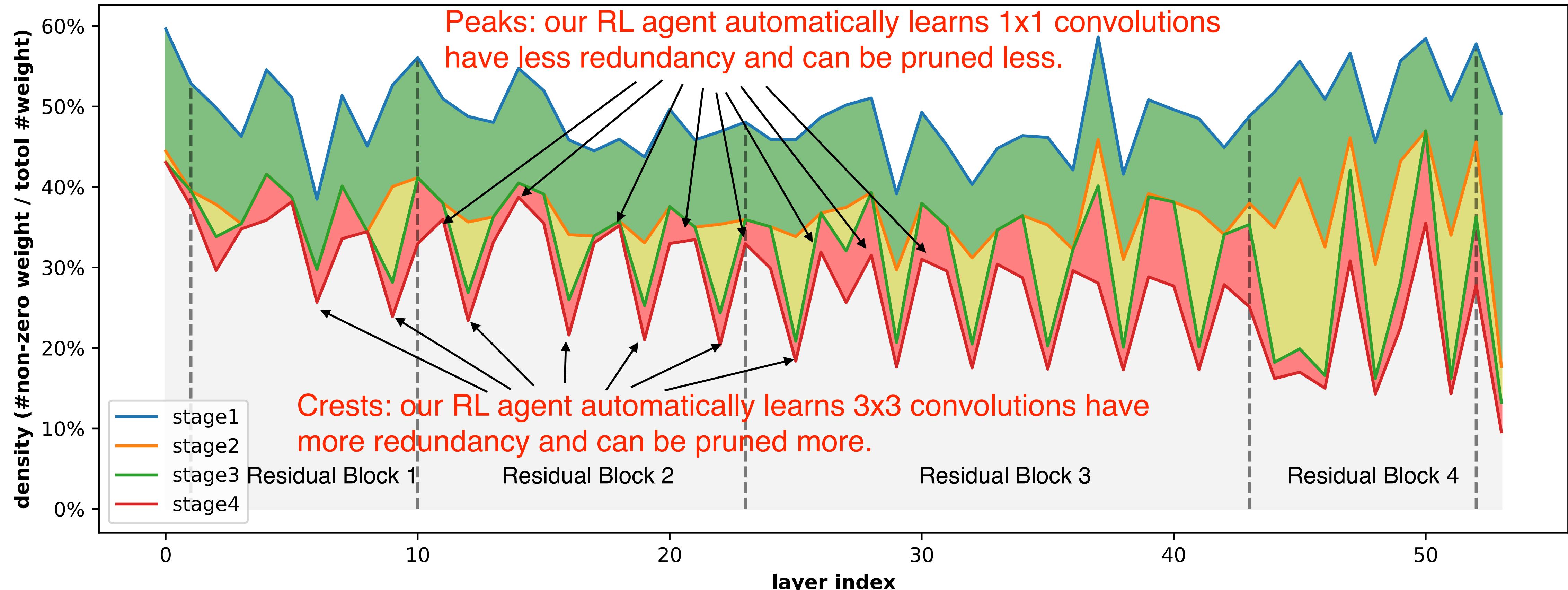


Figure 14: The pruning policy (sparsity ratio) given by our reinforcement learning agent for ResNet-50.

AMC: AutoML for Model Compression



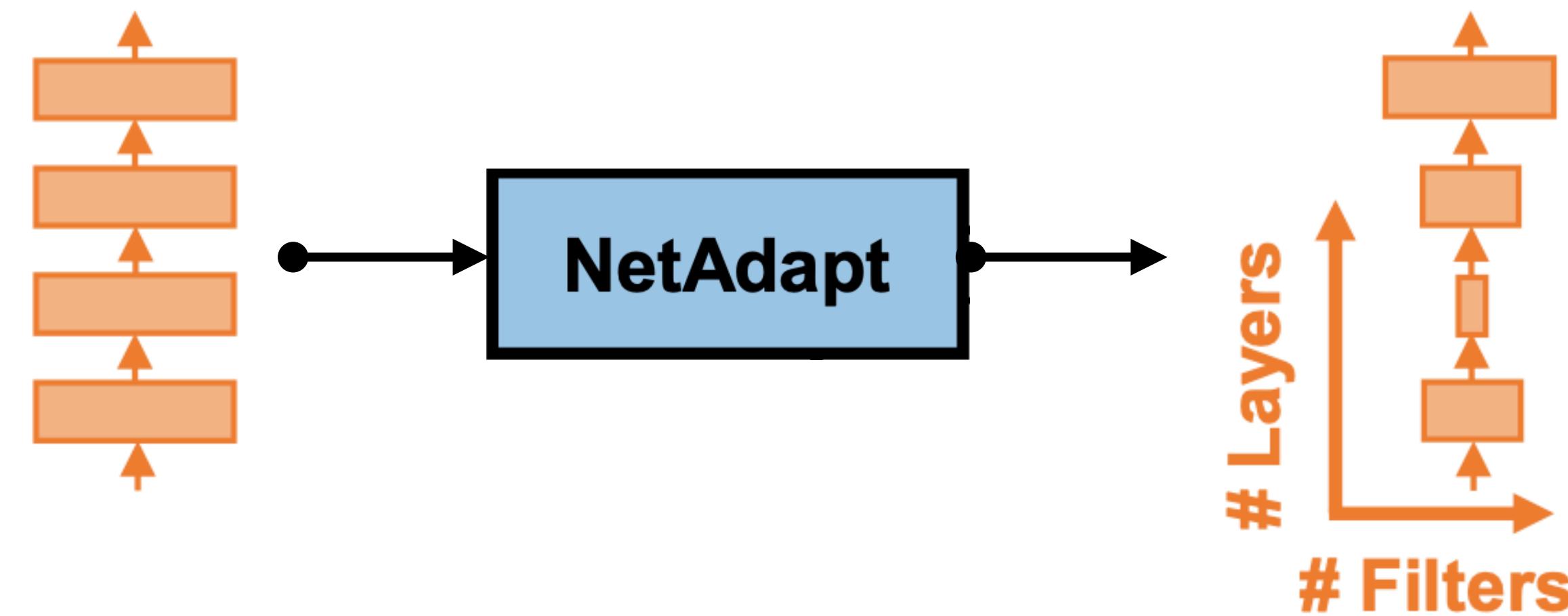
Model	MAC	Top-1	Latency*	Speedup	Memory
1.0 MobileNet	569M	70.6%	119.0ms	1x	20.1MB
AMC (50% FLOPs)	285M	70.5%	64.4ms	1.8x	14.3MB
AMC (50% Time)	272M	70.2%	59.7ms	2.0x	13.2MB
0.75 MobileNet	325M	68.4%	69.5ms	1.7x	14.8MB

* Measured with TF-Lite on Samsung Galaxy S7 Edge, which has Qualcomm Snapdragon SoC
Single core, Batch size = 1(mobile, latency oriented)

NetAdapt

A rule-based iterative/progressive method

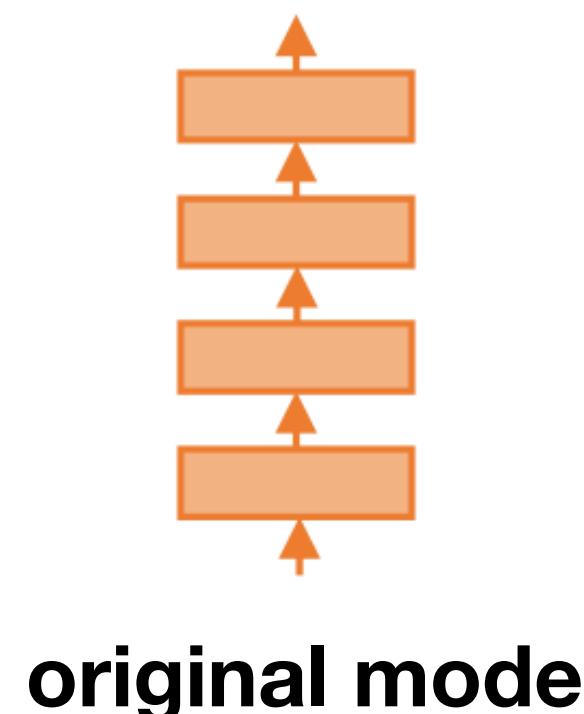
- The goal of NetAdapt is to find a per-layer pruning ratio to meet a global resource constraint (e.g., latency, energy, ...)
- The process is done iteratively
- We take **latency** constraint as an example



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

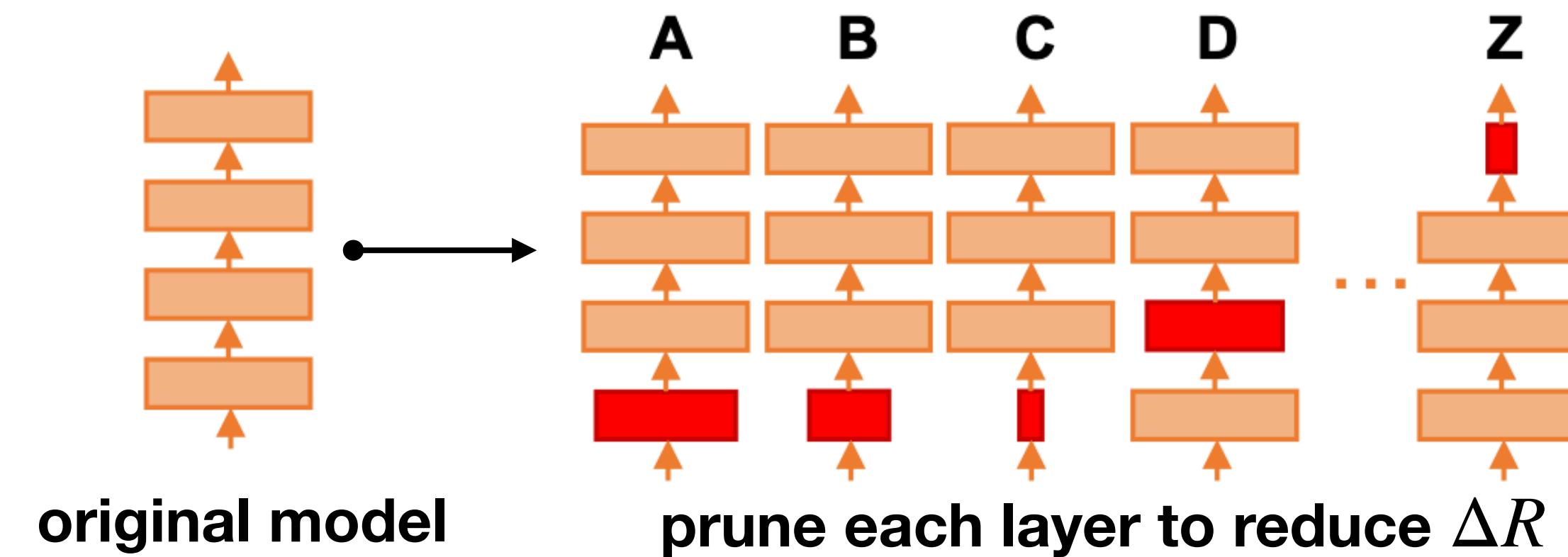
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

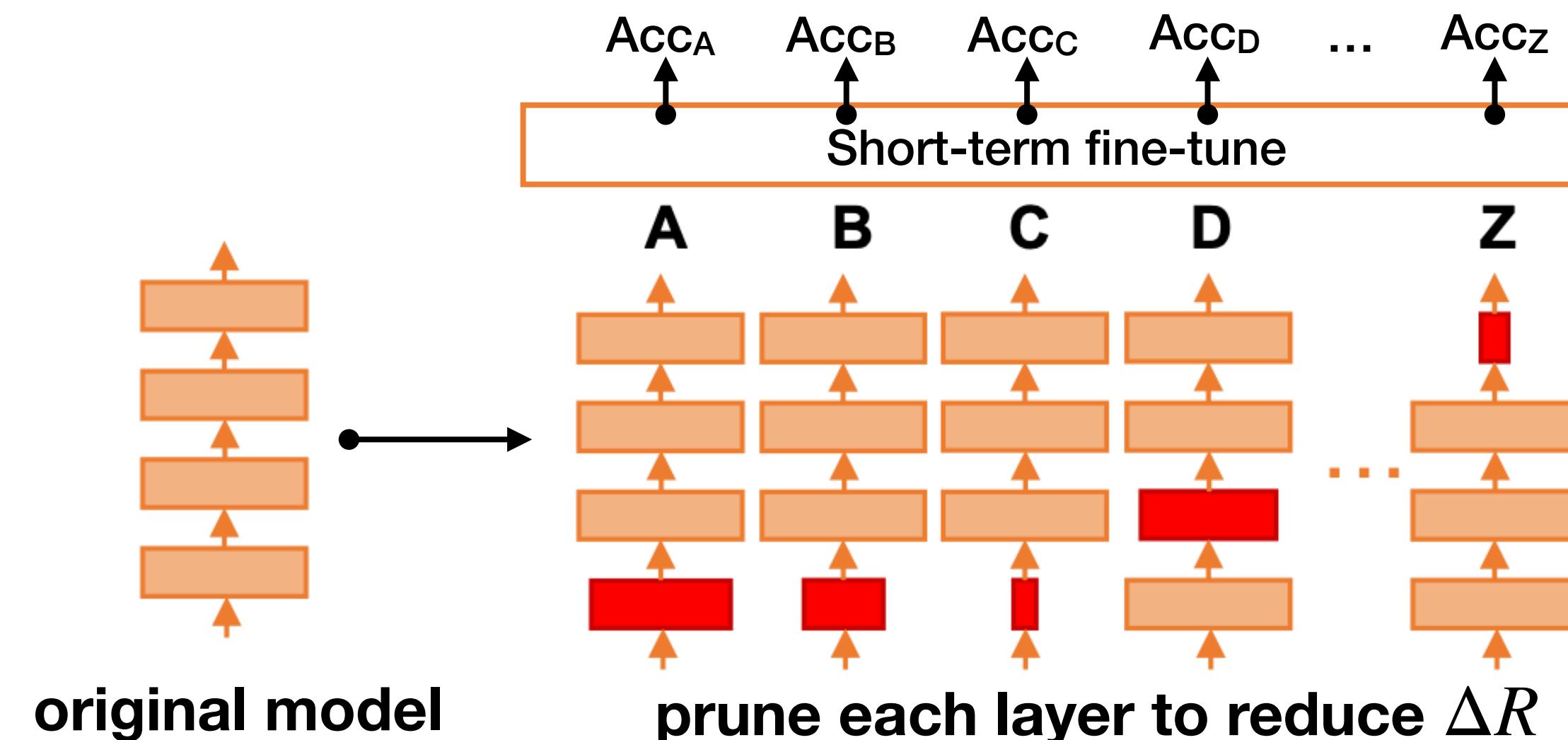
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
 - For each layer L_k (k in A-Z in the figure)
 - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

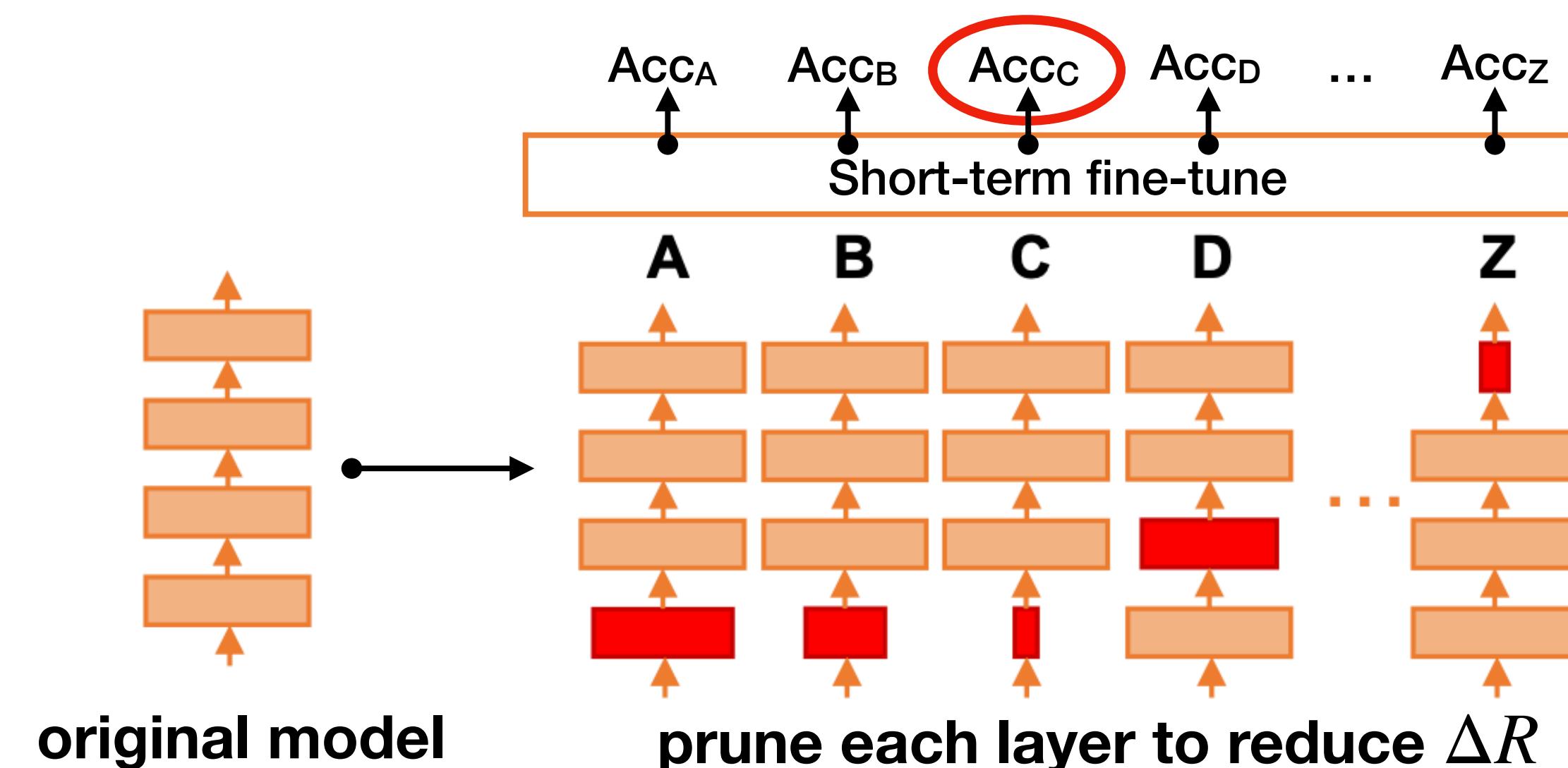
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
 - For each layer L_k (k in A-Z in the figure)
 - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
 - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

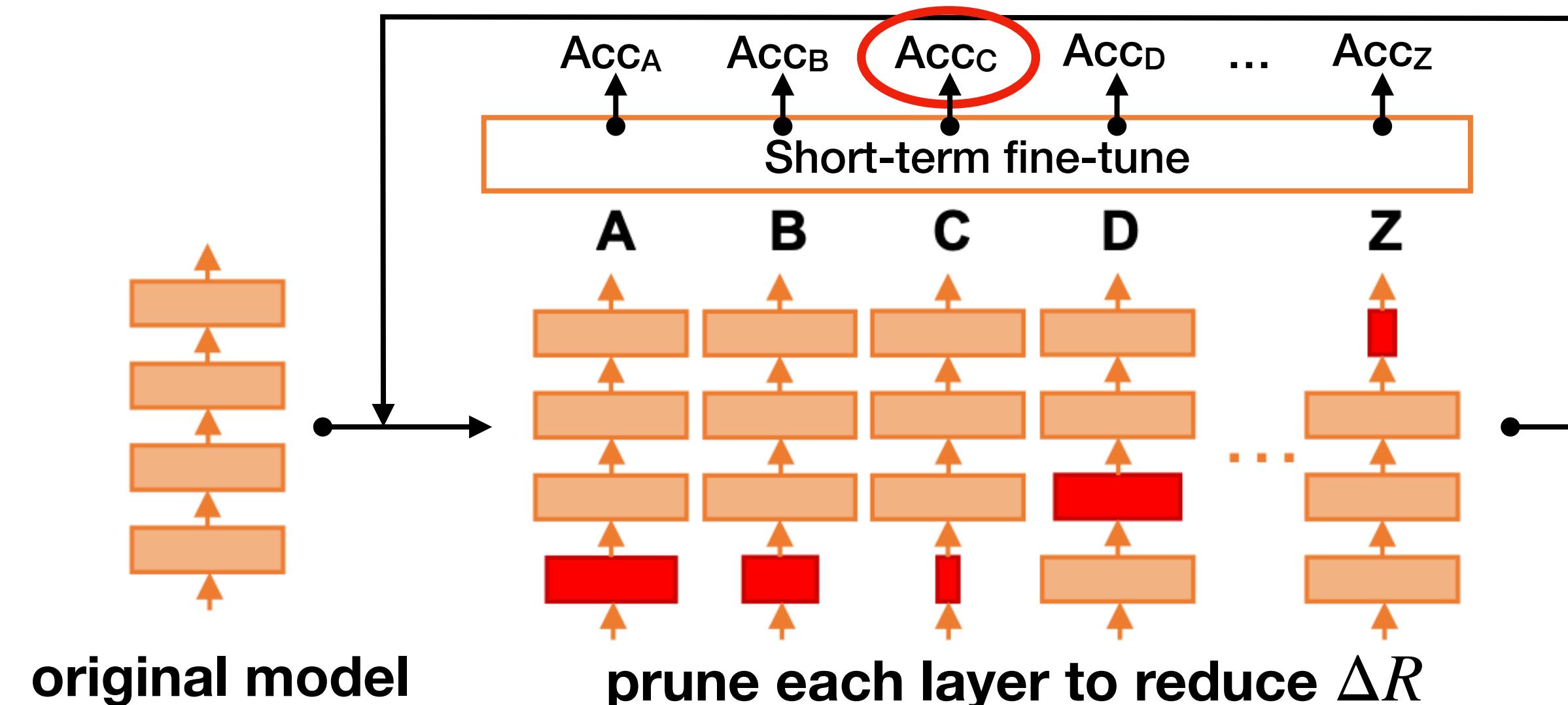
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
 - For each layer L_k (k in A-Z in the figure)
 - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
 - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning
 - Choose and prune the layer with the highest accuracy



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

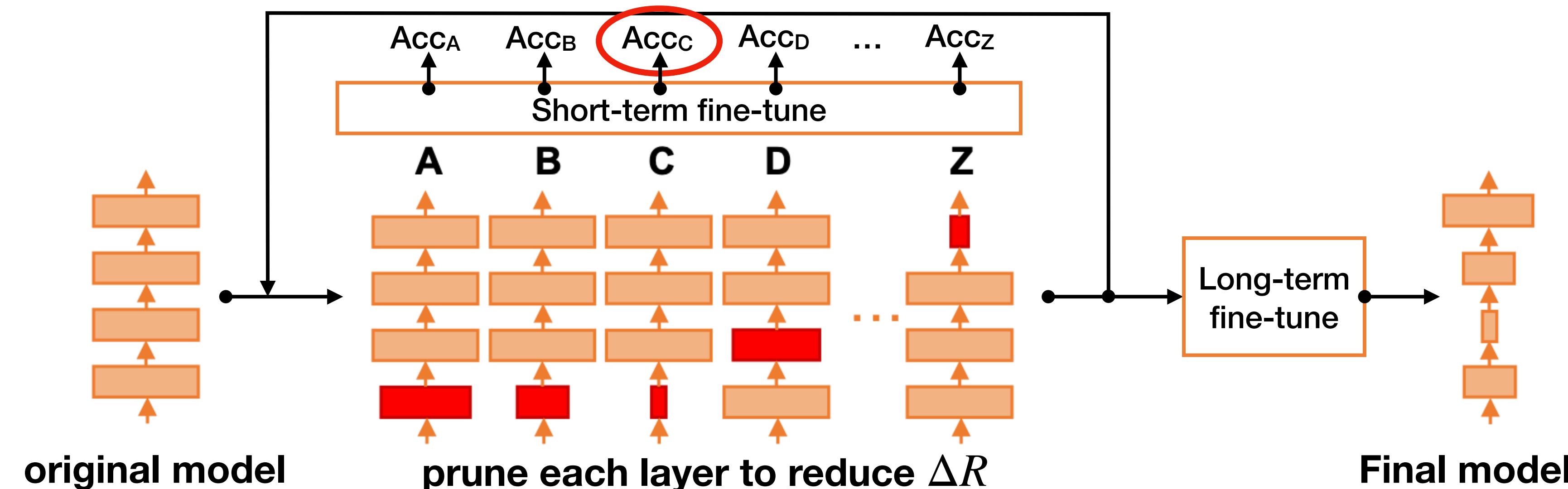
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
 - For each layer L_k (k in A-Z in the figure)
 - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
 - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning
 - Choose and prune the layer with the highest accuracy
 - Repeat until the total latency reduction satisfies the constraint



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

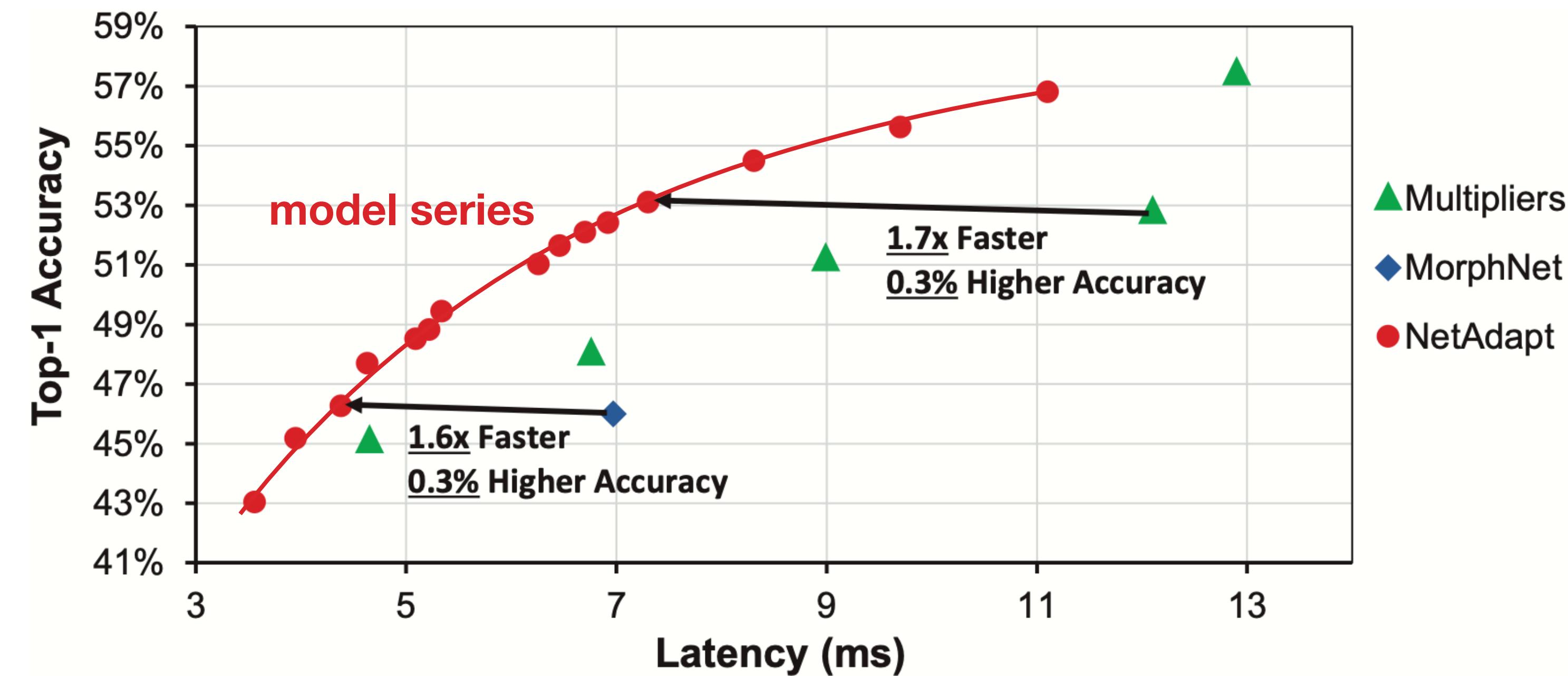
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
 - For each layer L_k (k in A-Z in the figure)
 - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
 - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning
 - Choose and prune the layer with the highest accuracy
- Repeat until the total latency reduction satisfies the constraint
- Long-term fine-tune to recover accuracy



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

NetAdapt

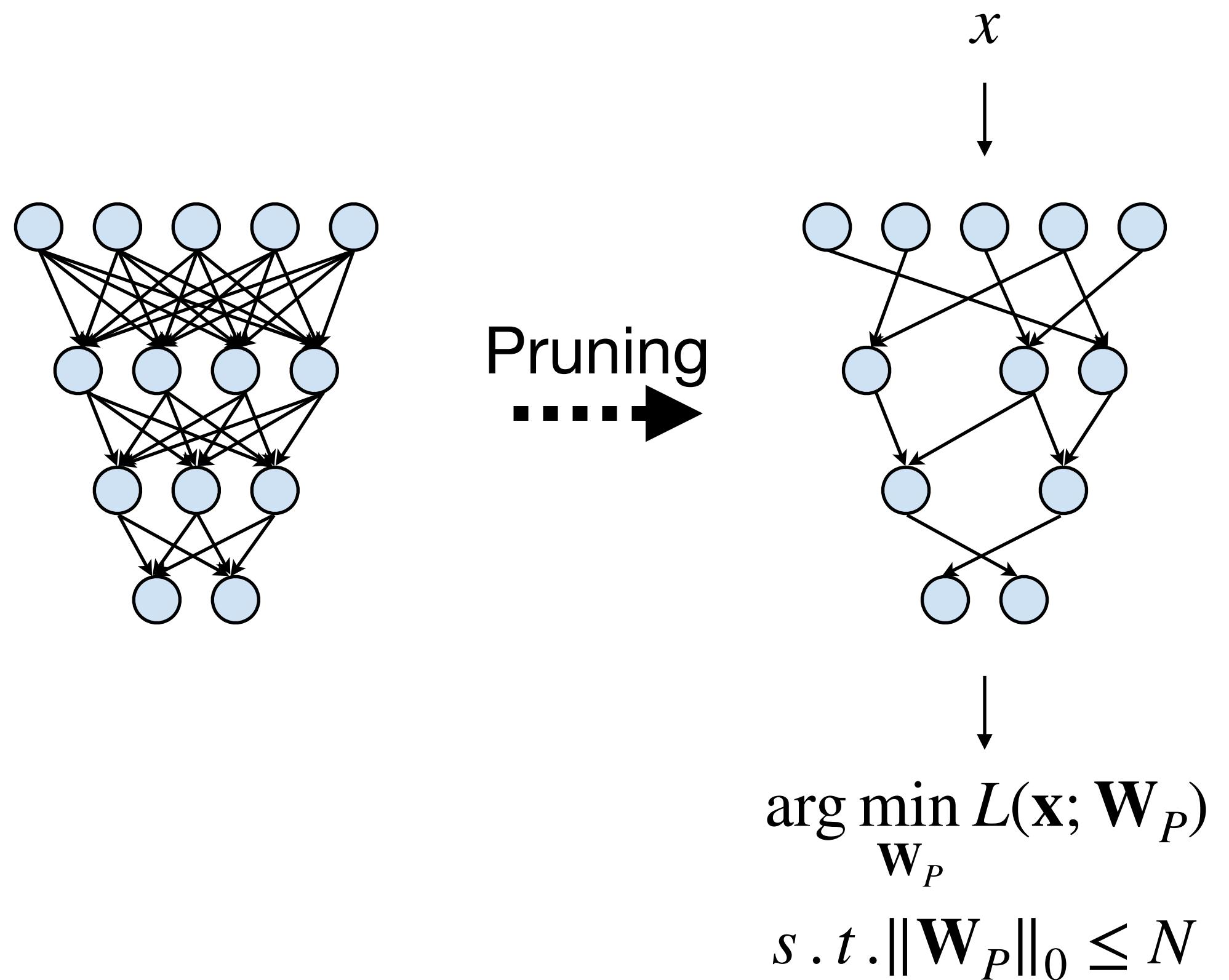
- The iterative nature allows us to obtain a **serial of models** with different costs
 - #models = #iterations



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang et al., ECCV 2018]

Neural Network Pruning

- **Introduction to Pruning**
 - What is pruning?
 - How should we formulate pruning?
- **Determine the Pruning Granularity**
 - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
 - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
 - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
 - How should we improve performance of pruned models?



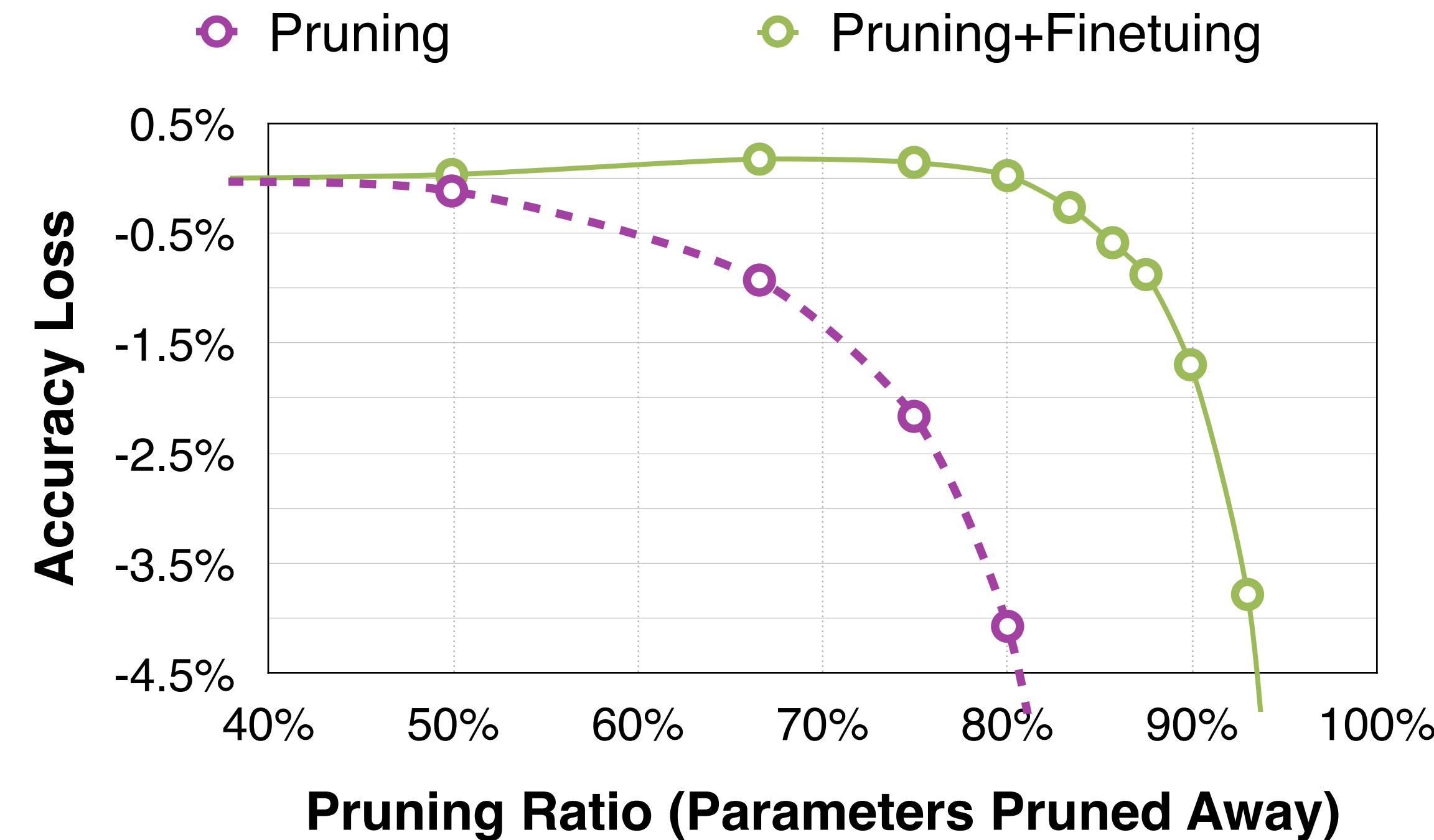
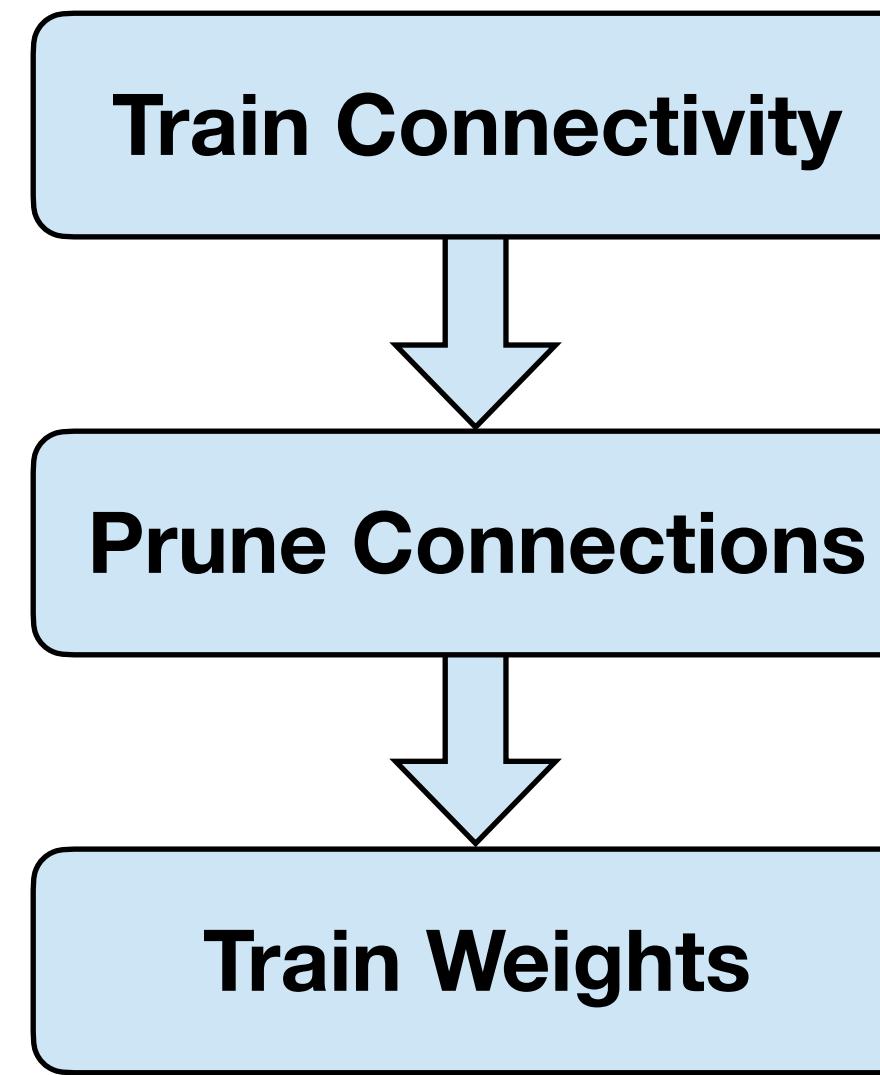
Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Section 2: Fine-tuning / Training

How should we improve performance of sparse models?

Finetuning Pruned Neural Networks

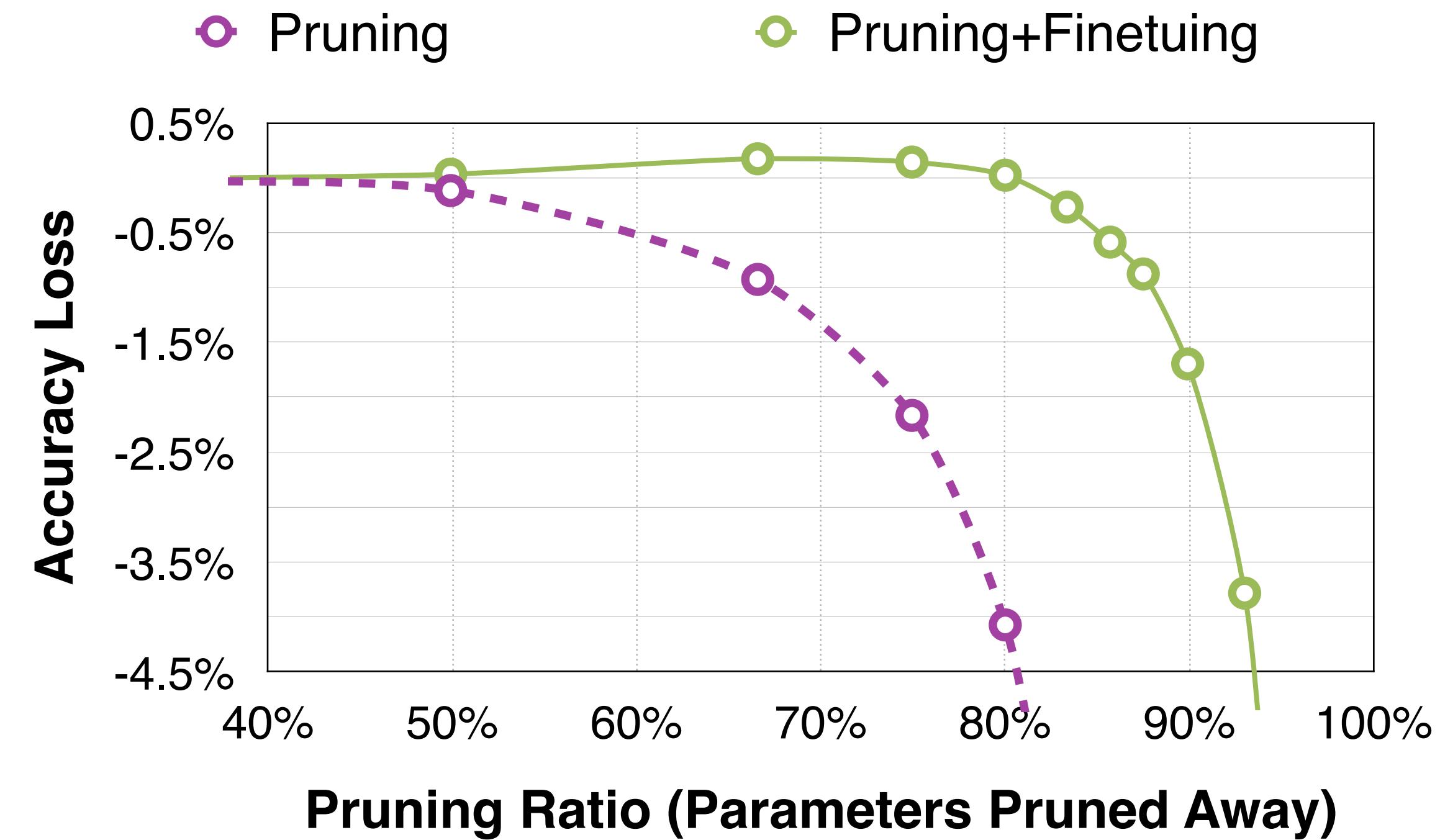
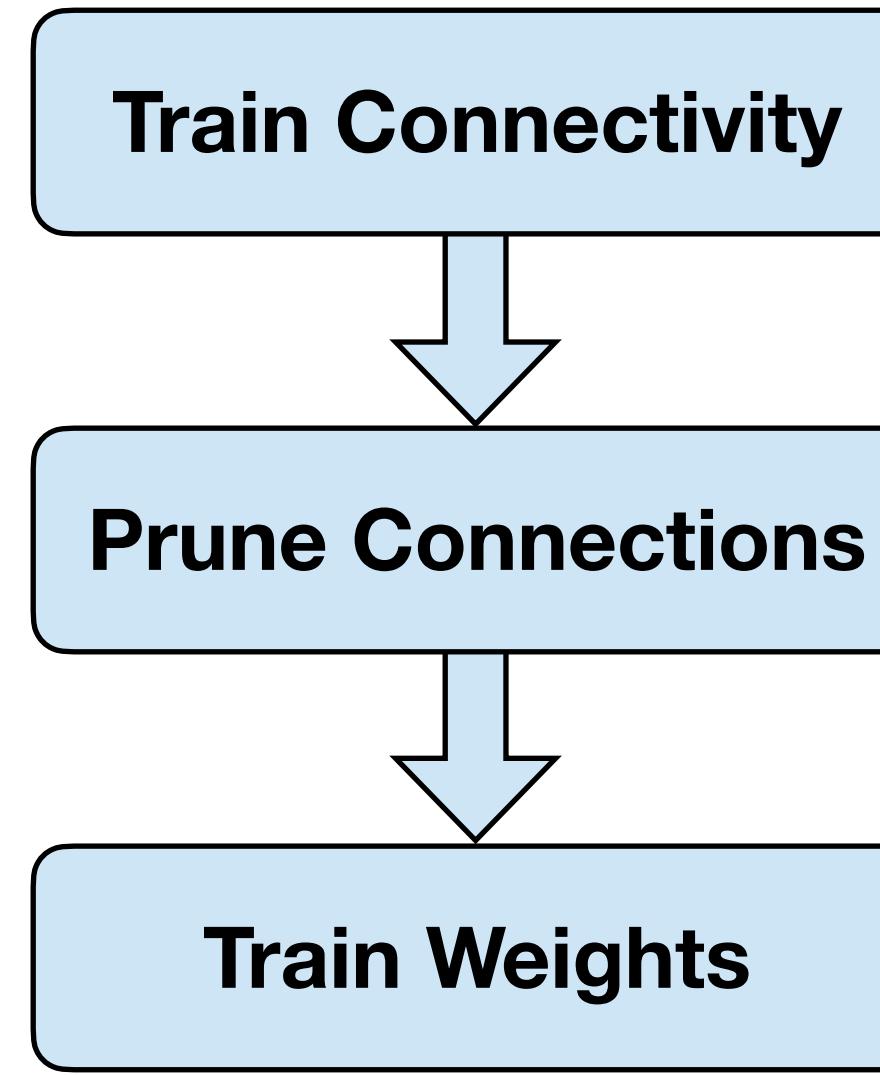
- After pruning, the model may decrease, especially for larger pruning ratio.
- Fine-tuning the pruned neural networks will help recover the accuracy and push the pruning ratio higher.
 - **Learning rate for fine-tuning is usually 1/100 or 1/10 of the original learning rate.**



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

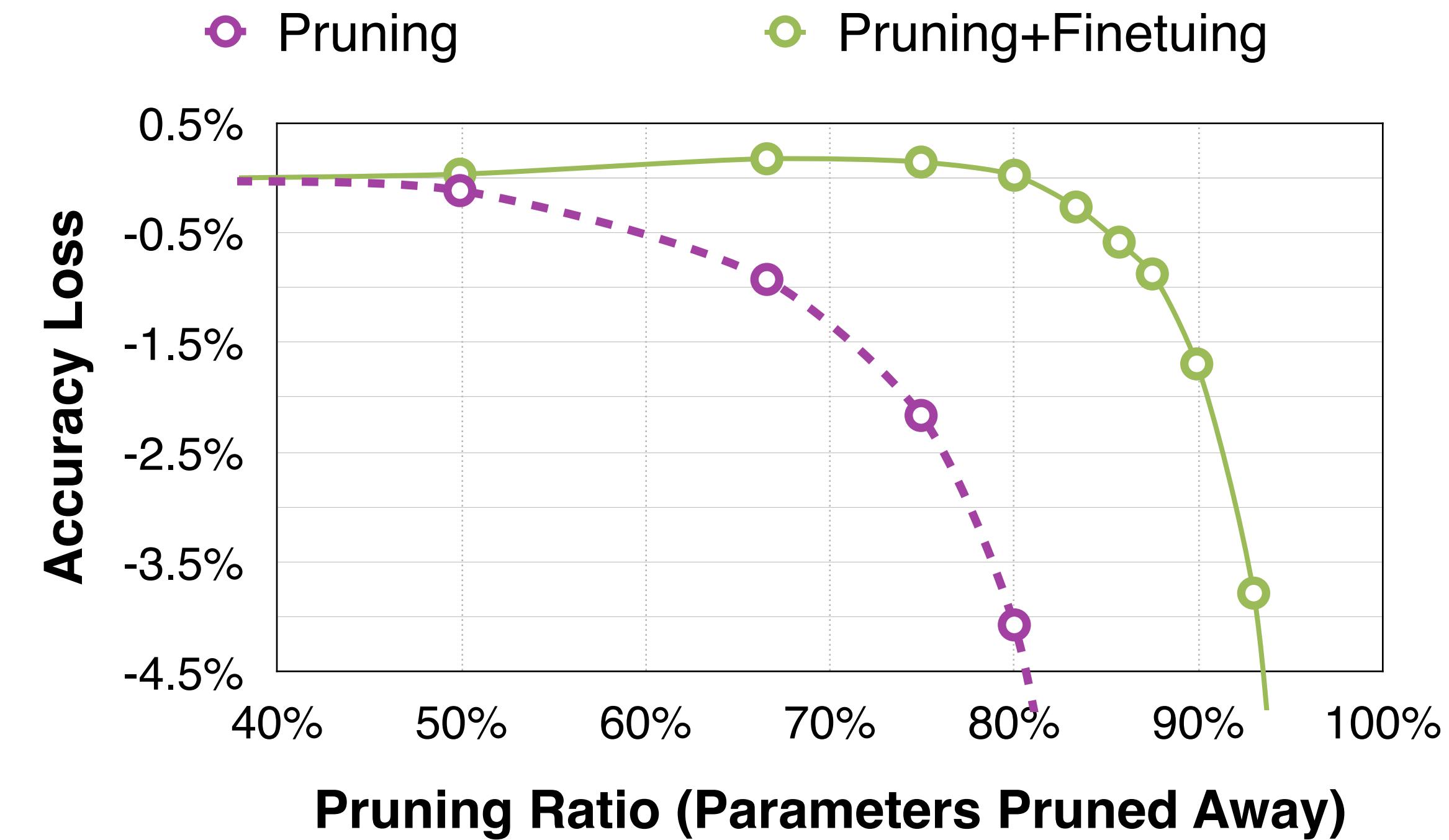
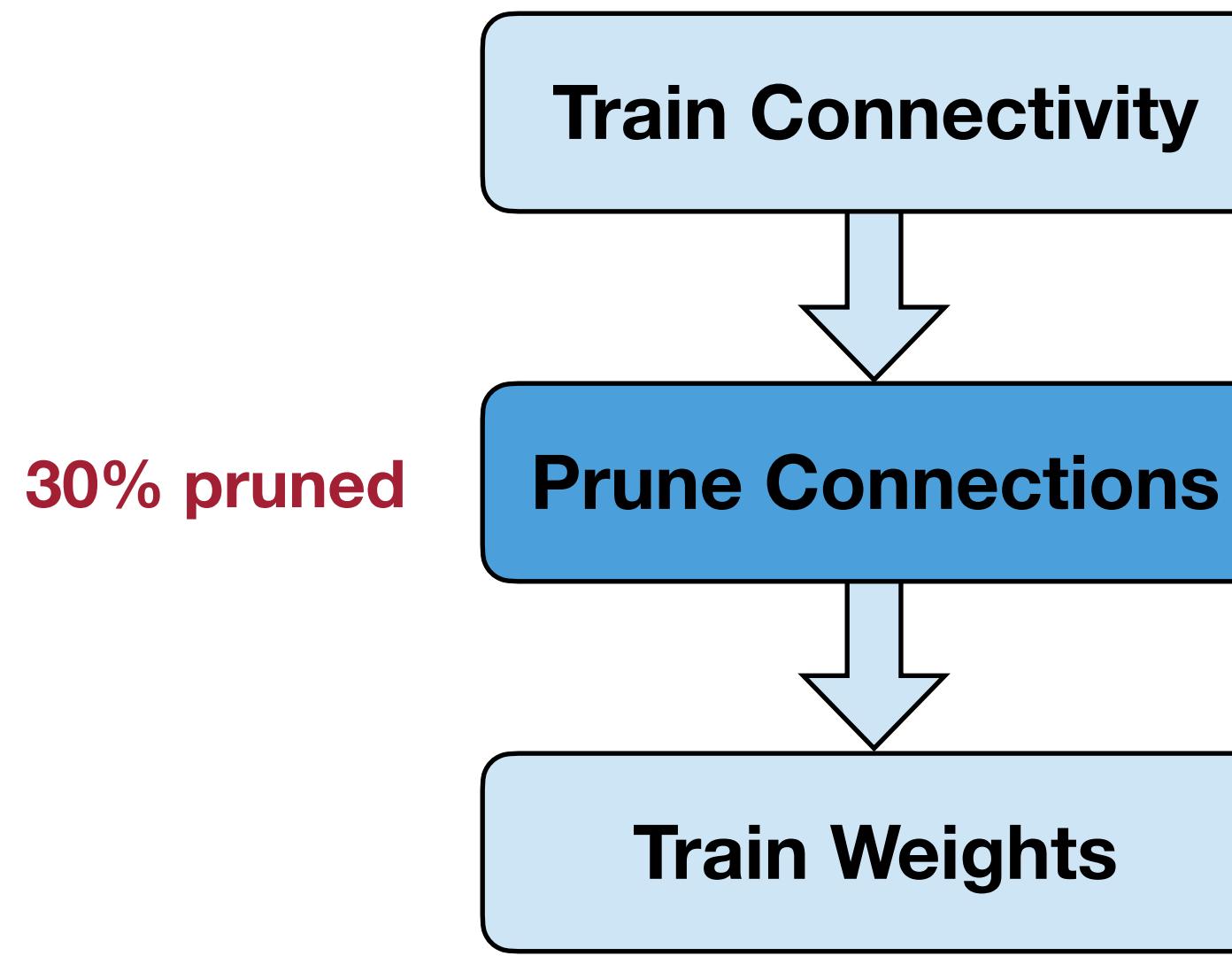
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

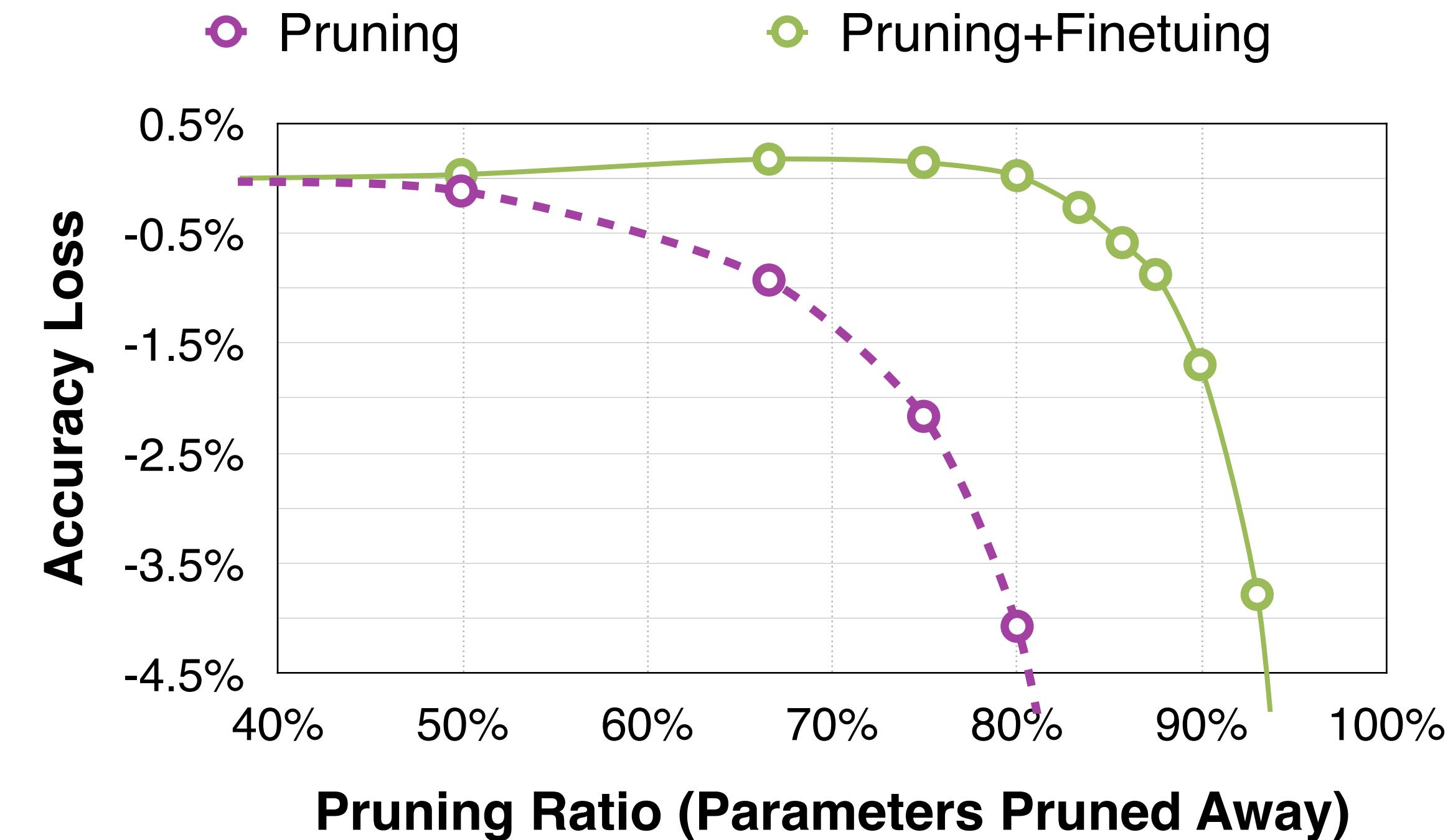
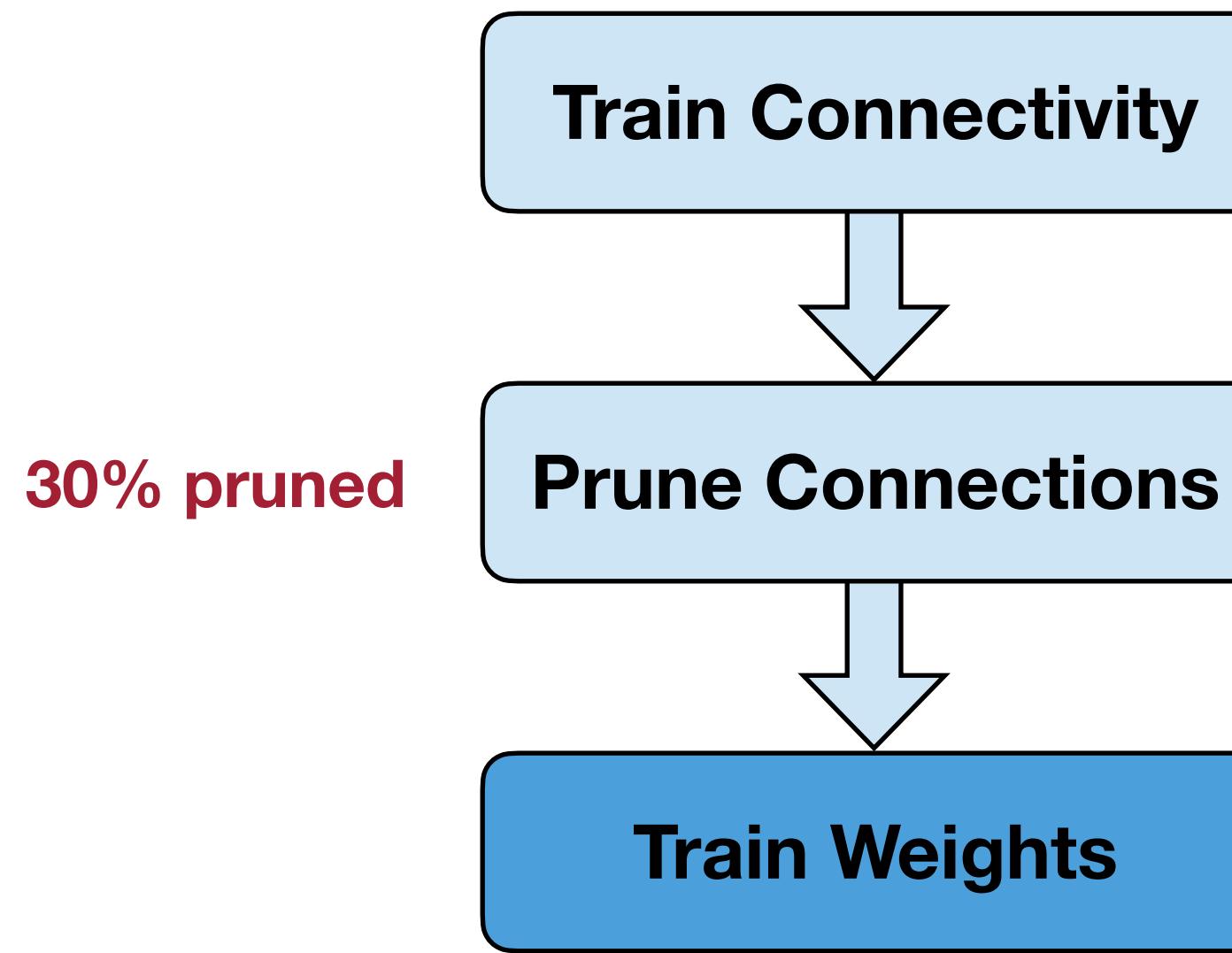
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

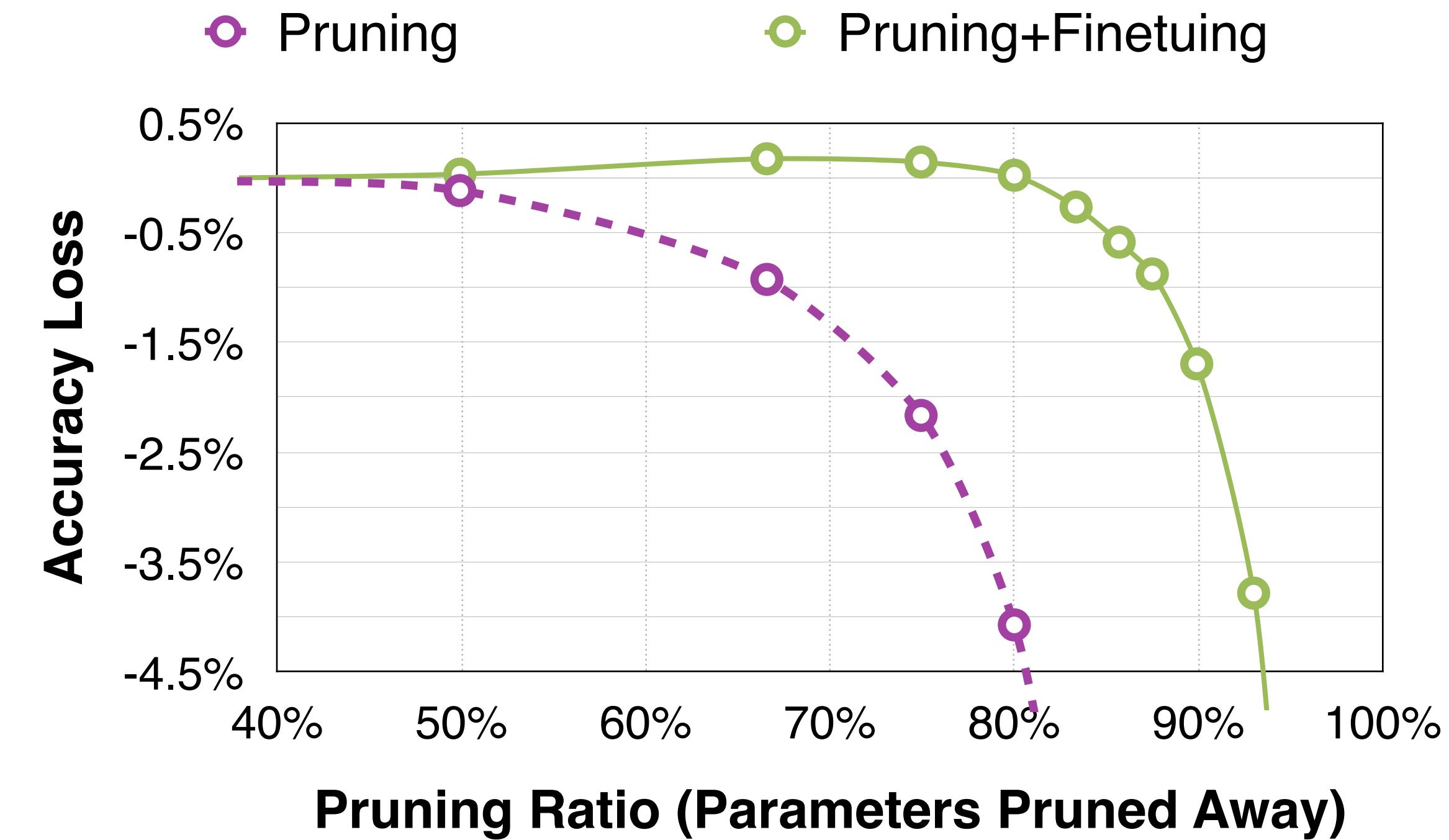
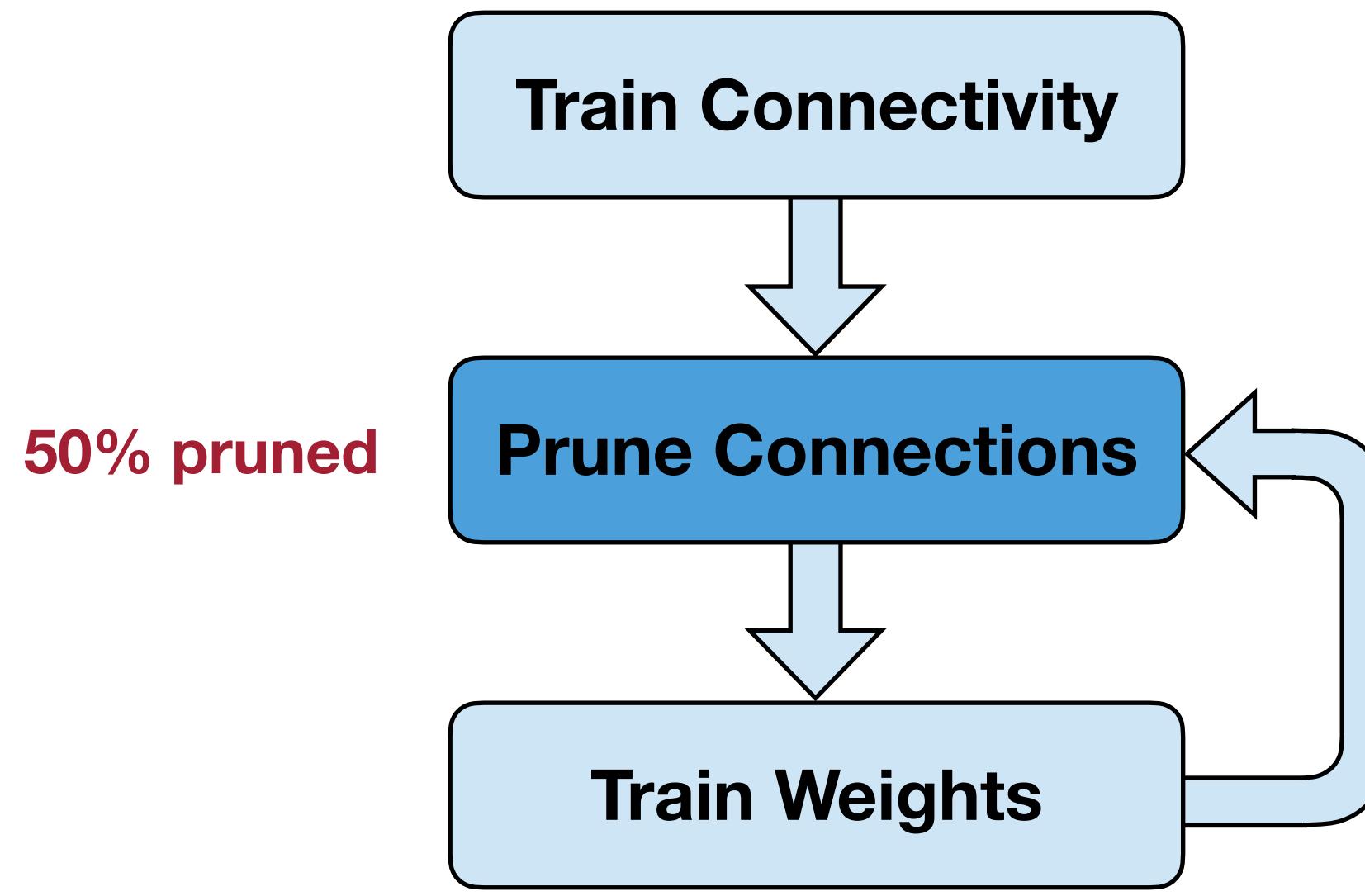
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

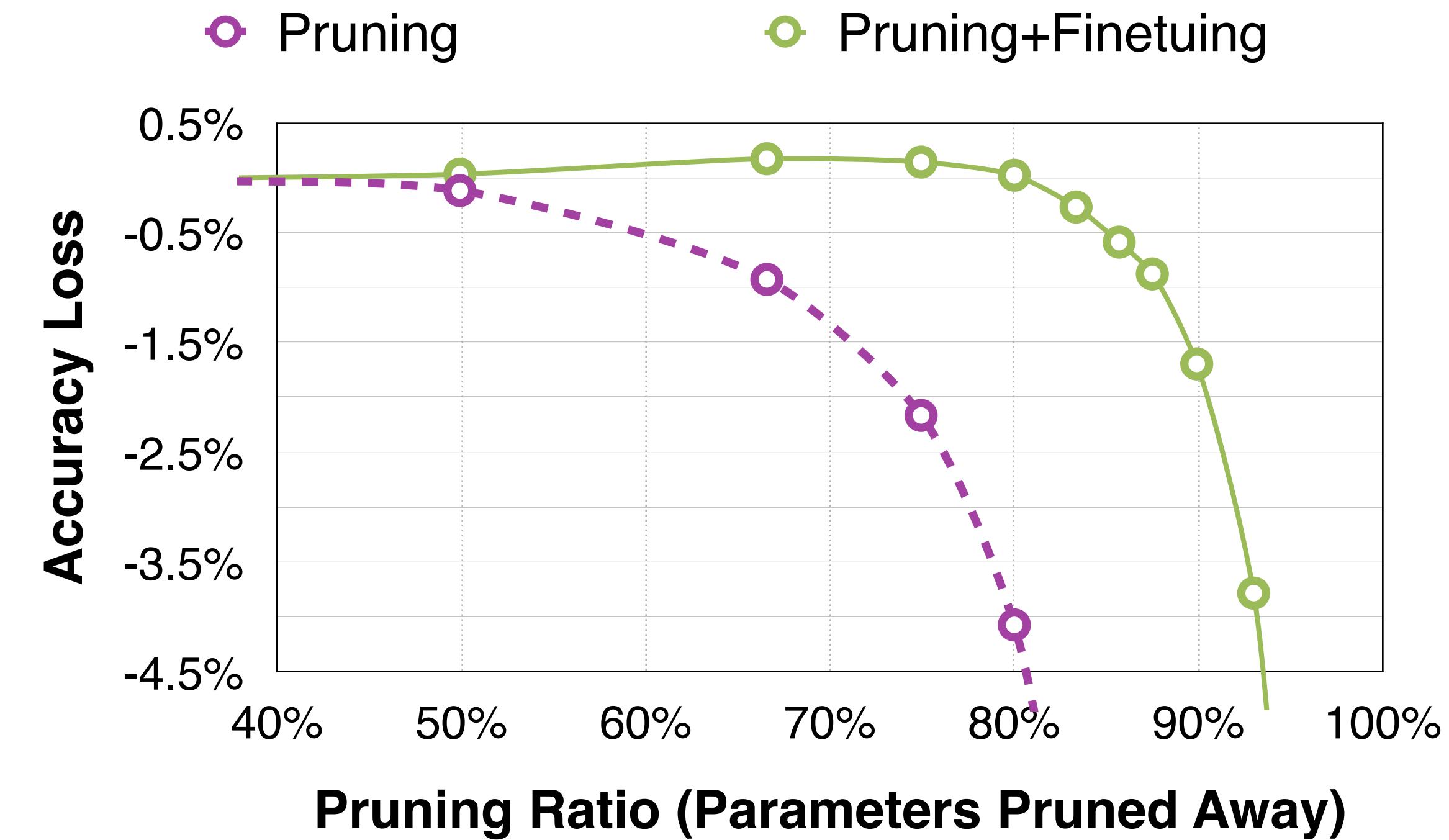
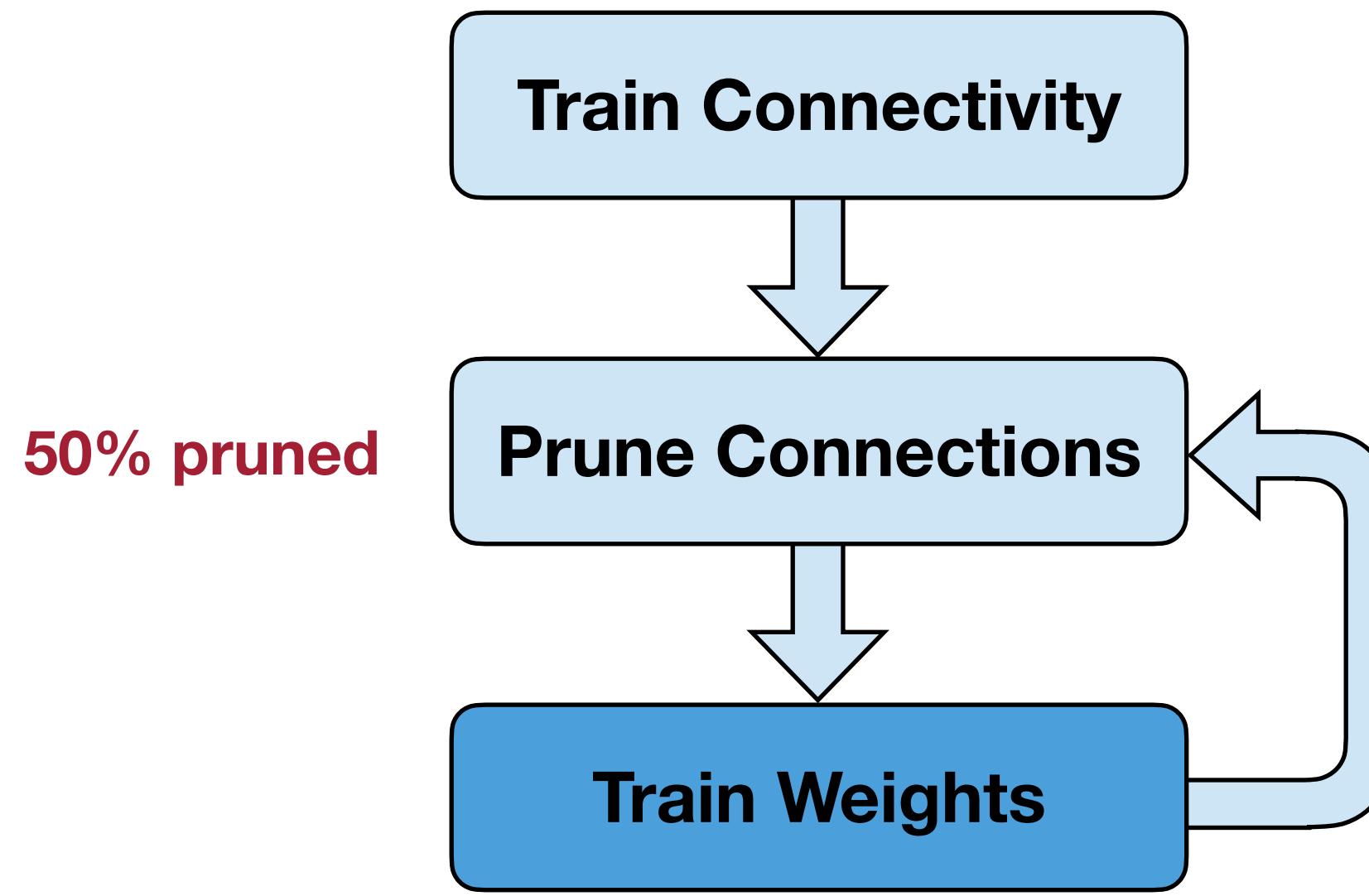
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

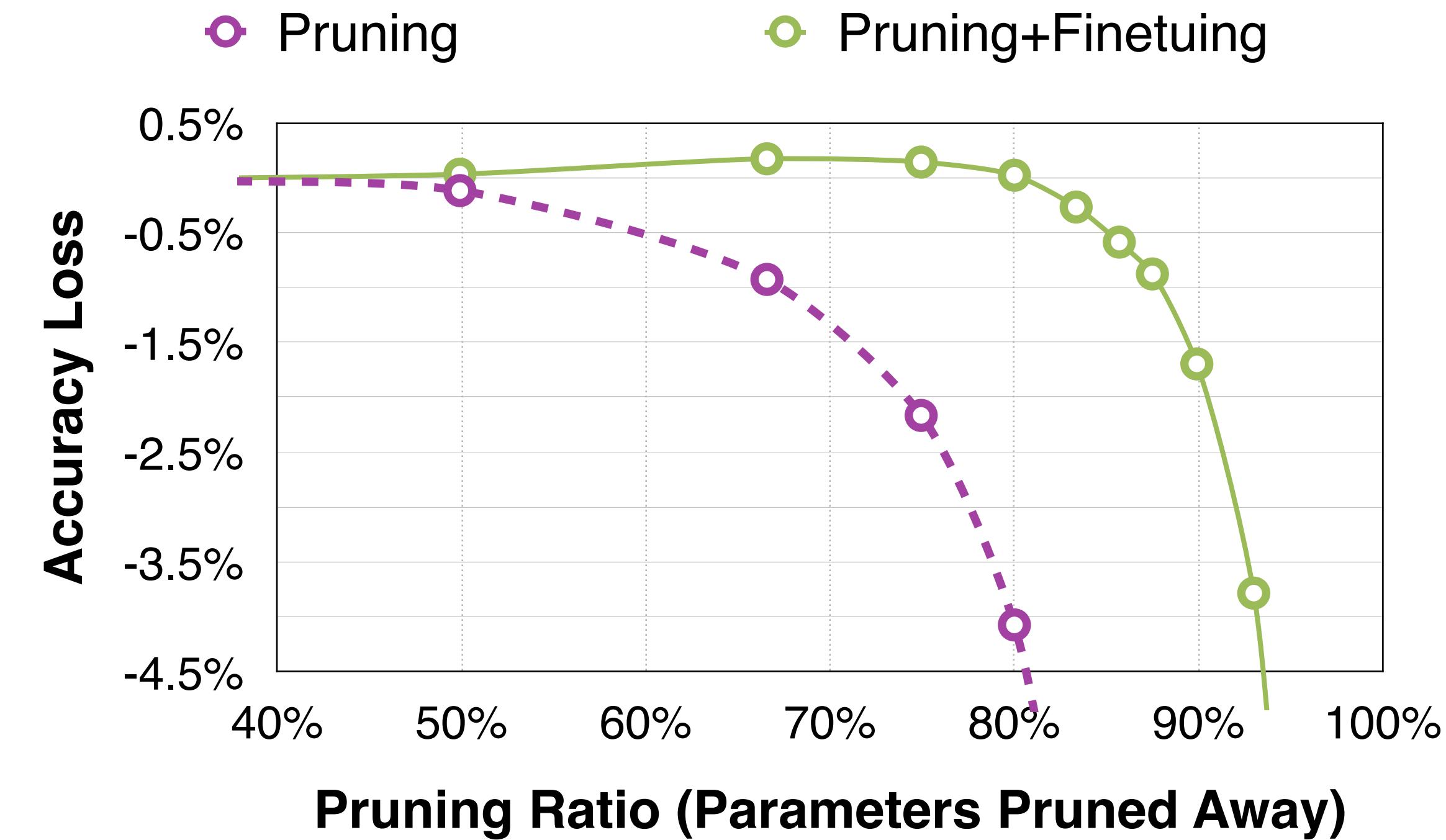
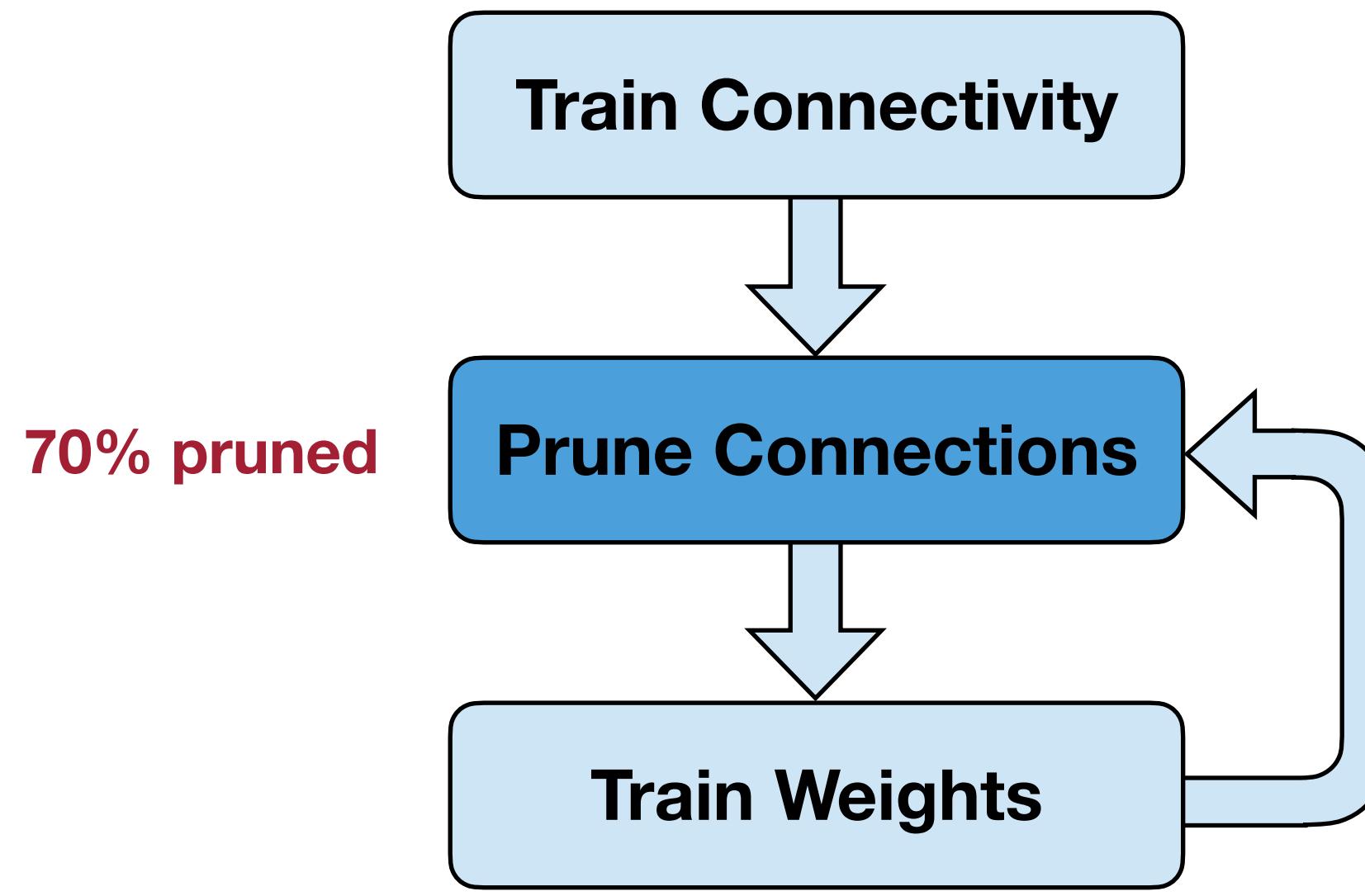
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

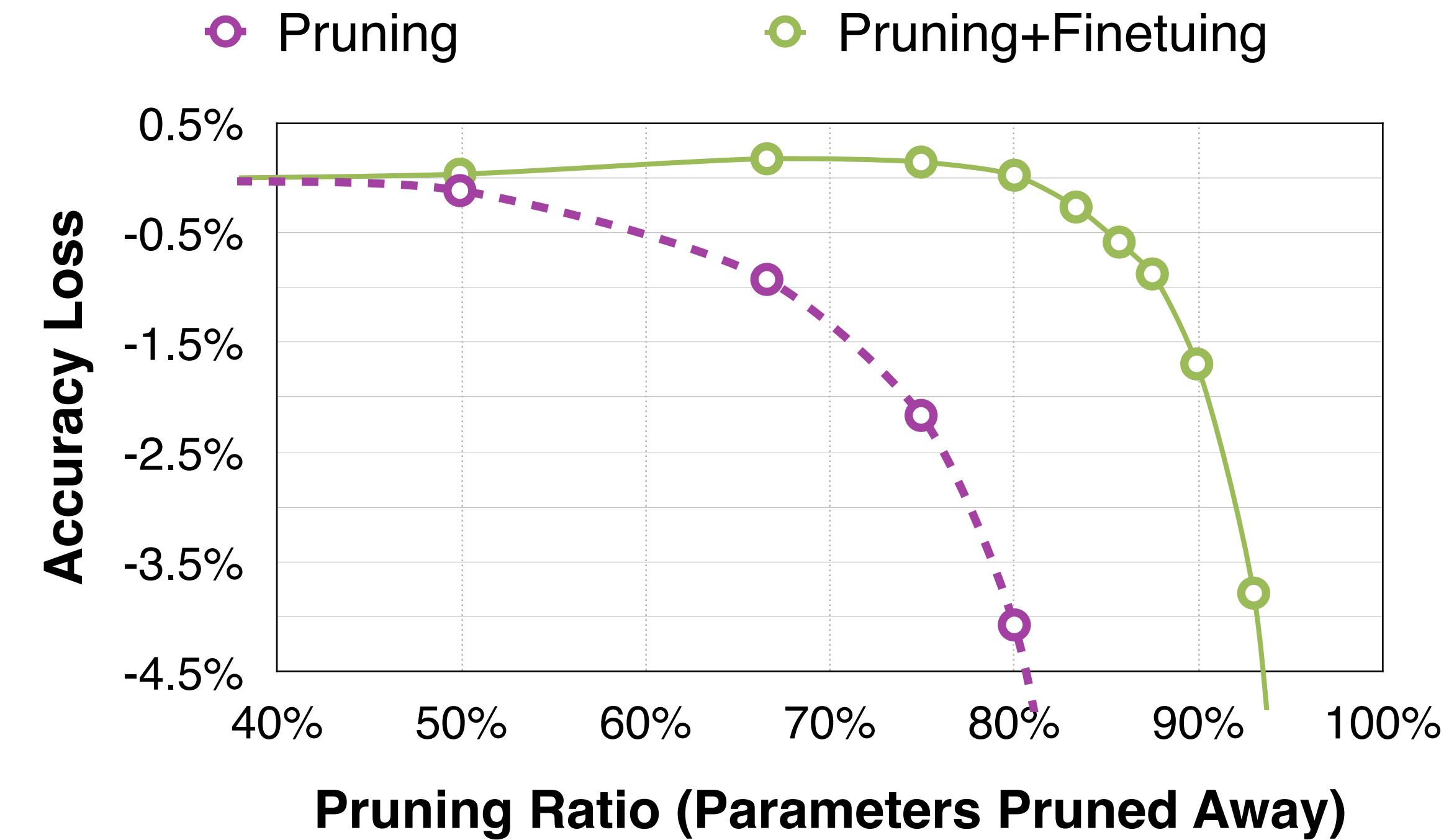
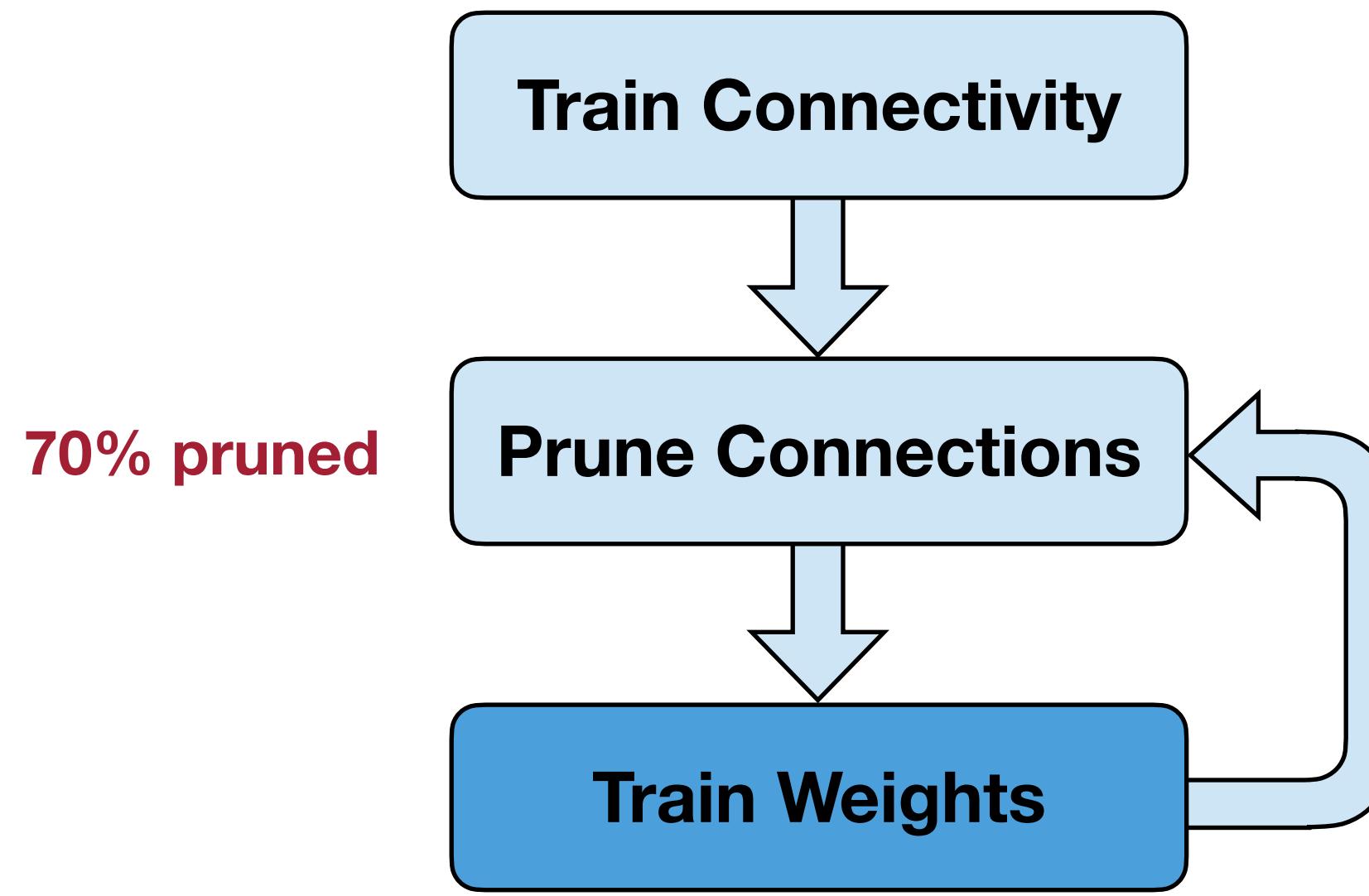
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

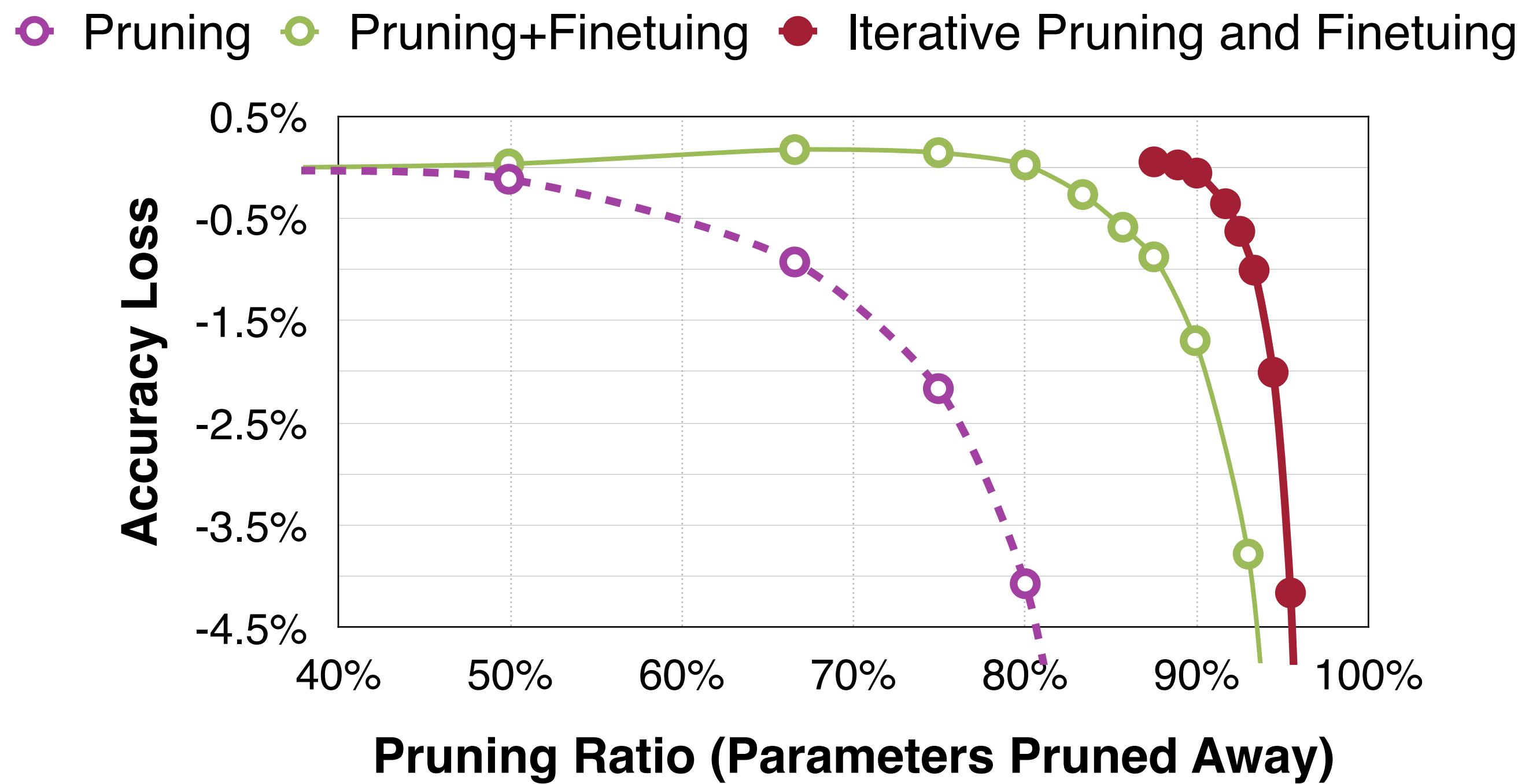
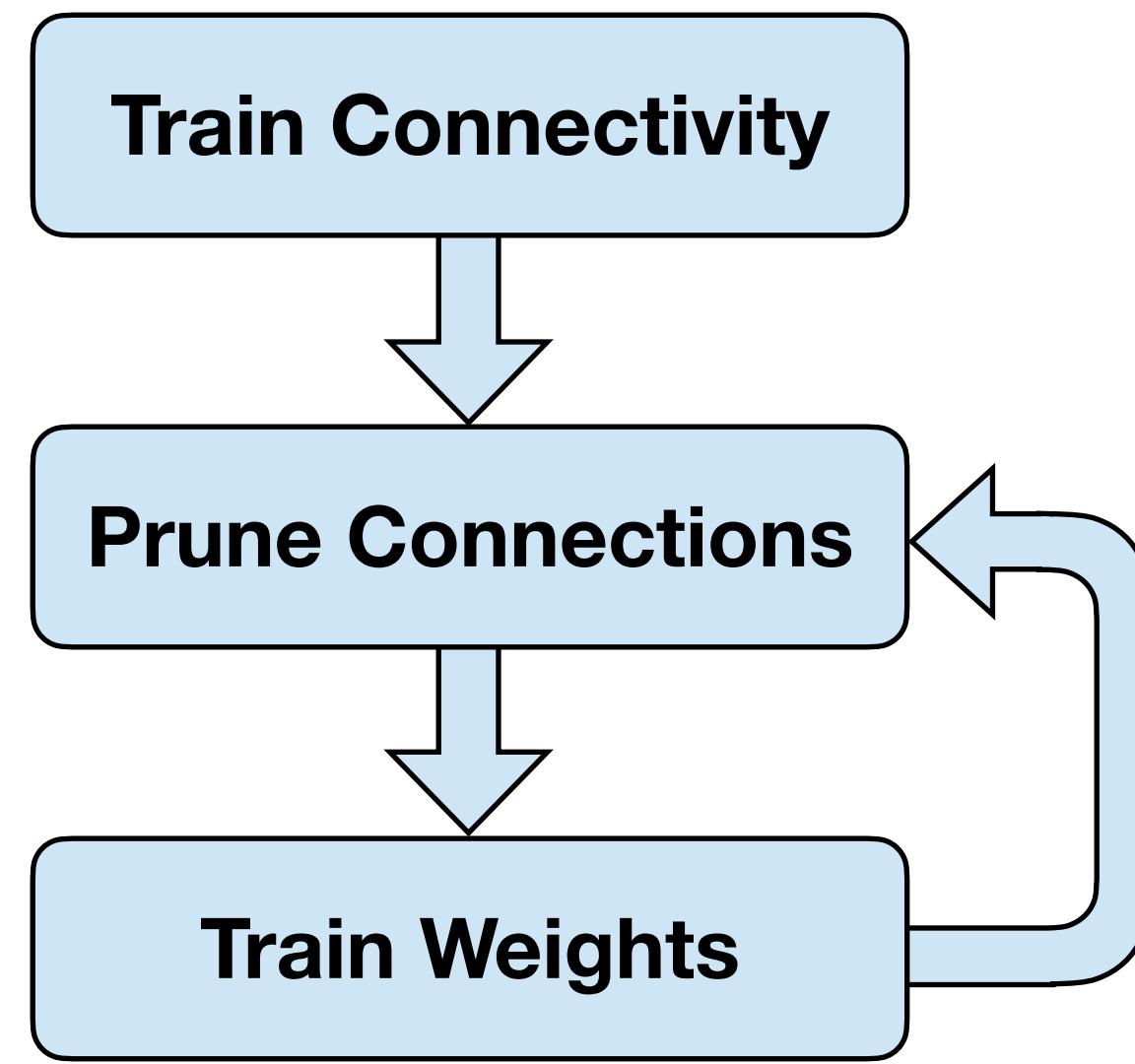
- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Iterative Pruning

- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.
 - boost pruning ratio from 5× to 9× on AlexNet compared to single-step aggressive pruning.



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Regularization

- When training neural networks or fine-tuning quantized neural networks, regularization is added to the loss term to
 - ***penalize non-zero parameters***
 - ***encourage smaller parameters***
- The most common regularization for improving performance of pruning is L1/L2 regularization.
 - L1-Regularization

$$L' = L(\mathbf{x}; \mathbf{W}) + \lambda |\mathbf{W}|$$

- L2-Regularization

$$L' = L(\mathbf{x}; \mathbf{W}) + \lambda \|\mathbf{W}\|^2$$

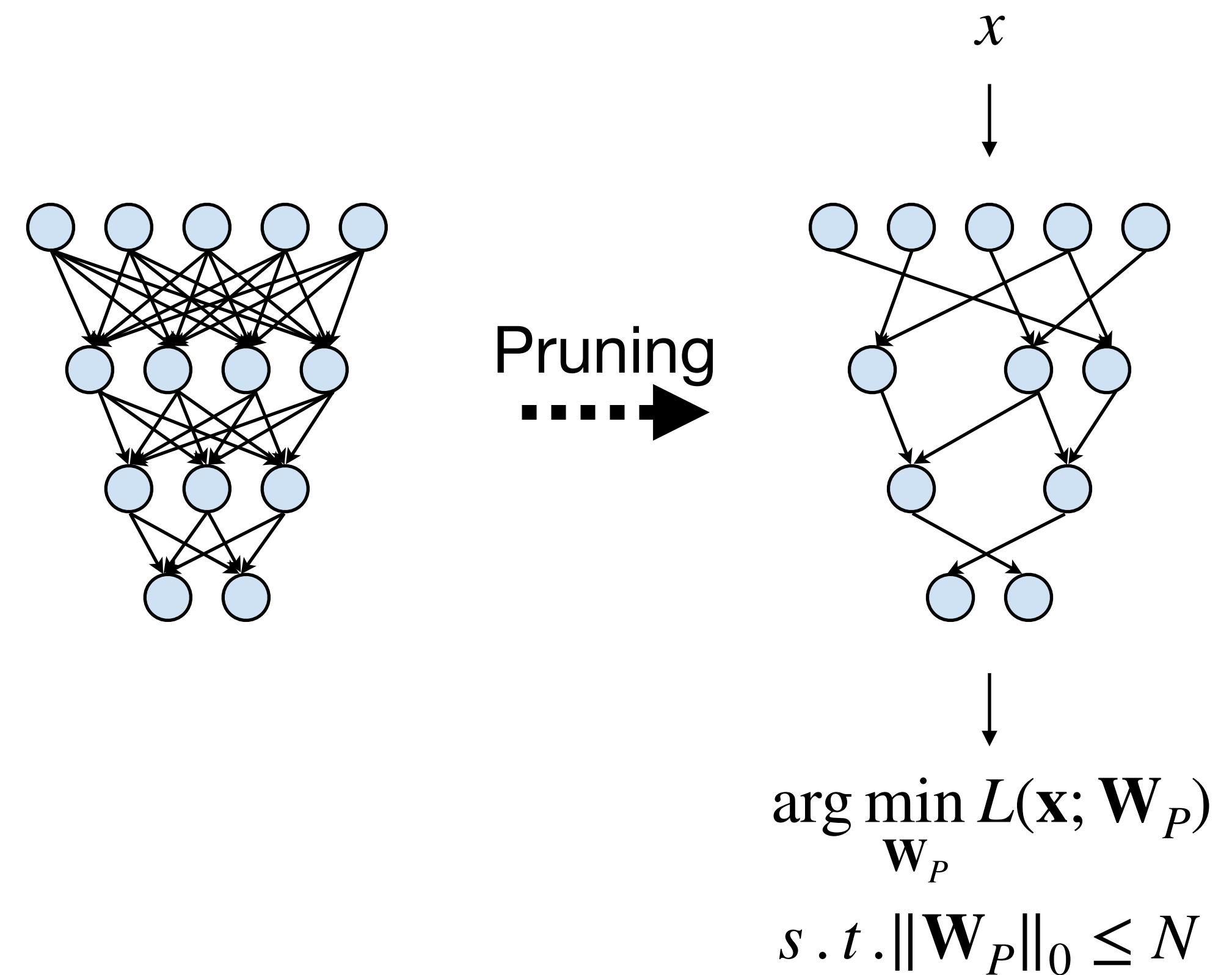
- **Examples:**
 - Magnitude-based Fine-grained Pruning applies L2 regularization on weights
 - Network Slimming applies smooth-L1 regularization on channel scaling factors.

Learning Efficient Convolutional Networks through Network Slimming [Liu et al., ICCV 2017]

Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

Neural Network Pruning

- **Introduction to Pruning**
 - What is pruning?
 - How should we formulate pruning?
- **Determine the Pruning Granularity**
 - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
 - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
 - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
 - How should we improve performance of pruned models?



Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]

System Support for Sparsity

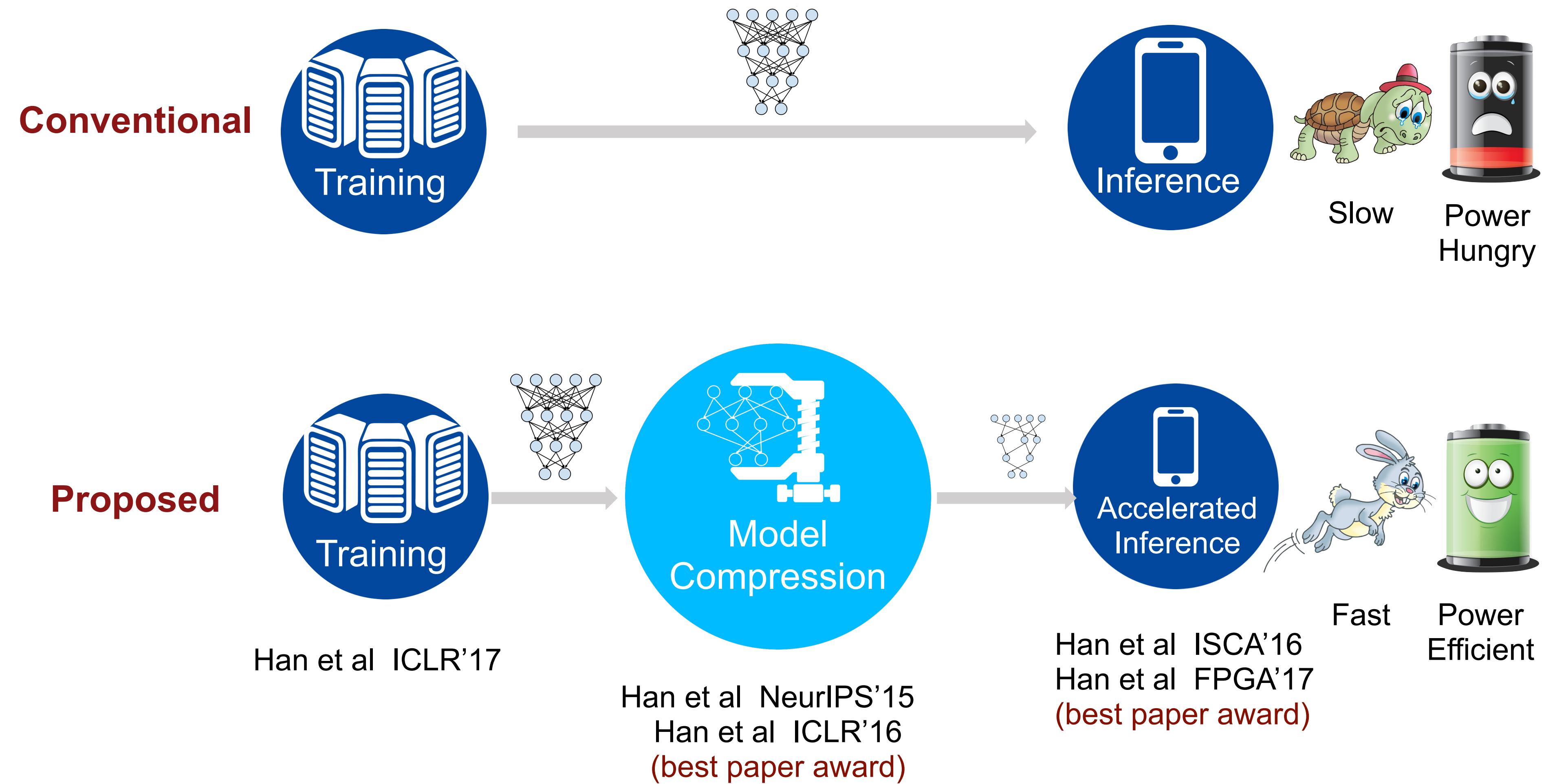
System & Hardware Support for Sparsity

- EIE: Weight Sparsity + Activation Sparsity
- NVIDIA Tensor Core: M:N Weight Sparsity
- TorchSparse & PointAcc: Activation Sparsity

System & Hardware Support for Sparsity

- **EIE: Weight Sparsity + Activation Sparsity**
- NVIDIA Tensor Core: M:N Weight Sparsity
- TorchSparse & PointAcc: Activation Sparsity

Proposed Paradigm



EIE: Efficient Inference Engine

The First DNN Accelerator for Sparse, Compressed Model

$$0 * A = 0$$

Sparse Weight

90% *static* sparsity

$$W * 0 = 0$$

Sparse Activation

70% *dynamic* sparsity

$$\cancel{2.09}, \cancel{1.92} \Rightarrow 2$$

Weight Sharing

4-bit weights

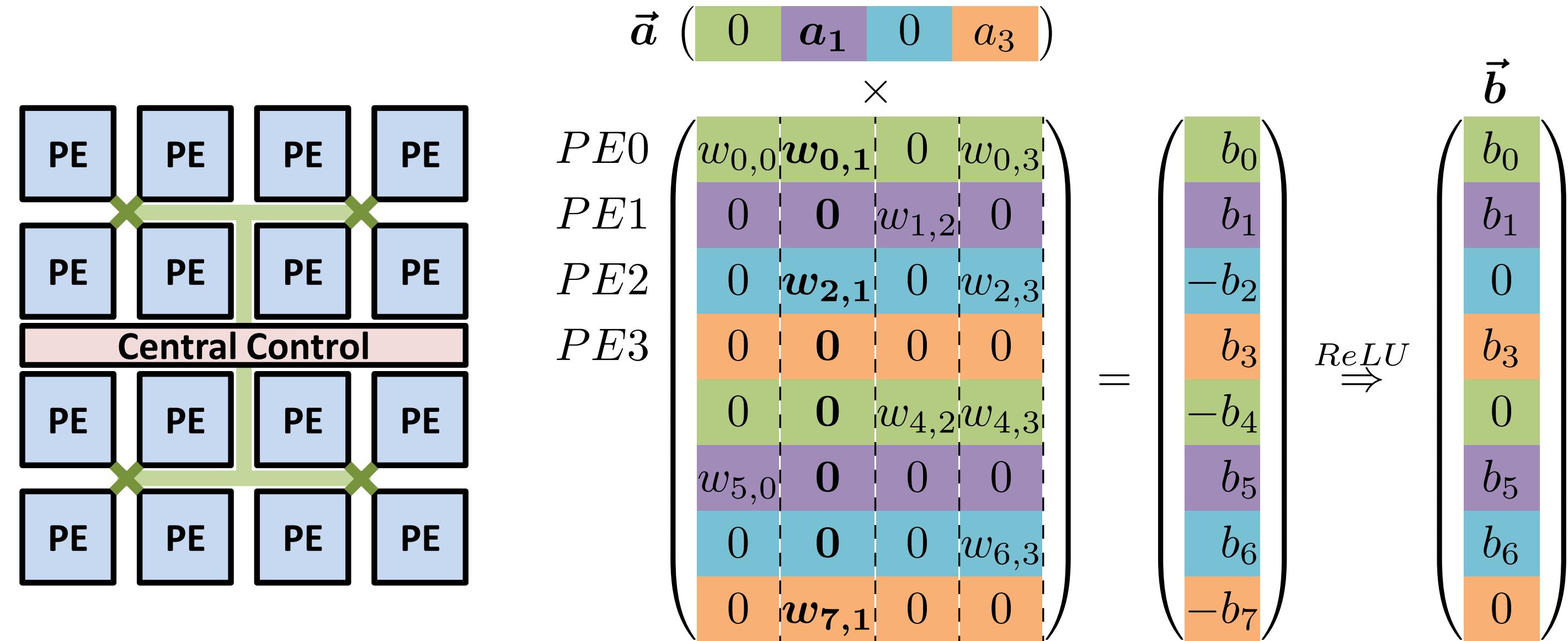
 10x less computation

 5x less memory footprint

 3x less computation

 8x less memory footprint

EIE: Parallelization on Sparsity



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

EIE: Parallelization on Sparsity

logically

$$\vec{a} \left(\begin{array}{cccc} 0 & a_1 & 0 & a_3 \end{array} \right) \times \begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \end{array} \left(\begin{array}{cccc} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_{2,1} & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{array} \right) = \left(\begin{array}{c} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{array} \right) \xrightarrow{\text{ReLU}} \left(\begin{array}{c} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{array} \right)$$

physically

Virtual Weight	$W_{0,0}$	$W_{0,1}$	$W_{4,2}$	$W_{0,3}$	$W_{4,3}$
Relative Index	0	1	2	0	0
Column Pointer	0	1	2	3	5

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Dataflow

$$\vec{a} \begin{pmatrix} 0 & a_1 & 0 & a_3 \end{pmatrix} \times \begin{matrix} PE0 \\ PE1 \\ PE2 \\ PE3 \end{matrix} \begin{pmatrix} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_{2,1} & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \xrightarrow{\text{ReLU}} \vec{b} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

rule of thumb:
 $0 * A = 0 \quad W * 0 = 0$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Dataflow

$$\vec{a} \left(\begin{array}{cccc} 0 & a_1 & 0 & a_3 \end{array} \right) \times \begin{matrix} PE0 \\ PE1 \\ PE2 \\ PE3 \end{matrix} \left(\begin{array}{cccc} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_{2,1} & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{array} \right) = \left(\begin{array}{c} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{array} \right) \xrightarrow{\text{ReLU}} \vec{b} \left(\begin{array}{c} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{array} \right)$$

rule of thumb:
 $0 * A = 0 \quad W * 0 = 0$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Dataflow

$$\vec{a} \left(\begin{array}{cccc} 0 & a_1 & 0 & a_3 \end{array} \right) \times \begin{array}{c} \\ \times \\ \end{array} \begin{array}{c} PE0 \\ PE1 \\ PE2 \\ PE3 \end{array} \left(\begin{array}{cccc} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_{2,1} & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{array} \right) = \left(\begin{array}{c} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{array} \right) \xrightarrow{\text{ReLU}} \vec{b} \left(\begin{array}{c} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{array} \right)$$

rule of thumb:
 $0 * A = 0 \quad W * 0 = 0$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

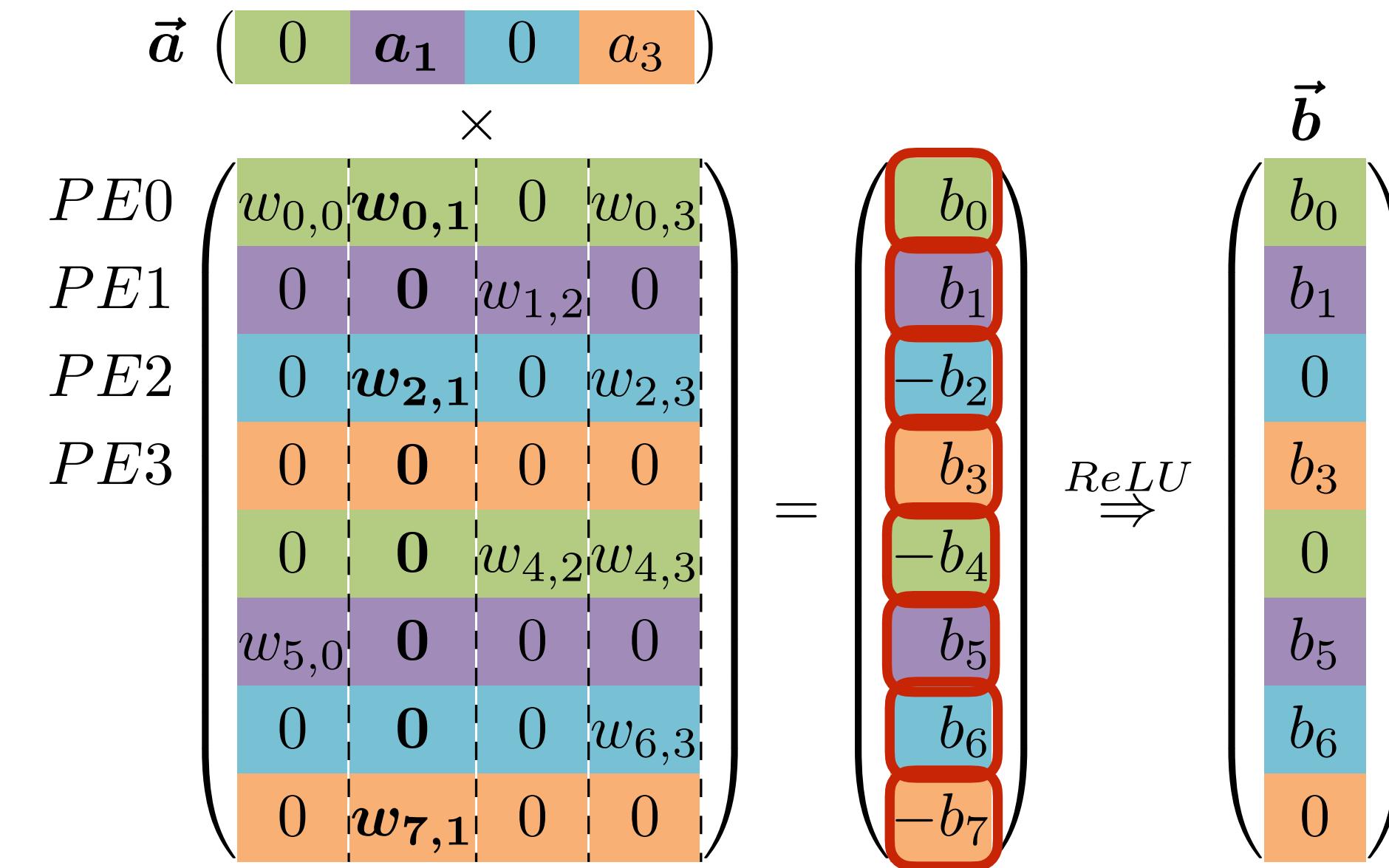
Dataflow

$$\vec{a} \begin{pmatrix} 0 & a_1 & 0 & a_3 \end{pmatrix} \times \begin{matrix} PE0 \\ PE1 \\ PE2 \\ PE3 \end{matrix} \begin{pmatrix} w_{0,0} & w_{0,1} & 0 & w_{0,3} \\ 0 & 0 & w_{1,2} & 0 \\ 0 & w_{2,1} & 0 & w_{2,3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & w_{4,2} & w_{4,3} \\ w_{5,0} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{6,3} \\ 0 & w_{7,1} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \xrightarrow{\text{ReLU}} \vec{b} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

rule of thumb:
 $0 * A = 0 \quad W * 0 = 0$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

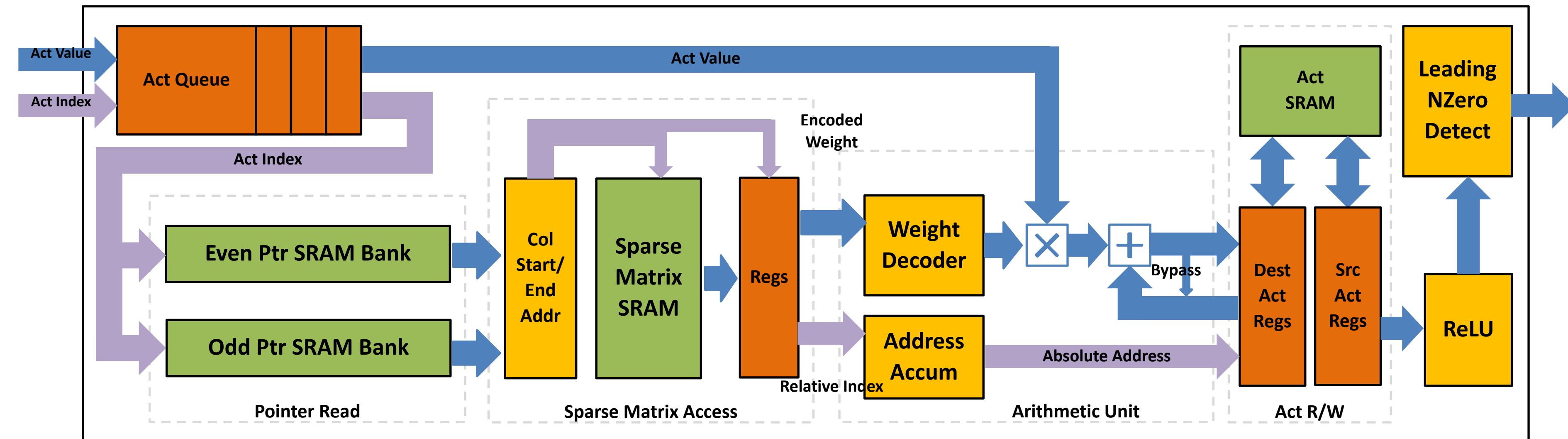
Dataflow



rule of thumb:
 $0 * A = 0 \quad W * 0 = 0$

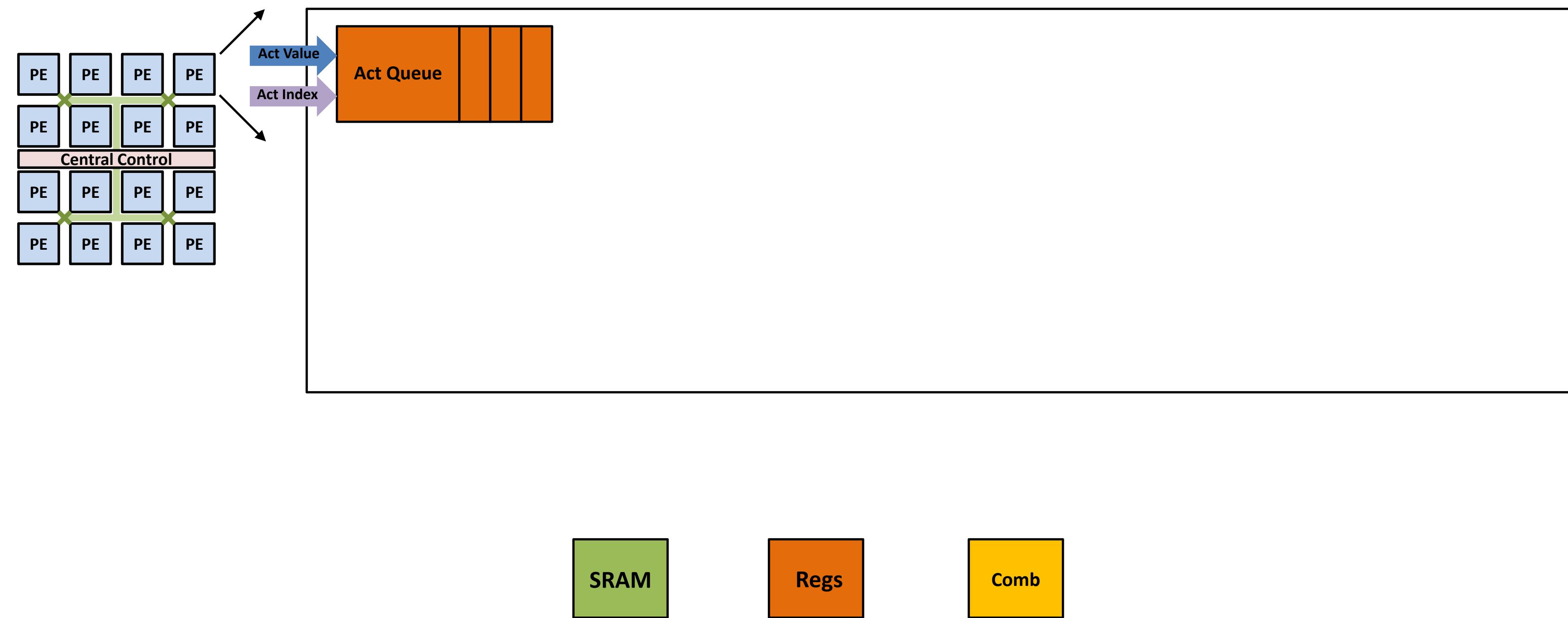
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Micro Architecture for each PE



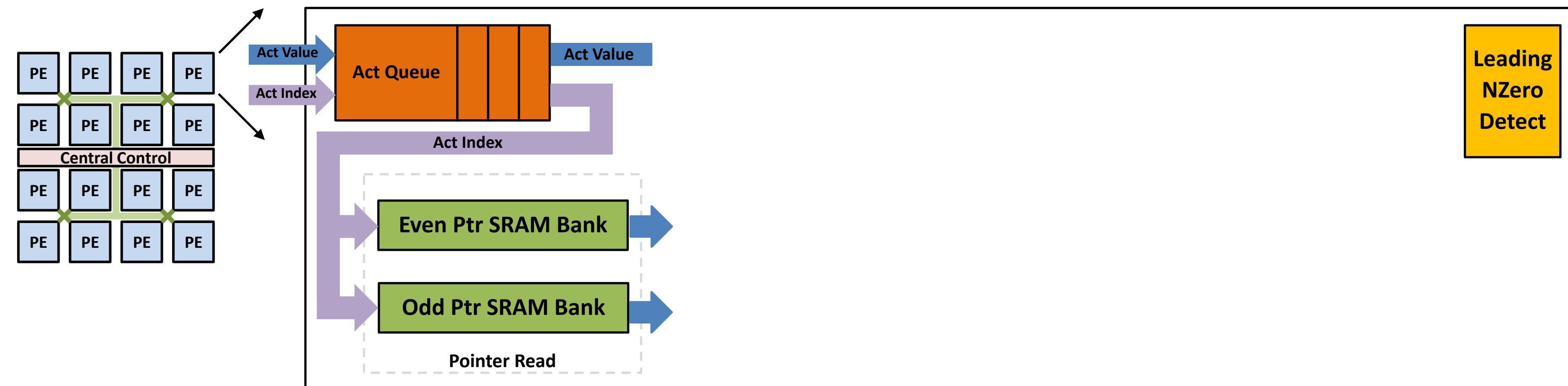
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Load Balance



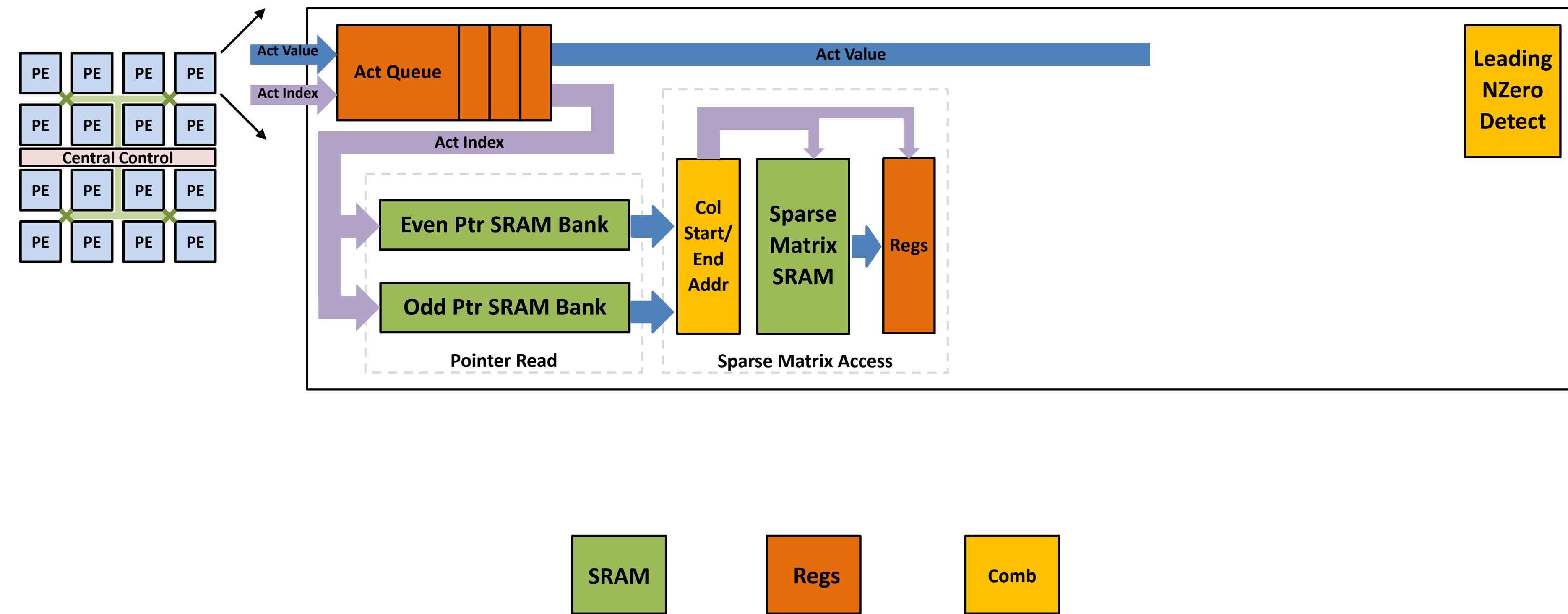
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Activation Sparsity



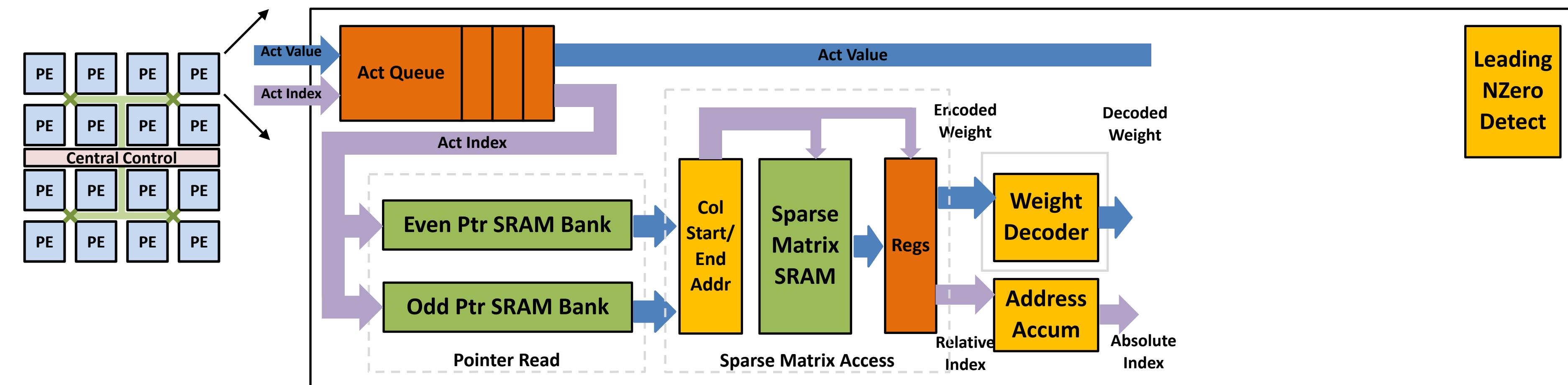
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Weight Sparsity



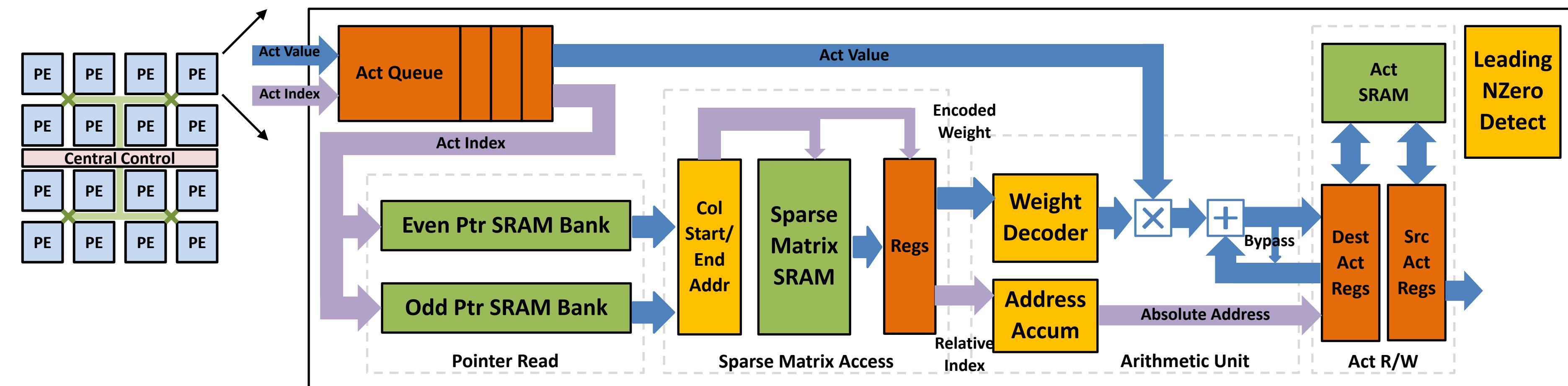
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Weight Sharing



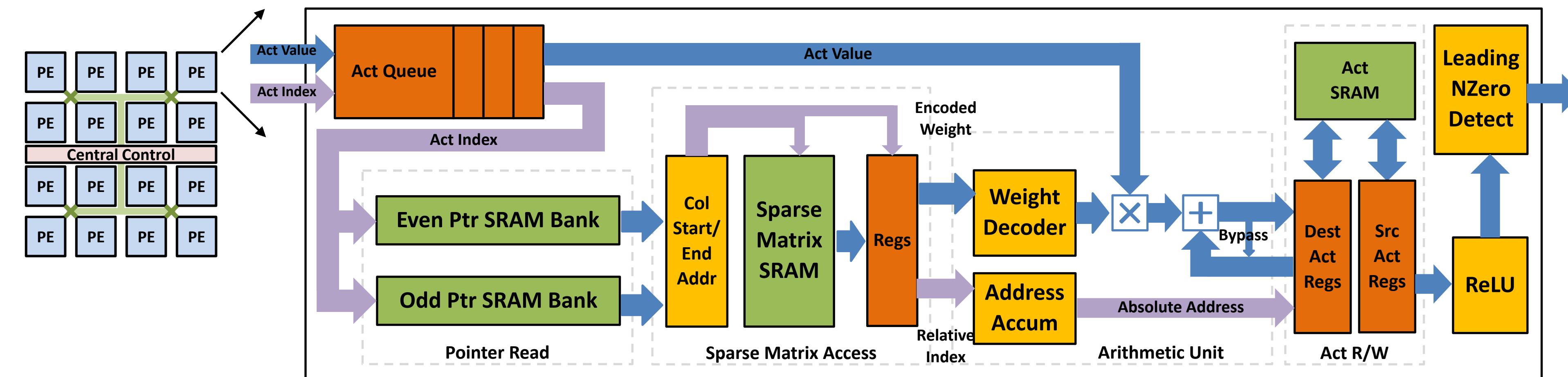
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Arithmetic & Write Back



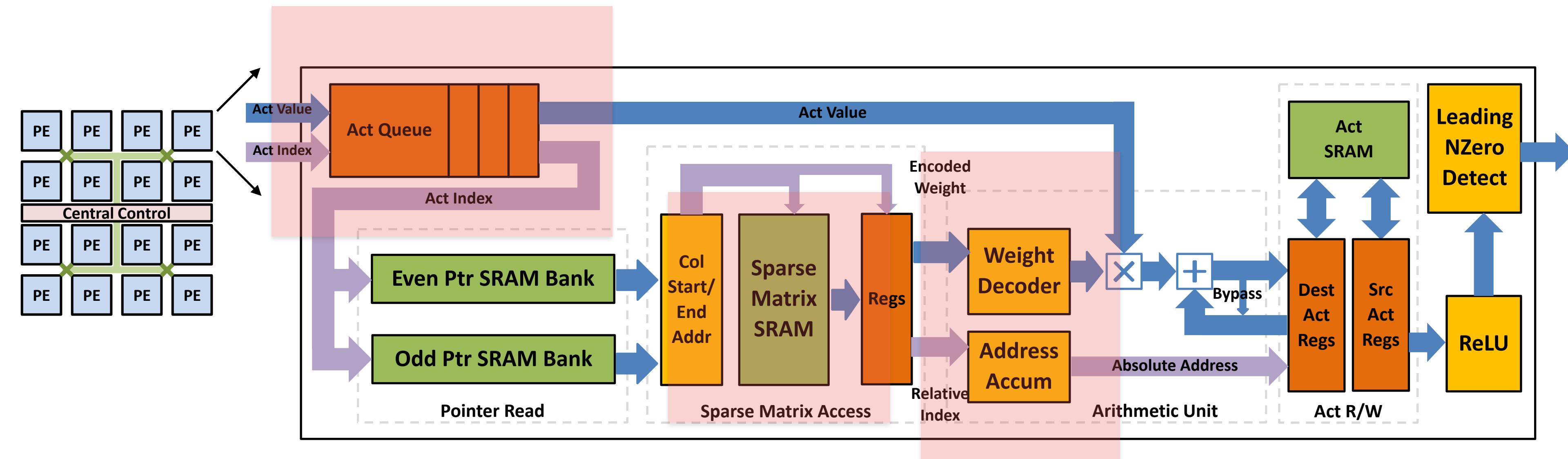
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Relu, Non-zero Detection



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

What's Special



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

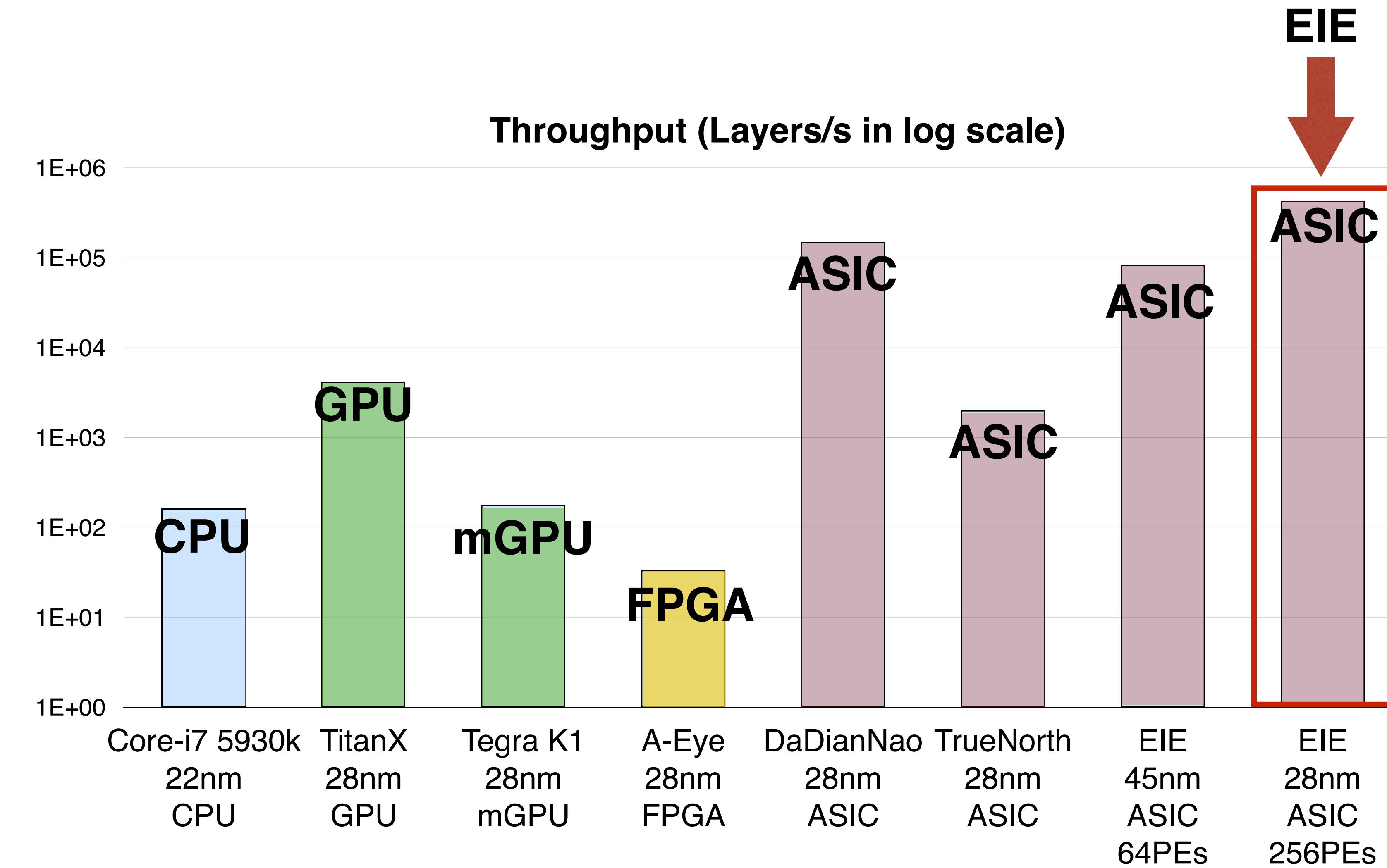
Benchmark

- CPU: Intel Core-i7 5930k
- GPU: NVIDIA TitanX
- Mobile GPU: NVIDIA Jetson TK1

Layer	Size	Weight Density	Activation Density	FLOP Reduction	Description
AlexNet-6	4096×9216	9%	35%	33x	AlexNet for image classification
AlexNet-7	4096×4096	9%	35%	33x	
AlexNet-8	1000×4096	25%	38%	10x	
VGG-6	4096×25088	4%	18%	100x	VGG-16 for image classification
VGG-7	4096×4096	4%	37%	50x	
VGG-8	1000×4096	23%	41%	10x	
NeuralTalk-We	600×4096	10%	100%	10x	RNN and LSTM for image caption
NeuralTalk-Wd	8791×600	11%	100%	10x	
NeuralTalk-LSTM	2400×1201	10%	100%	10x	

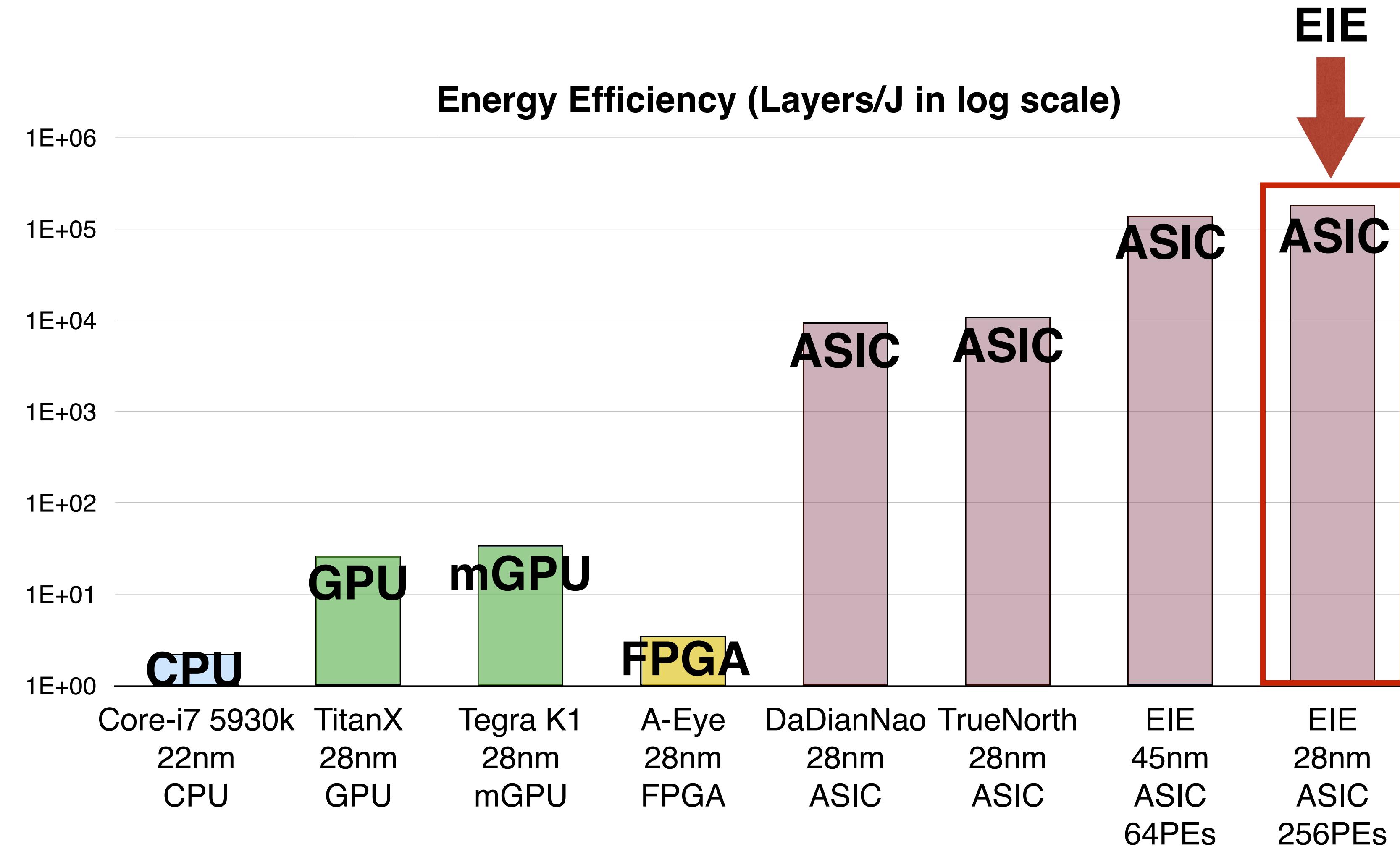
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Comparison: Throughput



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Comparison: Energy Efficiency



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]

Top-5 most cited papers in 50 years of ISCA

Rank	Citations	Year	Title (★ means it won the <i>ISCA Influential Paper Award</i>)	First Author + HOF Authors	Type	Topic
1	5351	1995	The SPLASH-2 programs: Characterization and methodological considerations	Stephen Woo , Anoop Gupta	Tool	Benchmark
2	4214	2017	In-datacenter performance analysis of a Tensor Processing Unit	Norm Jouppi , David Patterson	Arch	Machine Learning
3	3834	2000	★ Wattpc: A framework for architectural-level power analysis and optimizations	David Brooks , Margaret Martonosi	Tool	Power
4	3386	1993	★ Transactional memory: Architectural support for lock-free data structures	Maurice Herlihy	Micro	Parallelism
5	2690	2016	EIE: Efficient inference engine on compressed deep neural network	Song Han , Bill Dally , Mark Horowitz	Arch	Machine Learning

Retrospective: EIE: Efficient Inference Engine on Sparse and Compressed Neural Network

Song Han^{1,3}, Xingyu Liu⁴, Huizi Mao³, Jing Pu⁵, Ardavan Pedram^{2,6}, Mark A. Horowitz², William J. Dally^{2,3},

¹MIT ²Stanford ³ NVIDIA ⁴ CMU ⁵ Google ⁶ Samsung

Pro:

- EIE demonstrated that special-purpose hardware can make it cost-effective to do sparse operations with matrices that are up to 50% dense
- EIE exploits both weight sparsity and activation sparsity, not only saves energy by skipping zero weights, but also saves the cycle by not computing it.
- EIE supports fine-grained sparsity, and allows pruning to achieve a higher pruning ratio.
- Aggressive weight quantization (4bit) to save memory footprint. To maintain accuracy, EIE decodes the weight to 16bit and uses 16bit arithmetic. W4A16 approach is reborn in LLM: GPTQ, AWQ, llama.cpp, MLC LLM

Con:

- EIE isn't as easily applied to arrays of vector processors — improve: structured sparsity (N:M sparsity)
- EIE's Control flow overhead, storage overhead — improve: coarse grain sparsity
- EIE only support FC layers - actually reborn in LLM
- EIE fits everything on SRAM - practical for TinyML, not LLM

Retrospective: EIE: Efficient Inference Engine on Sparse and Compressed Neural Network

Song Han^{1,3}, Xingyu Liu⁴, Huizi Mao³, Jing Pu⁵, Ardavan Pedram^{2,6}, Mark A. Horowitz², William J. Dally^{2,3},

¹MIT ²Stanford ³ NVIDIA ⁴ CMU ⁵ Google ⁶ Samsung

The first principle of efficient AI computing is to be lazy: avoid redundant computation, quickly reject the work, or delay the work.

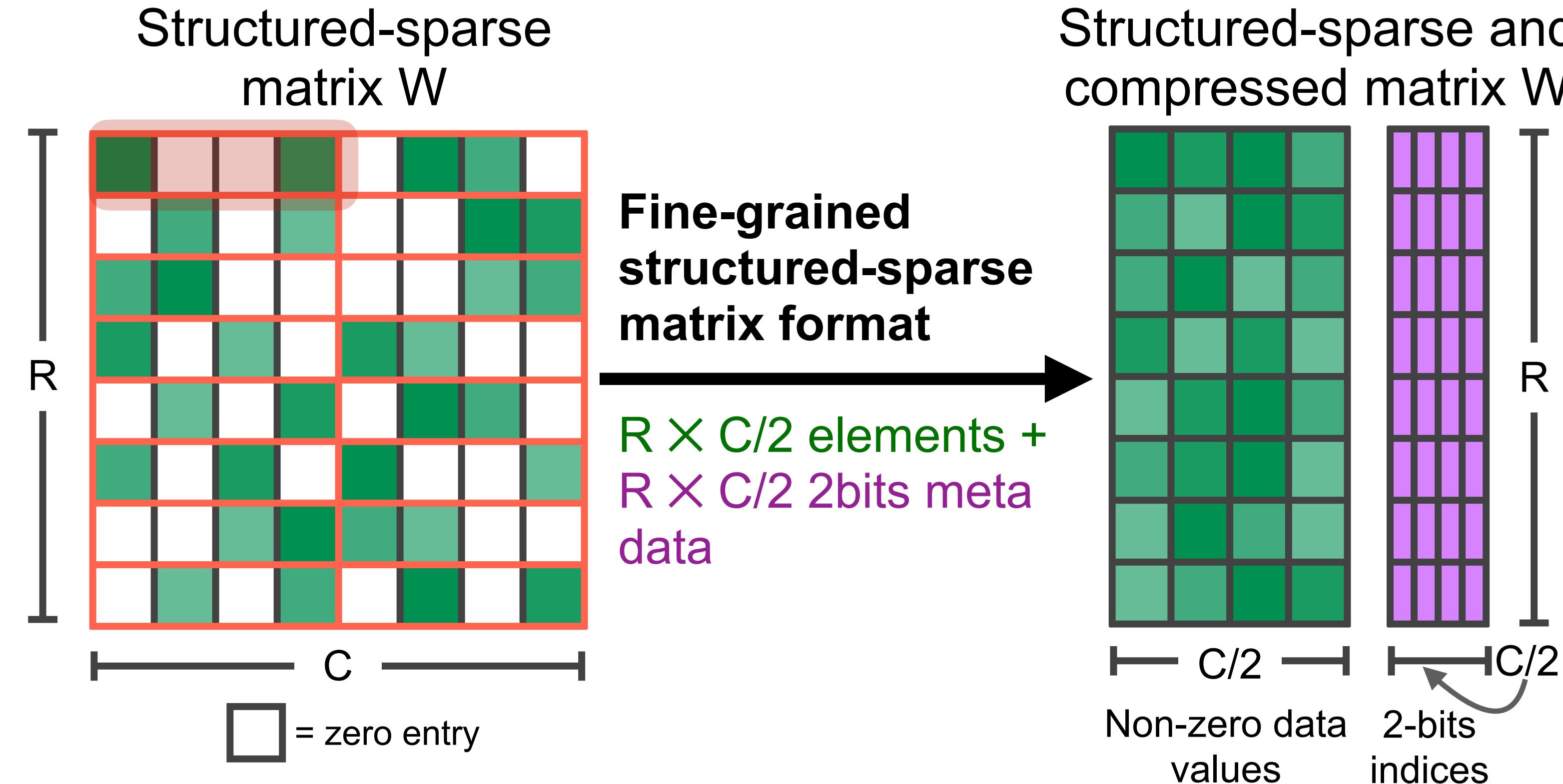
- Generative AI: spatial sparsity [SIGE, NeurIPS'22]
- Transformer: token sparsity, progressive quantization [SpAtten, HPCA'21]
- Video: temporal sparsity [TSM, ICCV'19]
- Point cloud: spatial sparsity [TorchSparse, MLSys'22 & PointAcc, Micro'22]

We envision future AI models will be sparse at various granularity and structures. Co-designed with specialized accelerators, sparse models will become more efficient and accessible.

System & Hardware Support for Sparsity

- EIE: Weight Sparsity + Activation Sparsity for GEMM
- **NVIDIA Tensor Core: M:N Weight Sparsity**
- TorchSparse & PointAcc: Activation Sparsity

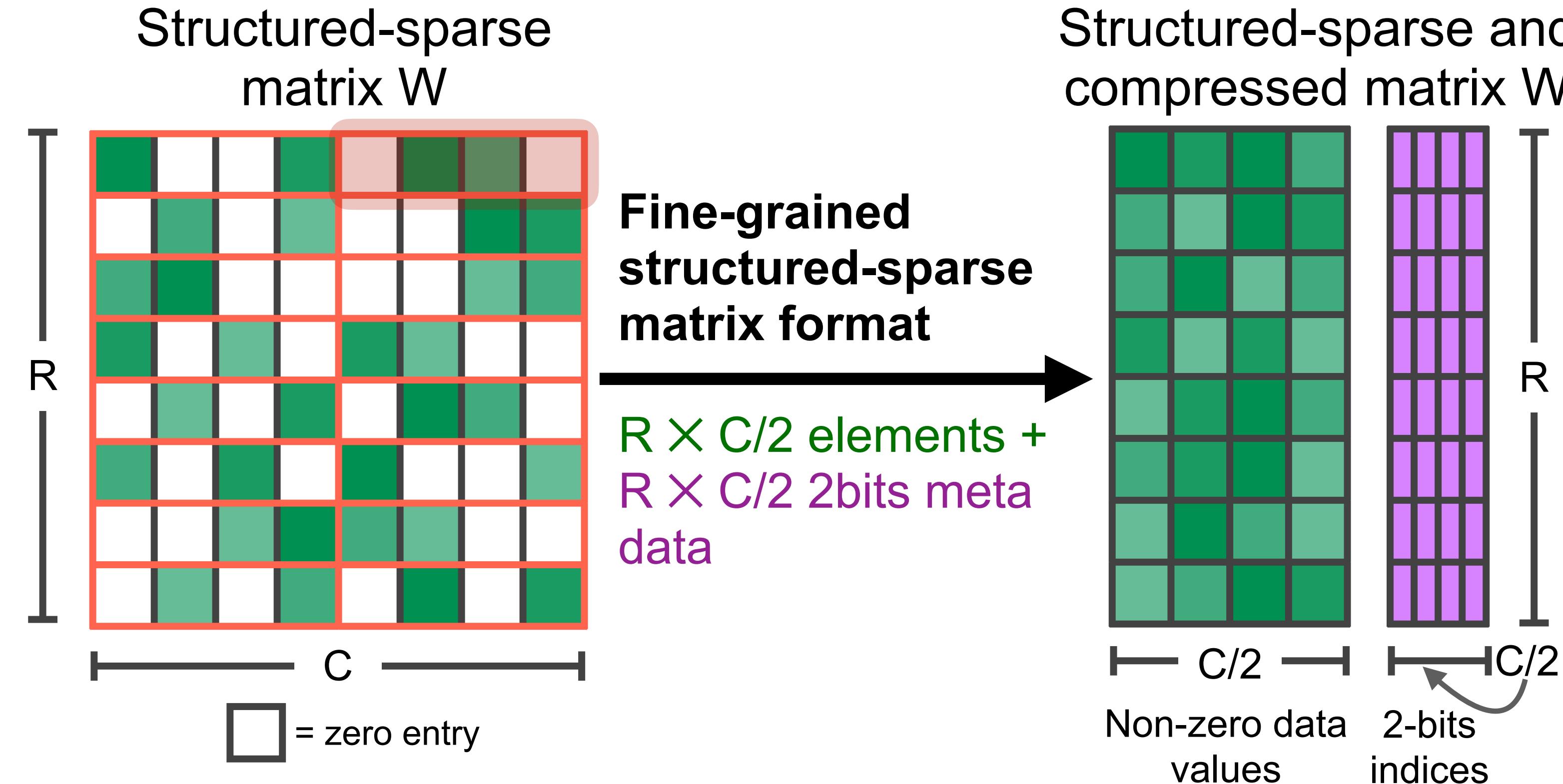
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

[Accelerating Sparse Deep Neural Networks \[Mishra et al., arXiv 2021\]](#)

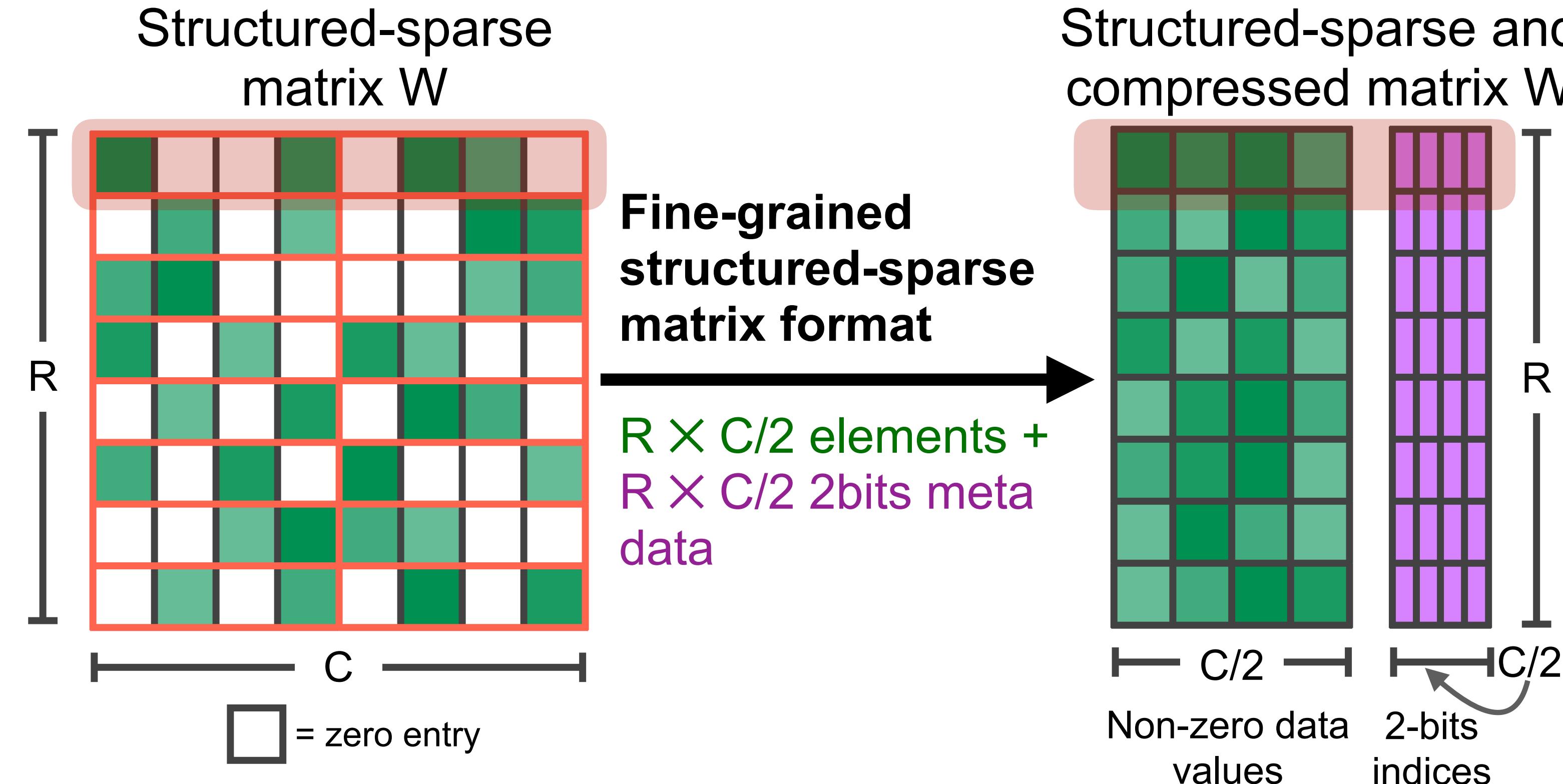
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

[Accelerating Sparse Deep Neural Networks \[Mishra et al., arXiv 2021\]](#)

M:N Sparsity

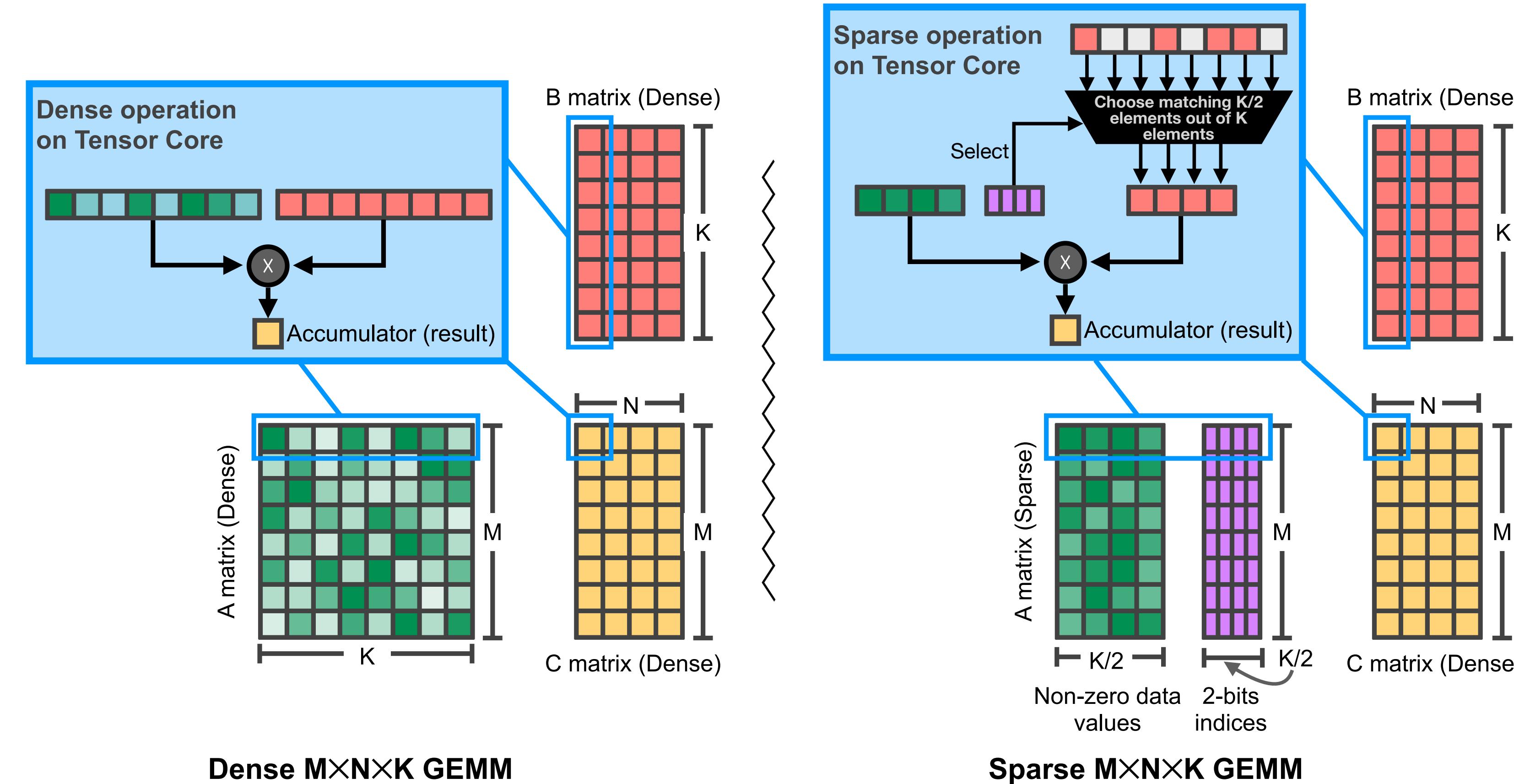


Push all the nonzero elements to the left in memory: save storage and computation.

Accelerating Sparse Deep Neural Networks [Mishra et al., arXiv 2021]

System Support for M:N Sparsity

Mapping M:N sparse matrices onto NVIDIA tensor cores



The indices are used to mask out the inputs. Only 2 multiplications will be done out of four.

Accelerating Sparse Deep Neural Networks [Mishra et al., arXiv 2021]

System Support for M:N Sparsity

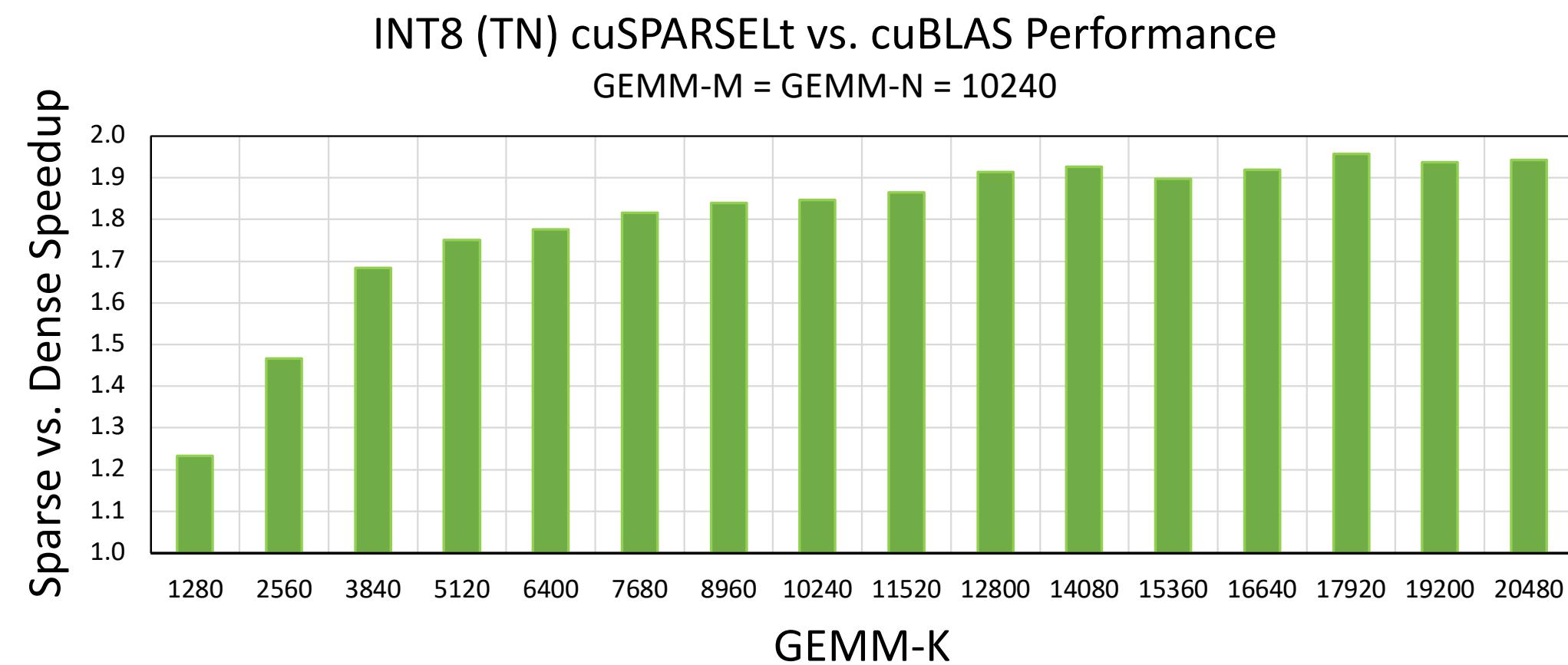


Fig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cores. Larger GEMMs achieve nearly a 2 \times speedup with Sparse Tensor Cores.

Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	78.8	78.8	78.9
Wide ResNet-50	78.5	78.6	78.5
Wide ResNet-101	78.9	79.2	79.1
Inception v3	77.1	77.1	77.1
Xception	79.2	79.2	79.2
VGG-11	70.9	70.9	70.8
VGG-16	74.0	74.1	74.1
VGG-19	75.0	75.0	75.0
SUNet-128	75.6	76.0	75.4
SUNet-7-128	76.4	76.5	76.3
DRN26	75.2	75.3	75.3
DRN-105	79.4	79.5	79.4

Pruning CNNs with 2:4 sparsity will bring about large speedup for GEMM workloads and it will not incur performance drop for DNN models.

Accelerating Sparse Deep Neural Networks [Mishra et al., arXiv 2021]

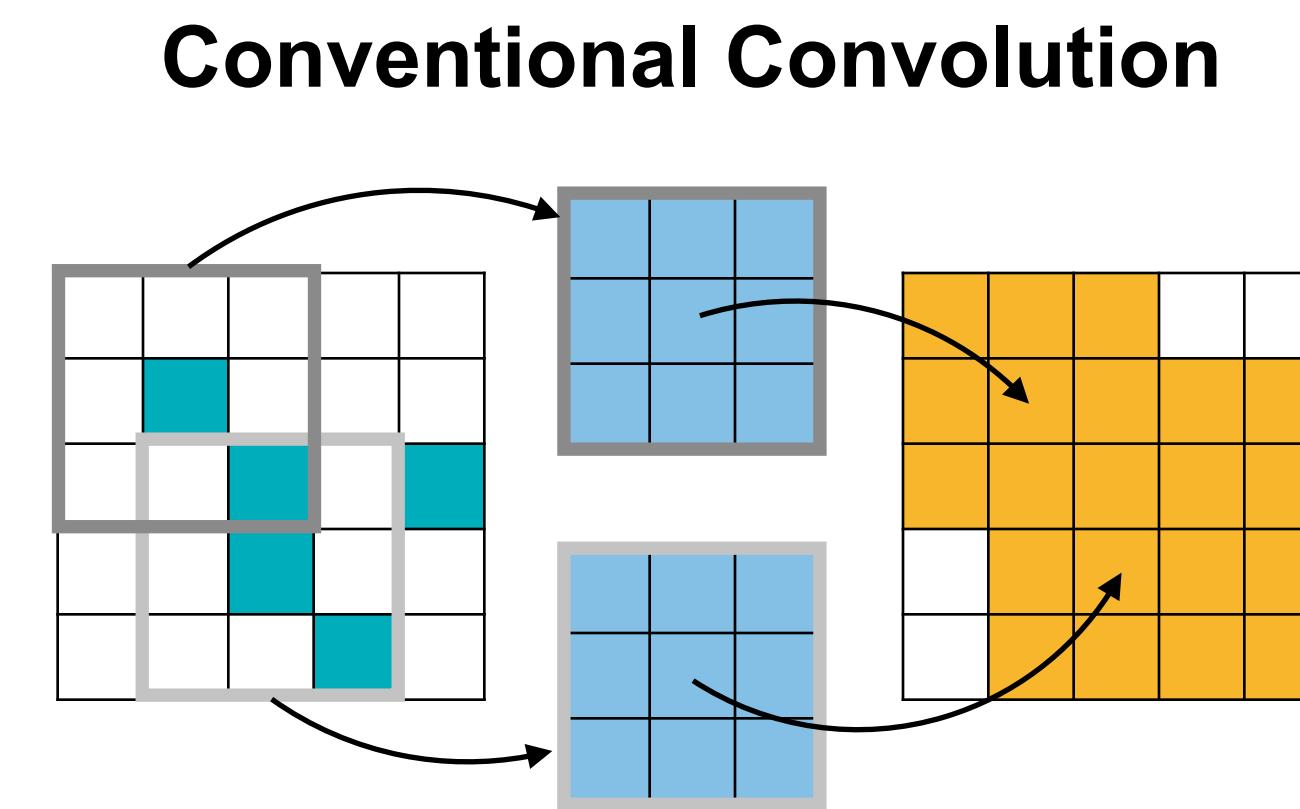
System & Hardware Support for Sparsity

- EIE: Weight Sparsity + Activation Sparsity
- NVIDIA Tensor Core: M:N Weight Sparsity
- **TorchSparse & PointAcc: Activation Sparsity**

System & Hardware Support for Sparsity

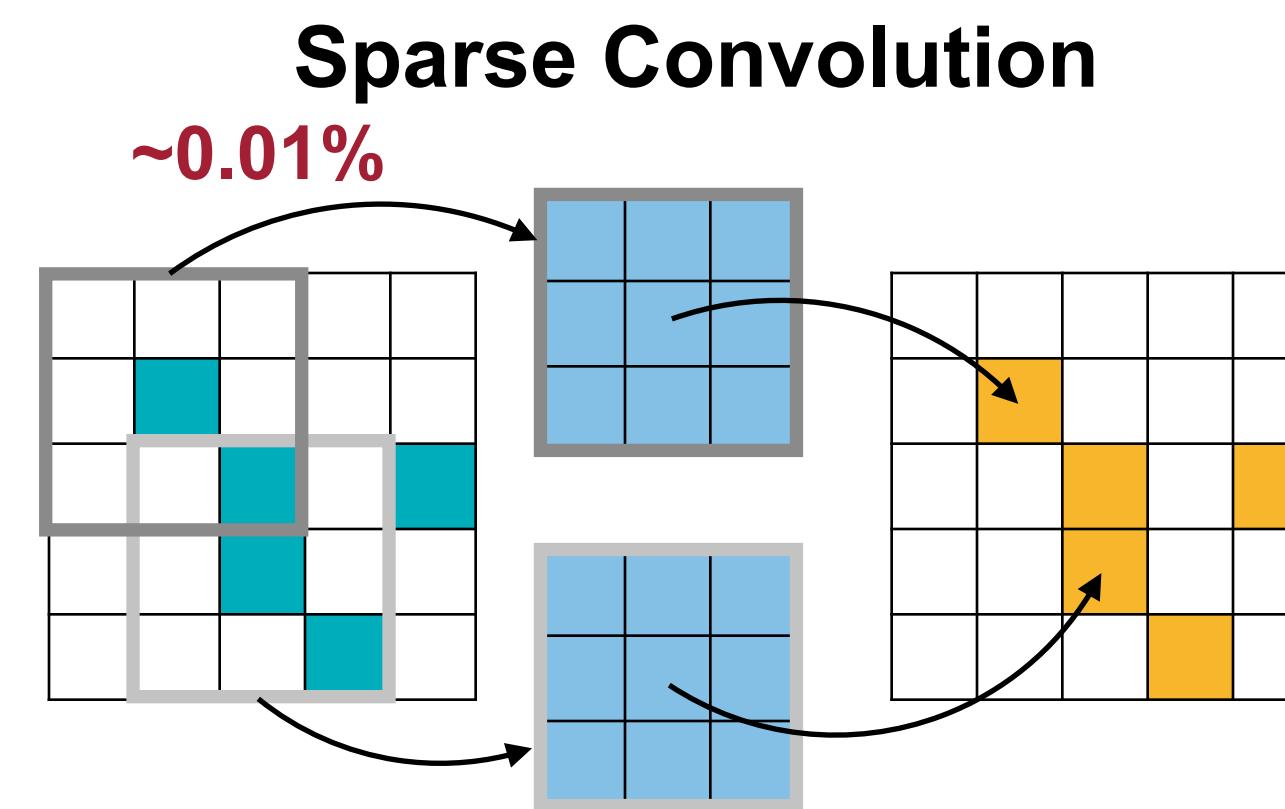
- EIE: Weight Sparsity + Activation Sparsity
- NVIDIA Tensor Core: M:N Weight Sparsity
- **TorchSparse & PointAcc: Activation Sparsity**
 - **TorchSparse: Sparse Convolution Library**
 - **PointAcc: Hardware Accelerator for Sparse Convolution**

Sparse convolution on sparse inputs



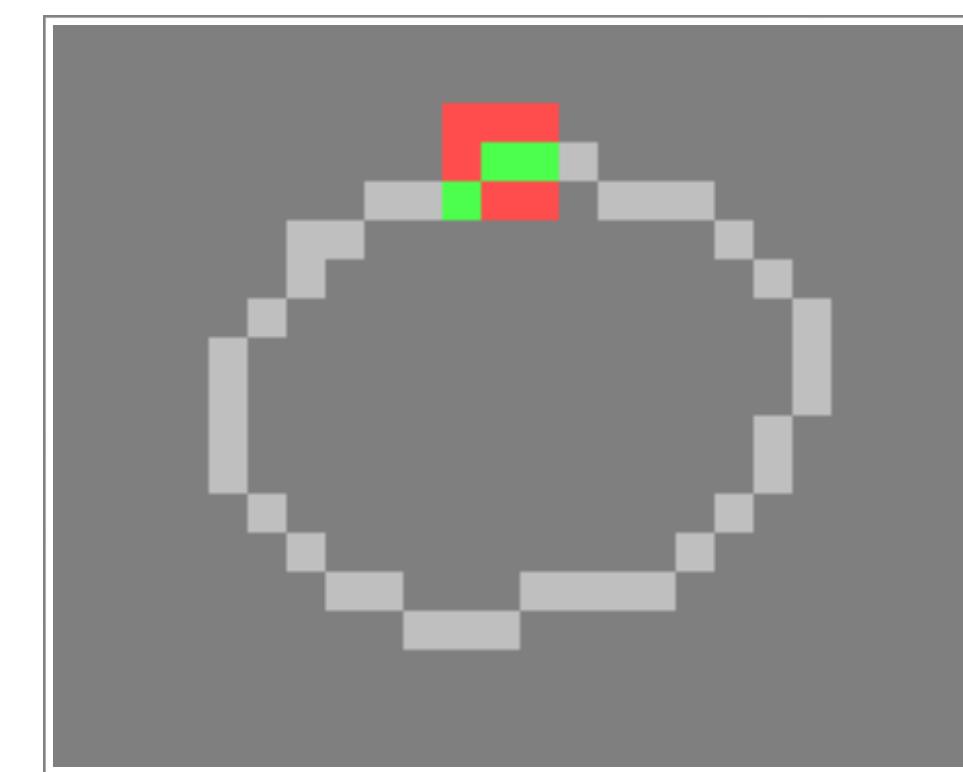
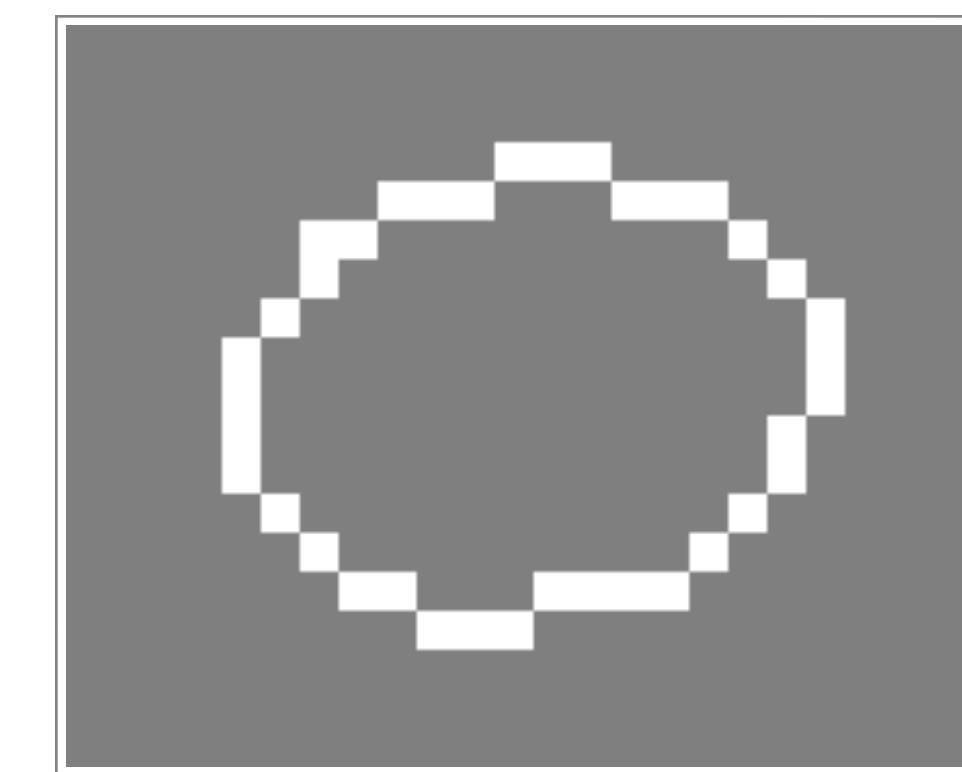
Input sparsity
from ReLU

Nonzeros
will dilate



Input sparsity from
the distribution in
physical space

Nonzeros
will not
dilate

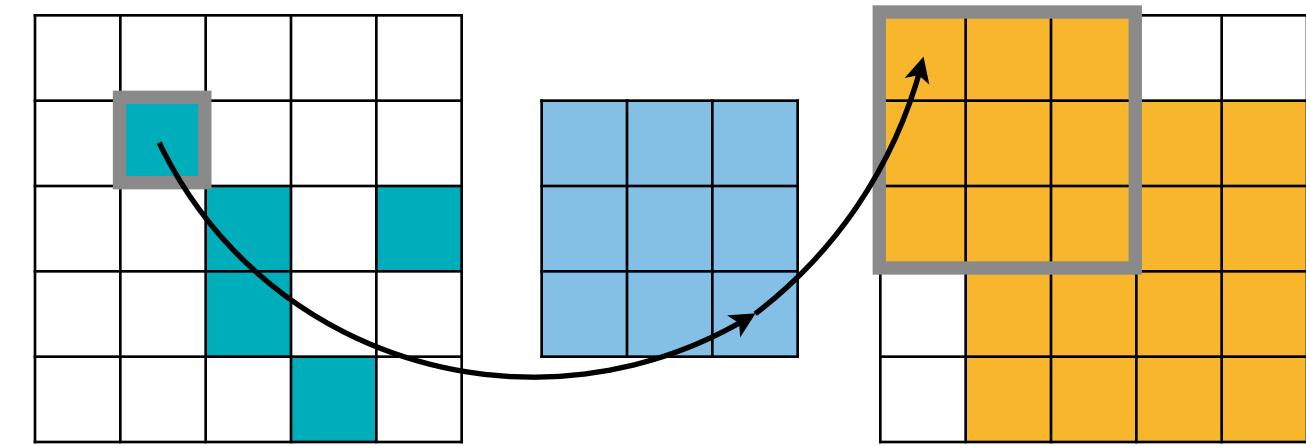


Submanifold Sparse Convolutional Neural Networks [Graham, BMVC 2015]

Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



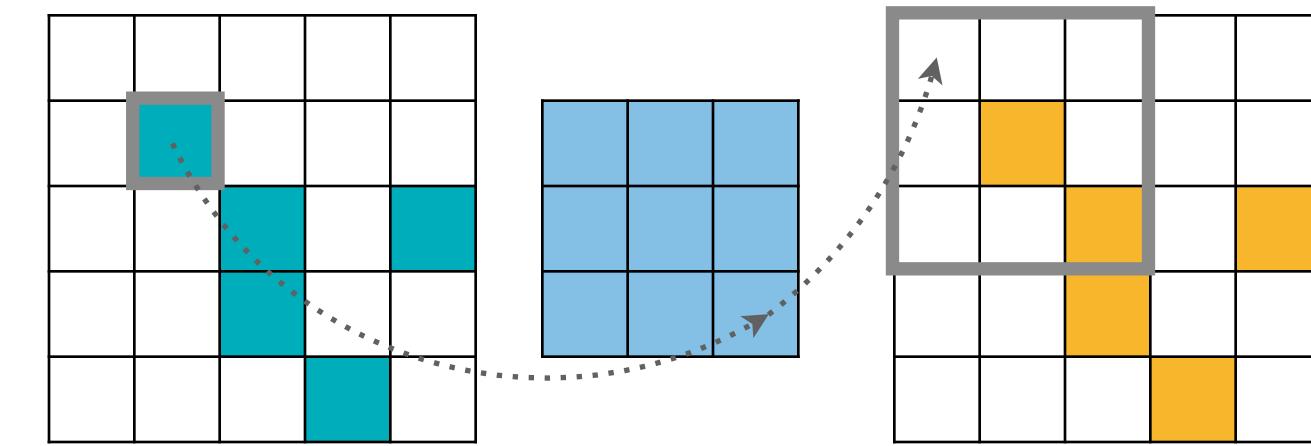
$$(\mathbf{P}_0, \mathbf{Q}_0, \mathbf{W}_{1,1})$$

Maps
(\mathbf{In} , \mathbf{Out} , \mathbf{Wgt})

Computation

$(\mathbf{f}_{\text{out}} = \mathbf{f}_{\text{out}} + \mathbf{f}_{\text{in}} \times \mathbf{W}_{\text{wgt}})$ for each entry in the maps

Sparse Convolution

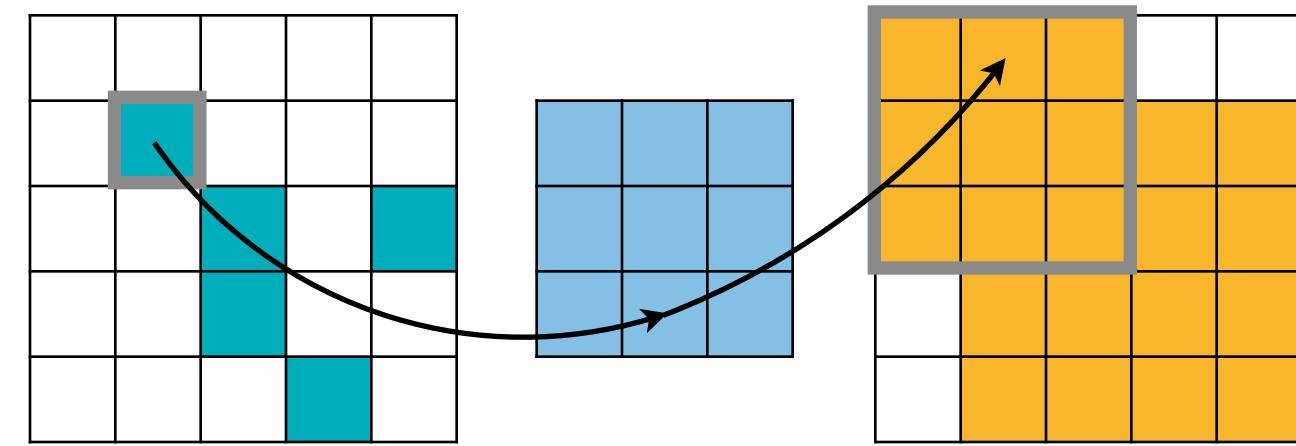


No compute

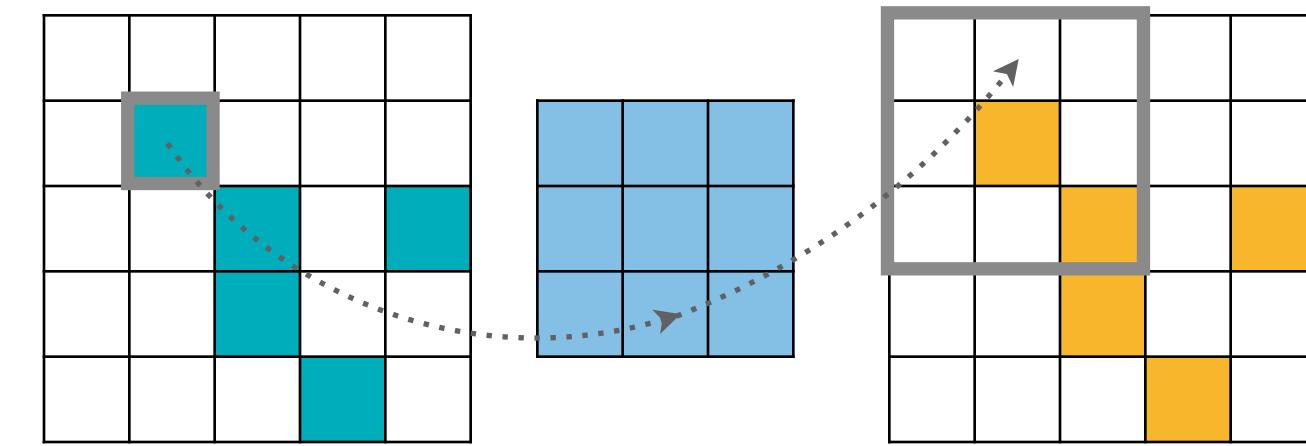
Sparse convolution computation

A **sparse** set of **dense** MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



$$\begin{aligned} & (\mathbf{P}_0, \mathbf{Q}_0, \mathbf{W}_{1,1}) \\ & (\mathbf{P}_0, \mathbf{Q}_1, \mathbf{W}_{1,0}) \end{aligned}$$

Maps
(\mathbf{In} , \mathbf{Out} , \mathbf{Wgt})

Computation

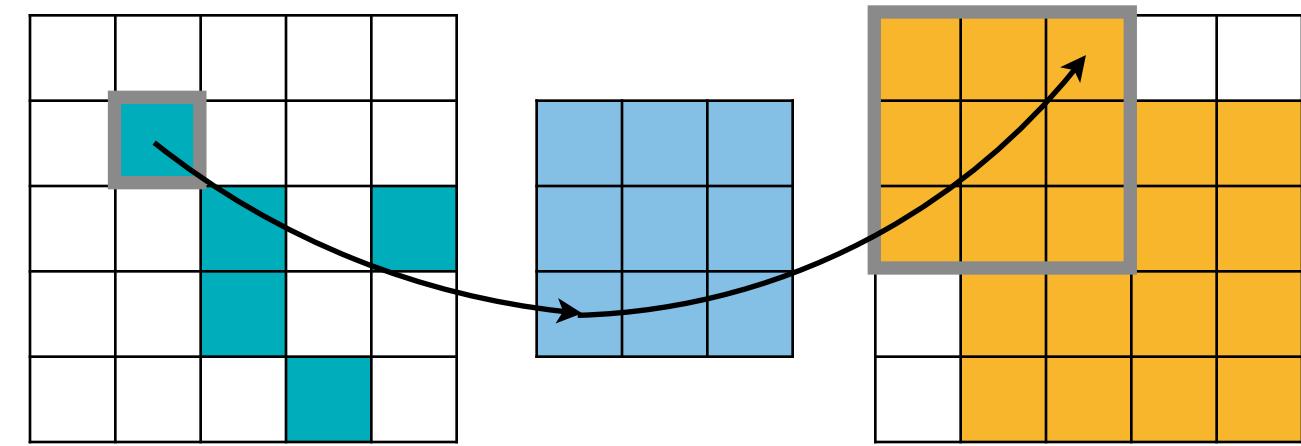
$(\mathbf{f}_{\text{out}} = \mathbf{f}_{\text{out}} + \mathbf{f}_{\text{in}} \times \mathbf{W}_{\text{Wgt}})$ for each entry in the maps

No compute
No compute

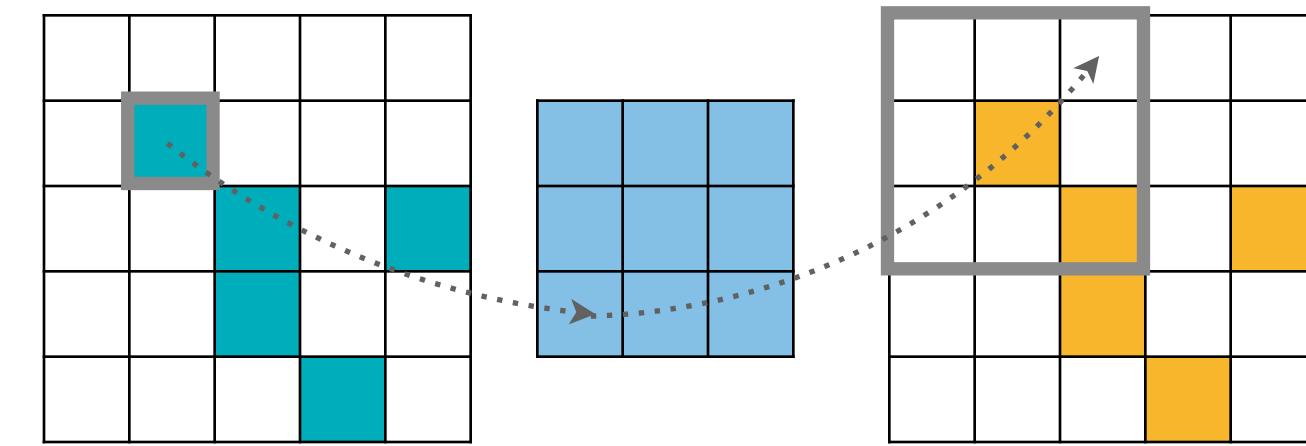
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$

No compute
No compute
No compute

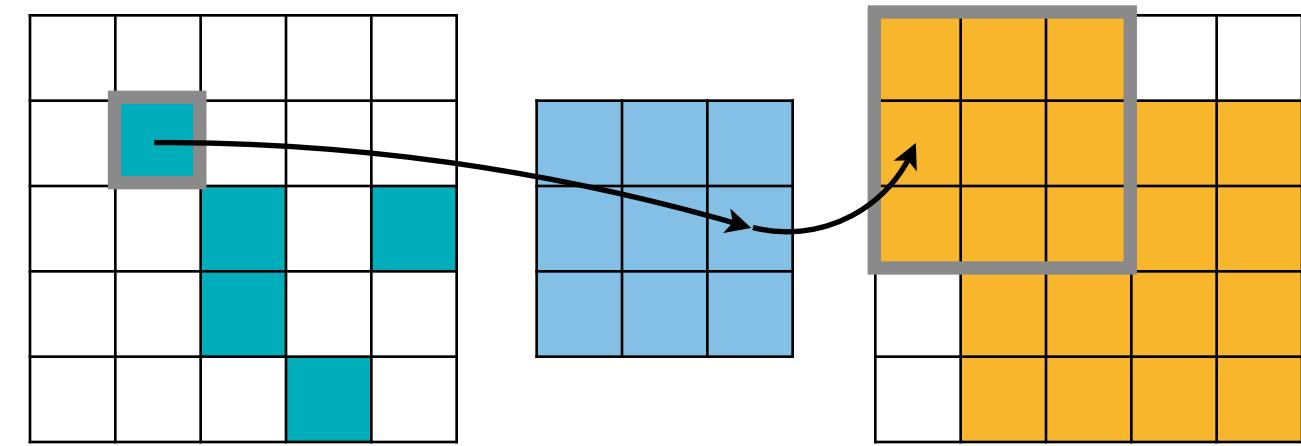
Computation

$(f_{\text{out}} = f_{\text{out}} + f_{\text{in}} \times W_{\text{wgt}})$ for each entry in the maps

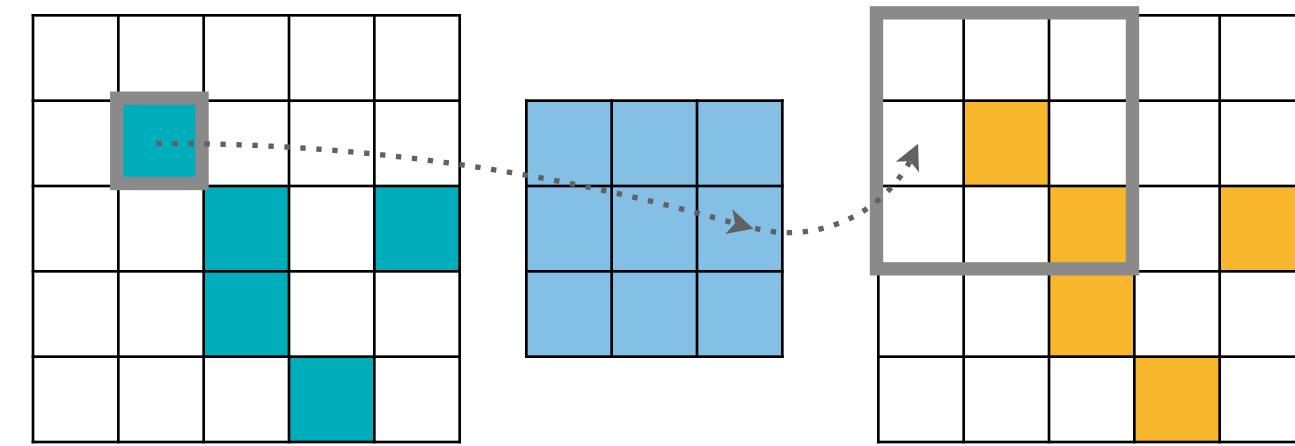
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$

No compute
No compute
No compute
No compute

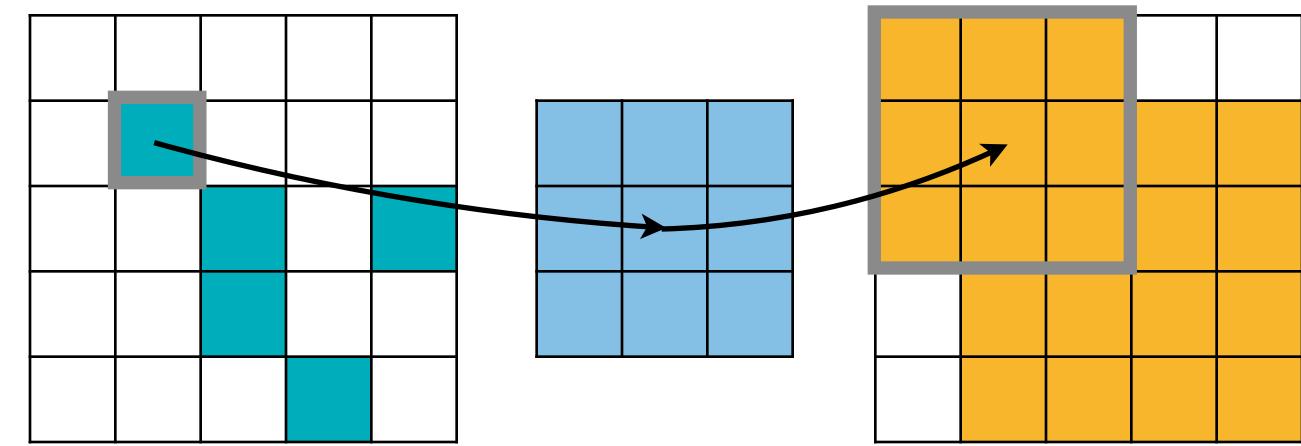
Computation

$(\text{f}_{\text{out}} = \text{f}_{\text{out}} + \text{f}_{\text{in}} \times \text{W}_{\text{wgt}})$ for each entry in the maps

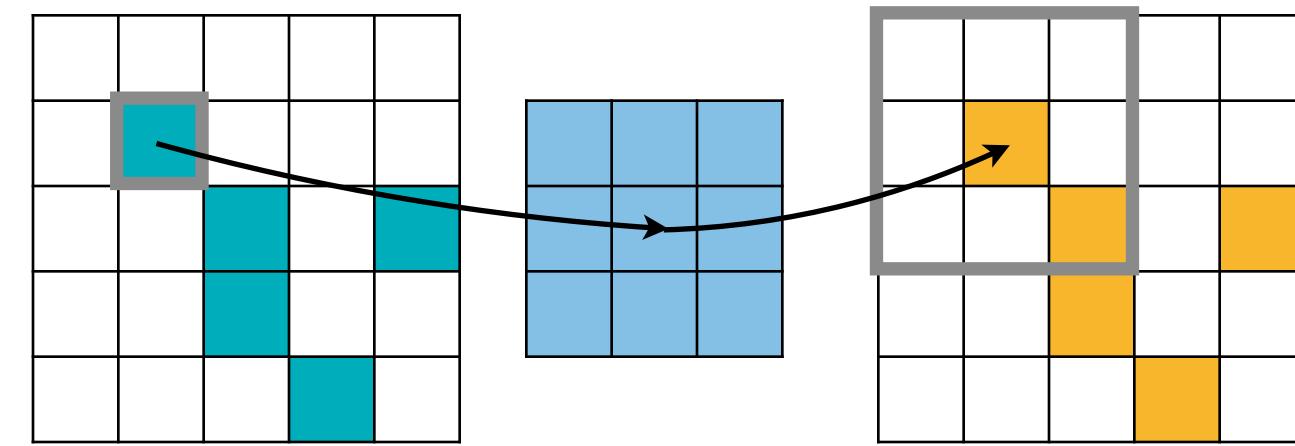
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$
 $(P_0, Q_4, W_{0,0})$

Computation

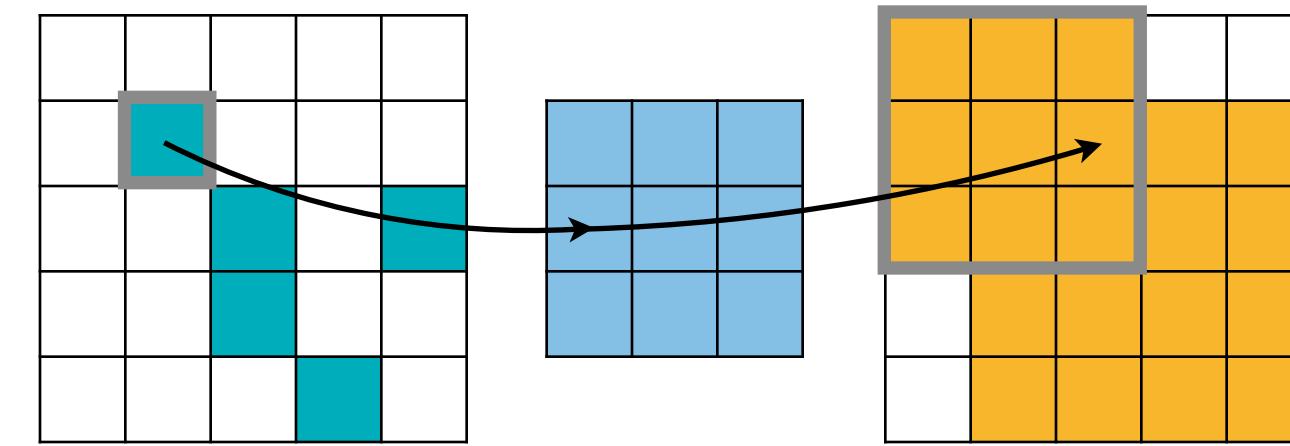
$(\text{f}_{\text{out}} = \text{f}_{\text{out}} + \text{f}_{\text{in}} \times \text{W}_{\text{wgt}})$ for each entry in the maps

No compute
No compute
No compute
No compute
 $(P_0, Q_0, W_{0,0})$

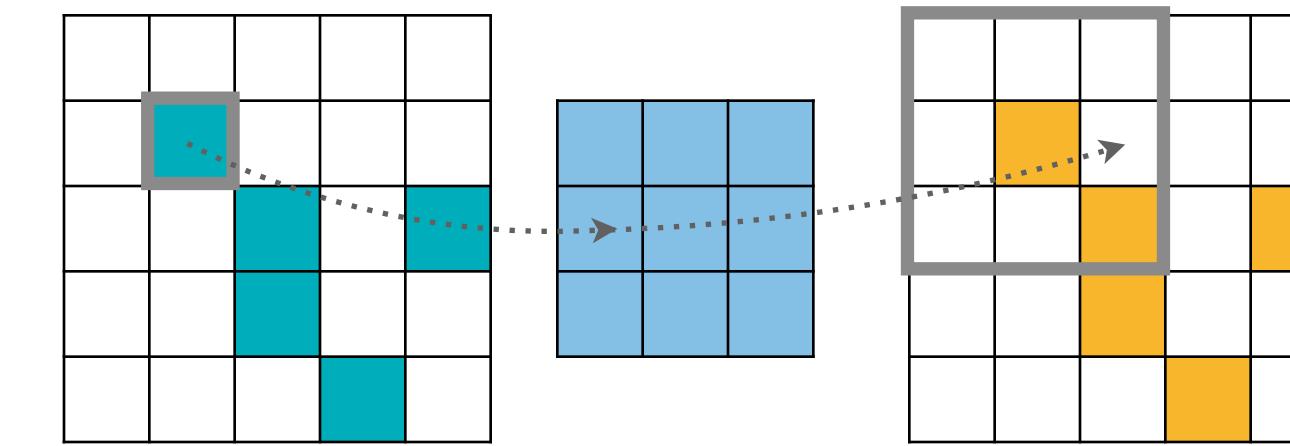
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$
 $(P_0, Q_4, W_{0,0})$
 $(P_0, Q_5, W_{0,-1})$

Computation

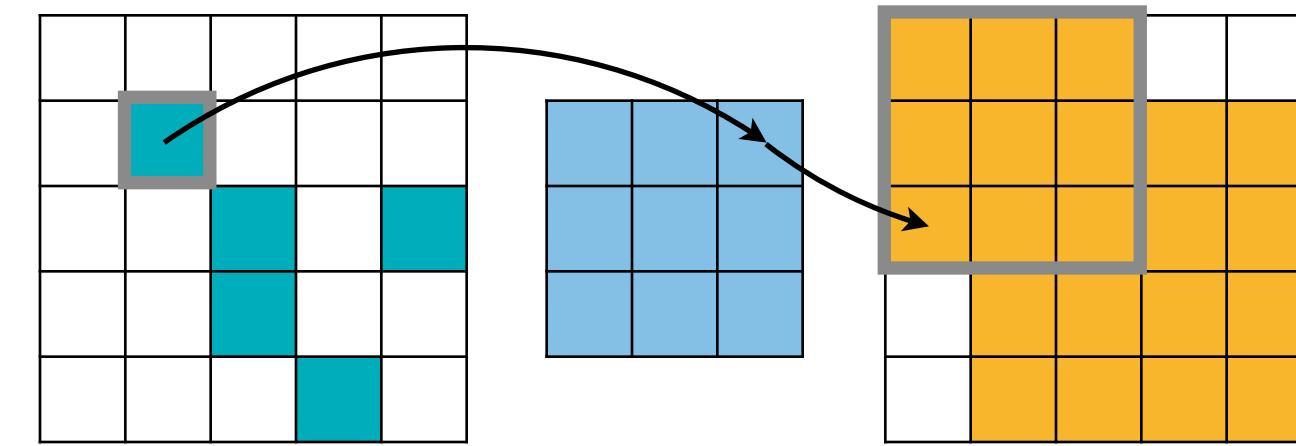
No compute
No compute
No compute
No compute
 $(P_0, Q_0, W_{0,0})$
No compute

$(\text{f}_{\text{out}} = \text{f}_{\text{out}} + \text{f}_{\text{in}} \times \text{W}_{\text{wgt}})$ for
each entry in the maps

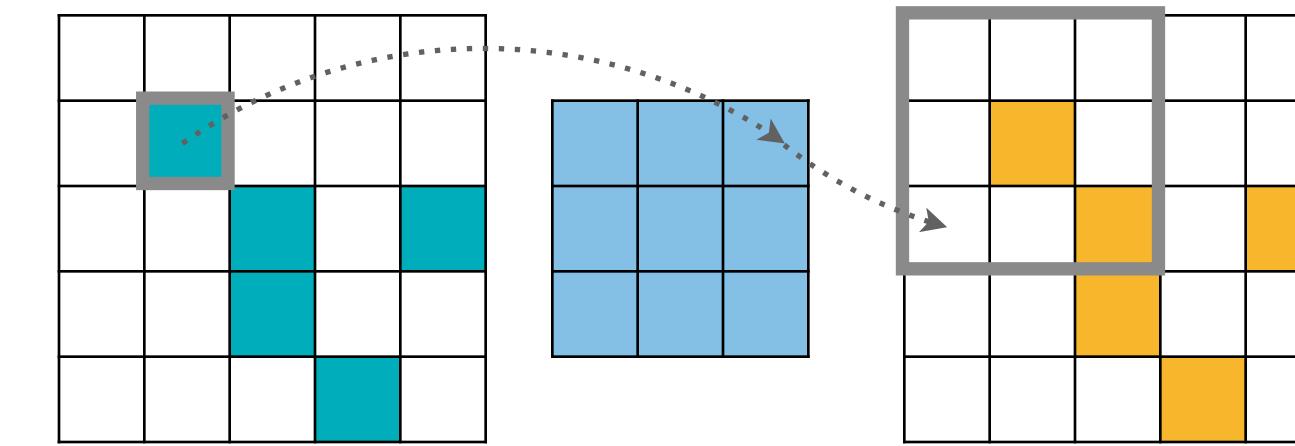
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$
 $(P_0, Q_4, W_{0,0})$
 $(P_0, Q_5, W_{0,-1})$
 $(P_0, Q_8, W_{-1,1})$

Computation

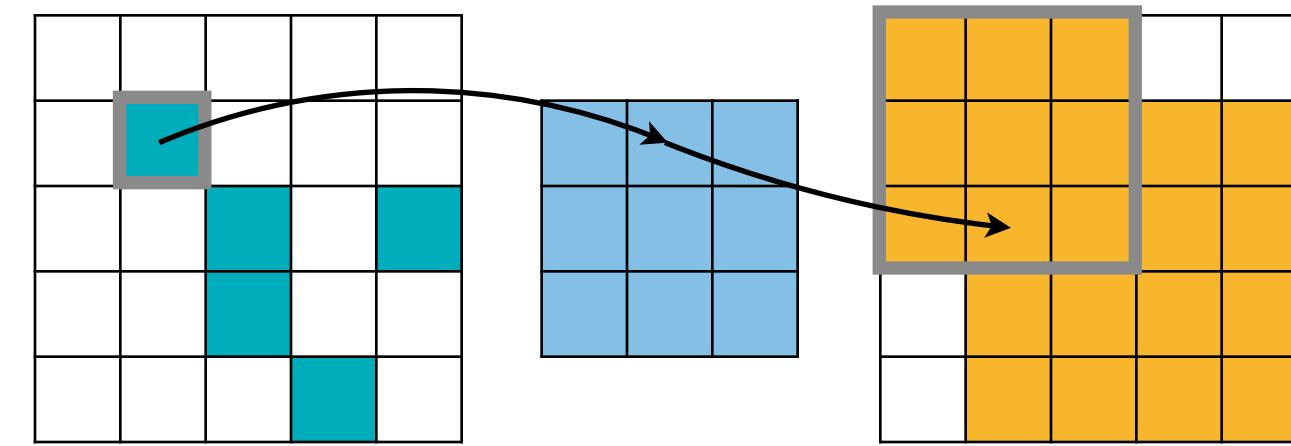
$(f_{\text{Out}} = f_{\text{Out}} + f_{\text{In}} \times W_{\text{Wgt}})$ for each entry in the maps

No compute
No compute
No compute
No compute
 $(P_0, Q_0, W_{0,0})$
No compute
No compute

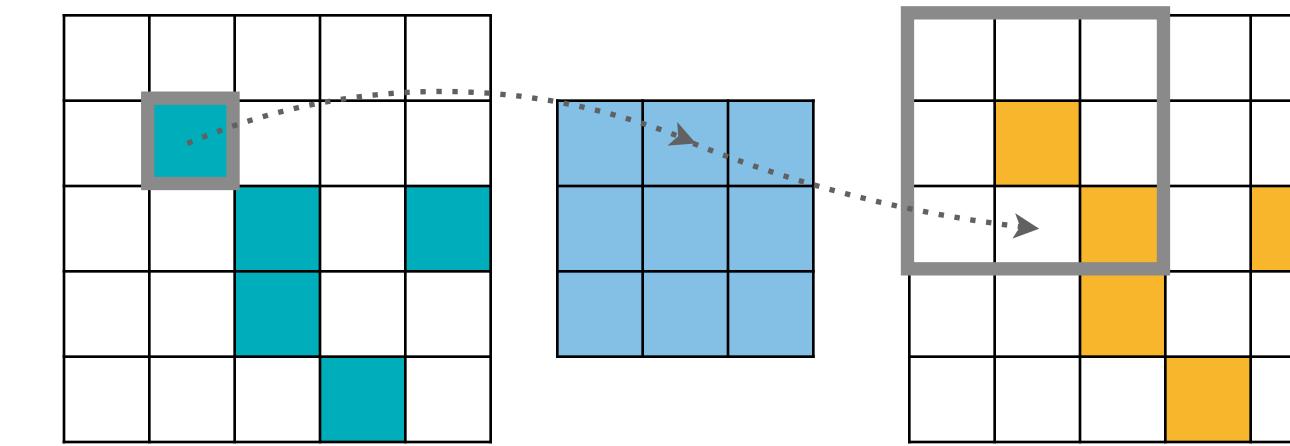
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$
 $(P_0, Q_4, W_{0,0})$
 $(P_0, Q_5, W_{0,-1})$
 $(P_0, Q_8, W_{-1,1})$
 $(P_0, Q_9, W_{-1,0})$

Computation

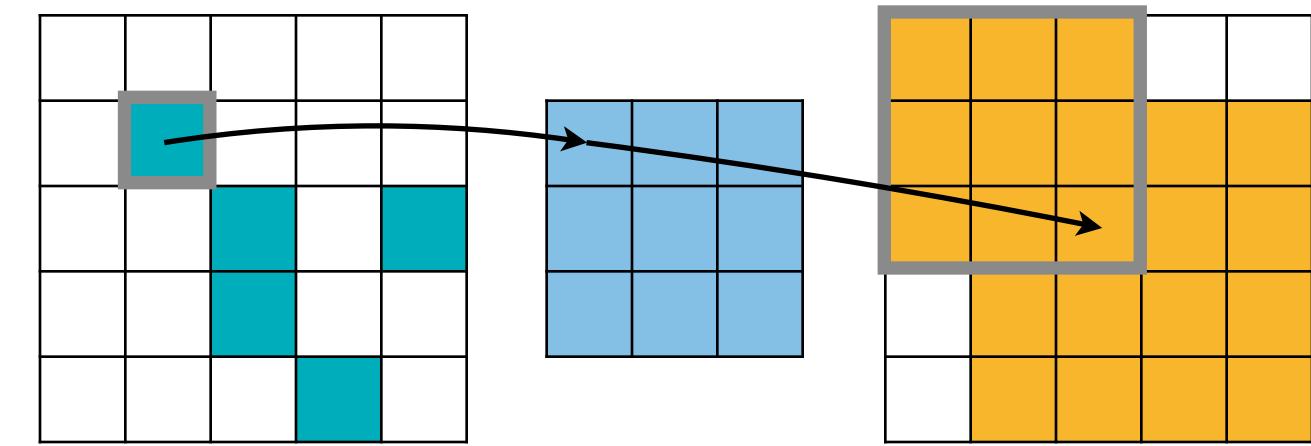
$(\text{f}_{\text{out}} = \text{f}_{\text{out}} + \text{f}_{\text{in}} \times \text{W}_{\text{wgt}})$ for
each entry in the maps

No compute
No compute
No compute
No compute
No compute
 $(P_0, Q_0, W_{0,0})$
No compute
No compute
No compute

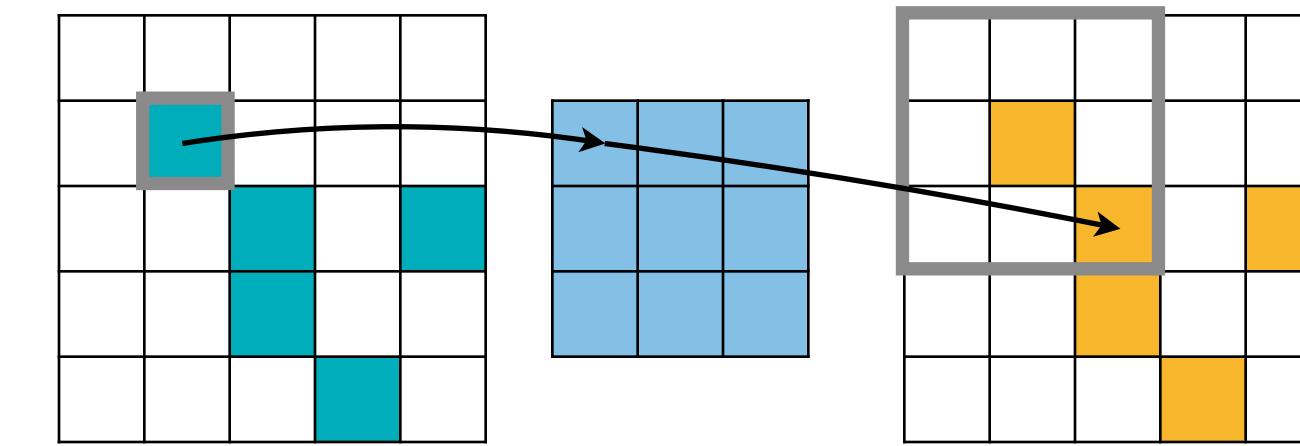
Sparse convolution computation

A sparse set of *dense* MMA, with rules defined by *maps*

Conventional Convolution



Sparse Convolution



Maps
(In , Out , Wgt)

$(P_0, Q_0, W_{1,1})$
 $(P_0, Q_1, W_{1,0})$
 $(P_0, Q_2, W_{1,-1})$
 $(P_0, Q_3, W_{0,1})$
 $(P_0, Q_4, W_{0,0})$
 $(P_0, Q_5, W_{0,-1})$
 $(P_0, Q_8, W_{-1,1})$
 $(P_0, Q_9, W_{-1,0})$
 $(P_0, Q_{10}, W_{-1,-1})$

Computation

$(\text{f}_{\text{out}} = \text{f}_{\text{out}} + \text{f}_{\text{In}} \times \text{W}_{\text{Wgt}})$ for
each entry in the maps

No compute
No compute
No compute
No compute
No compute
 $(P_0, Q_0, W_{0,0})$
No compute
No compute
No compute
 $(P_0, Q_1, W_{-1,-1})$

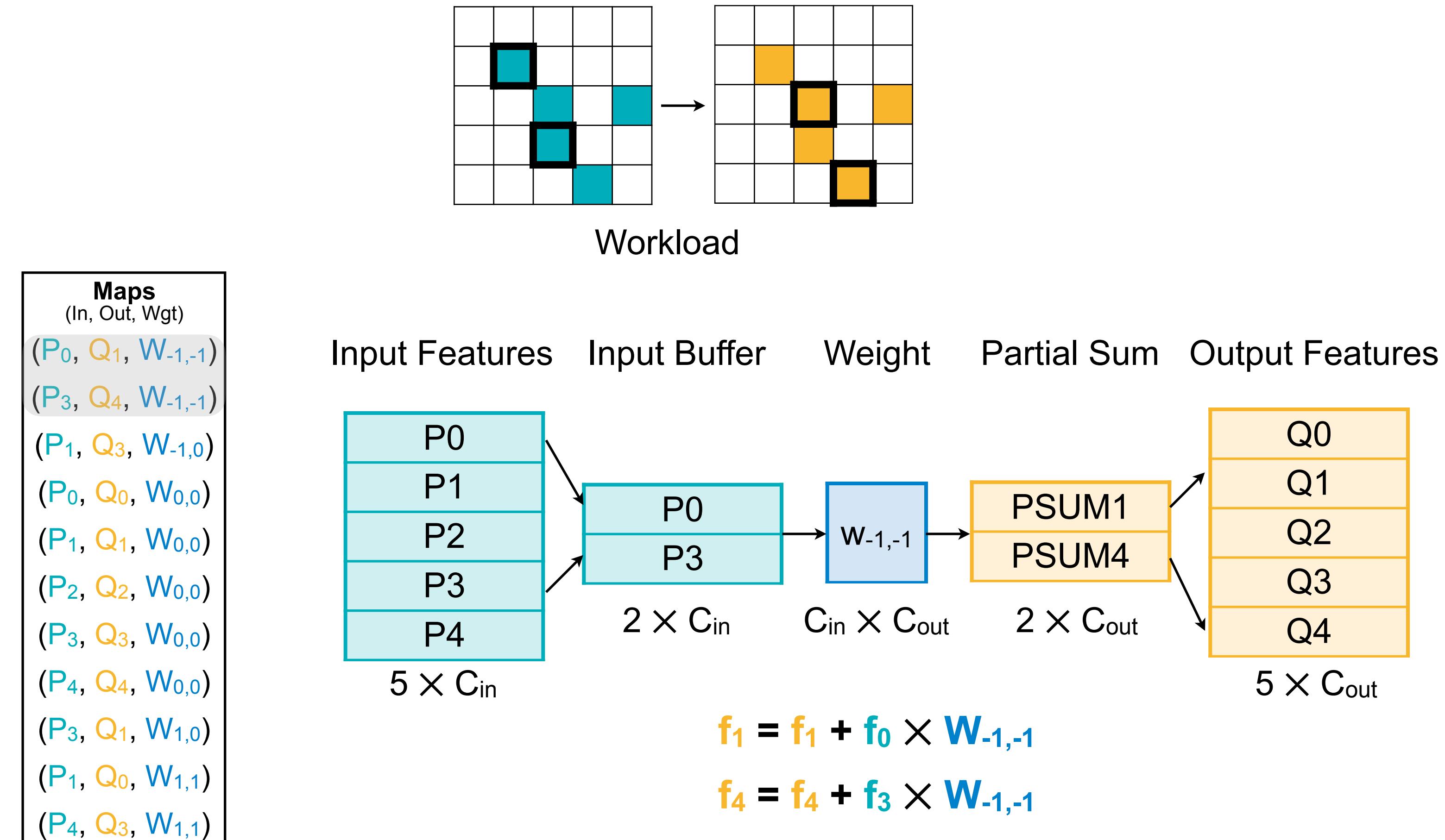
9 matrix multiplications

2 matrix multiplications

TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Existing GPU implementation of sparse convolution

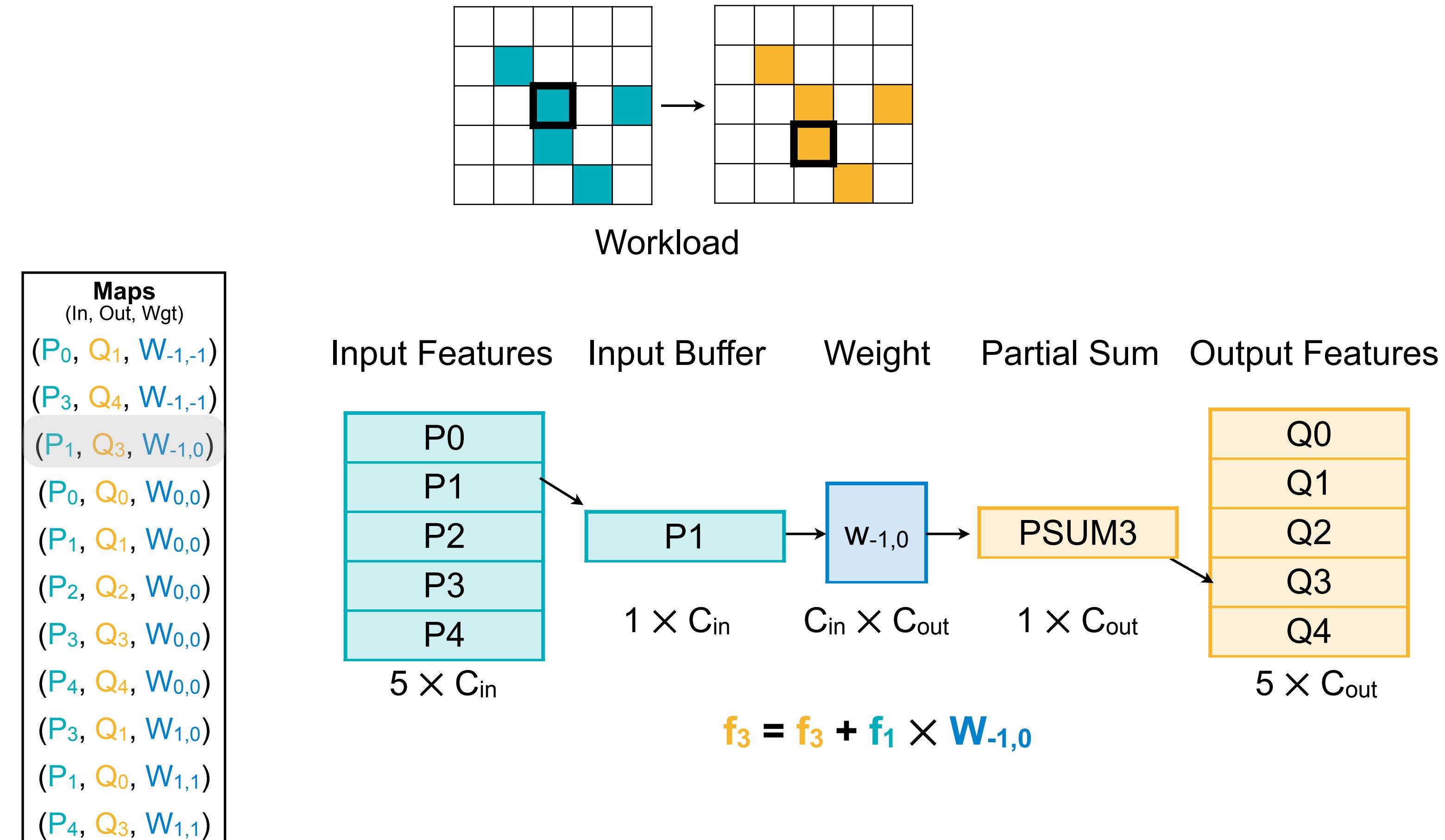
Weight-stationary computation, separate matmul for different weights



TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Existing GPU implementation of sparse convolution

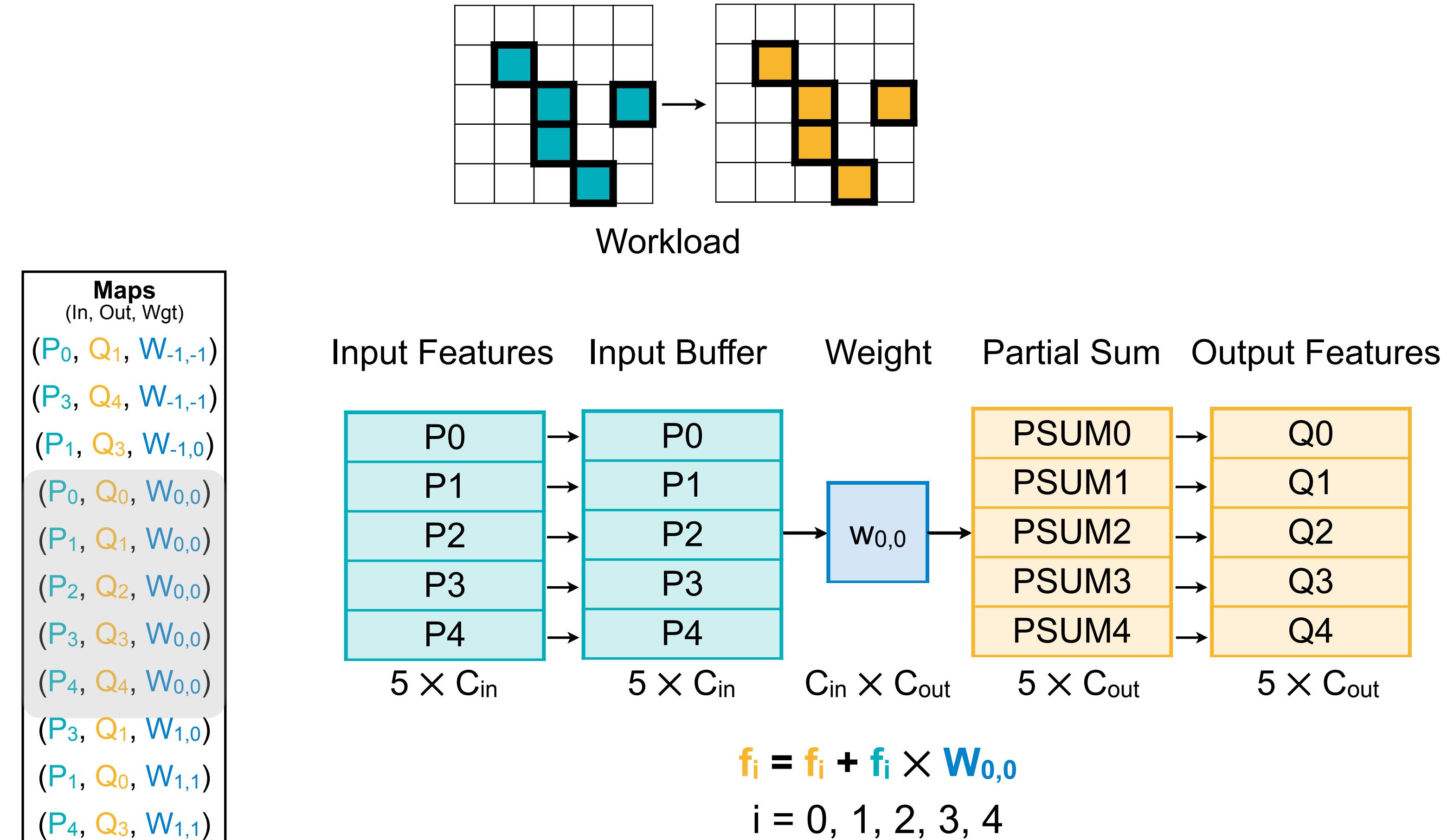
Weight-stationary computation, separate matmul for different weights



TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights

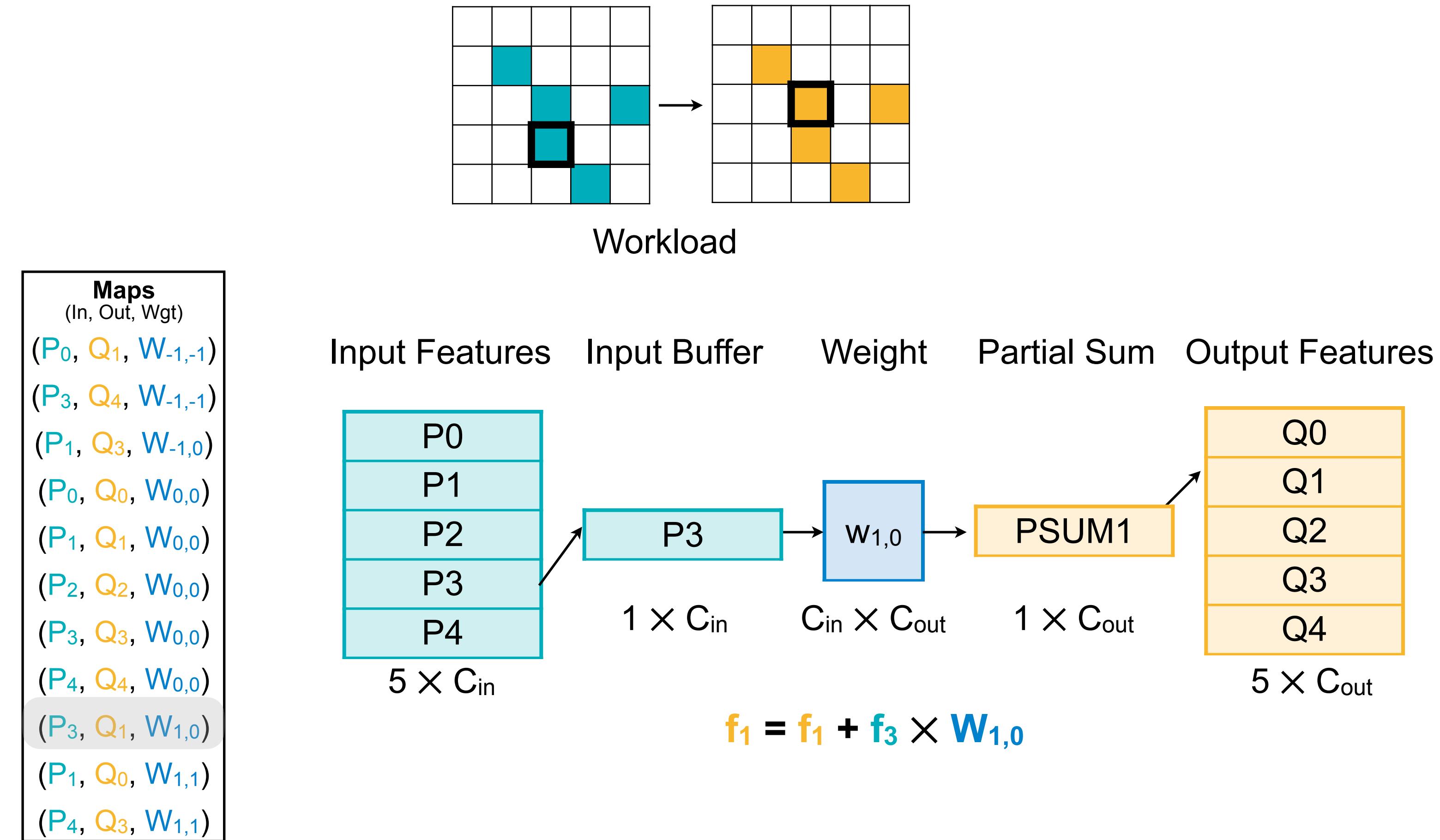


Note: maps for $W_{0,0}$ contains all entries.

TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Existing GPU implementation of sparse convolution

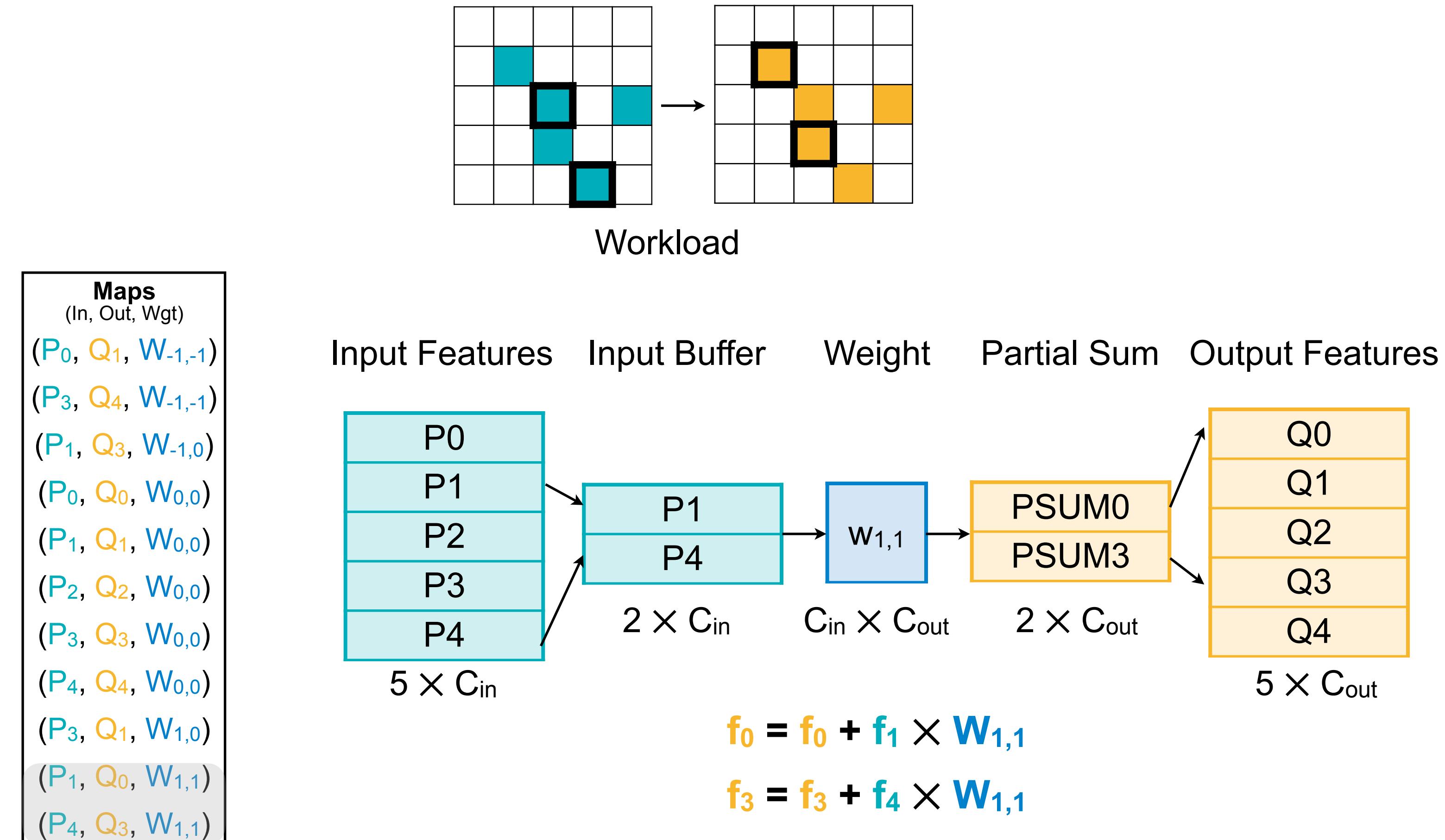
Weight-stationary computation, separate matmul for different weights



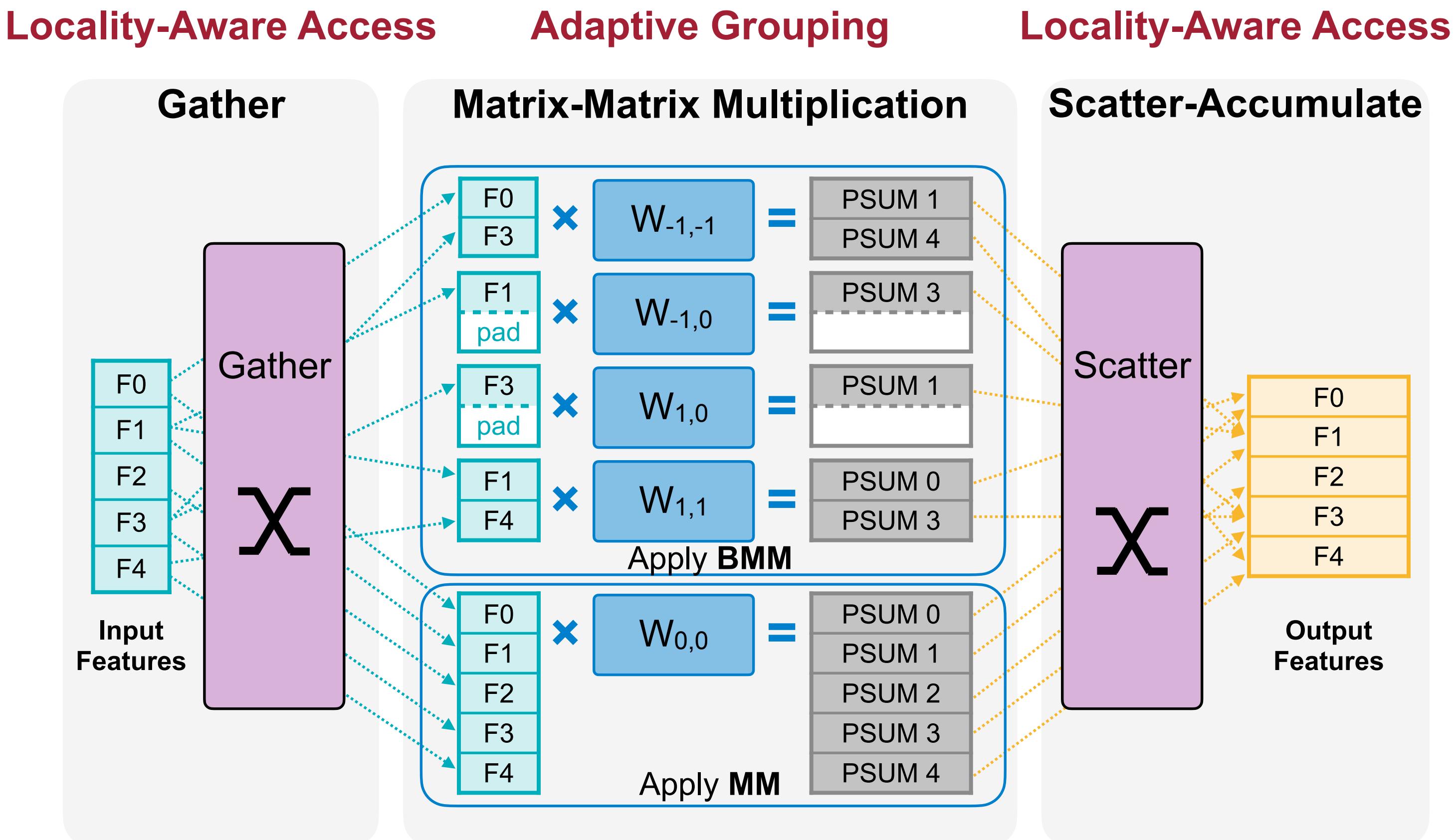
TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Existing GPU implementation of sparse convolution

Weight-stationary computation, separate matmul for different weights



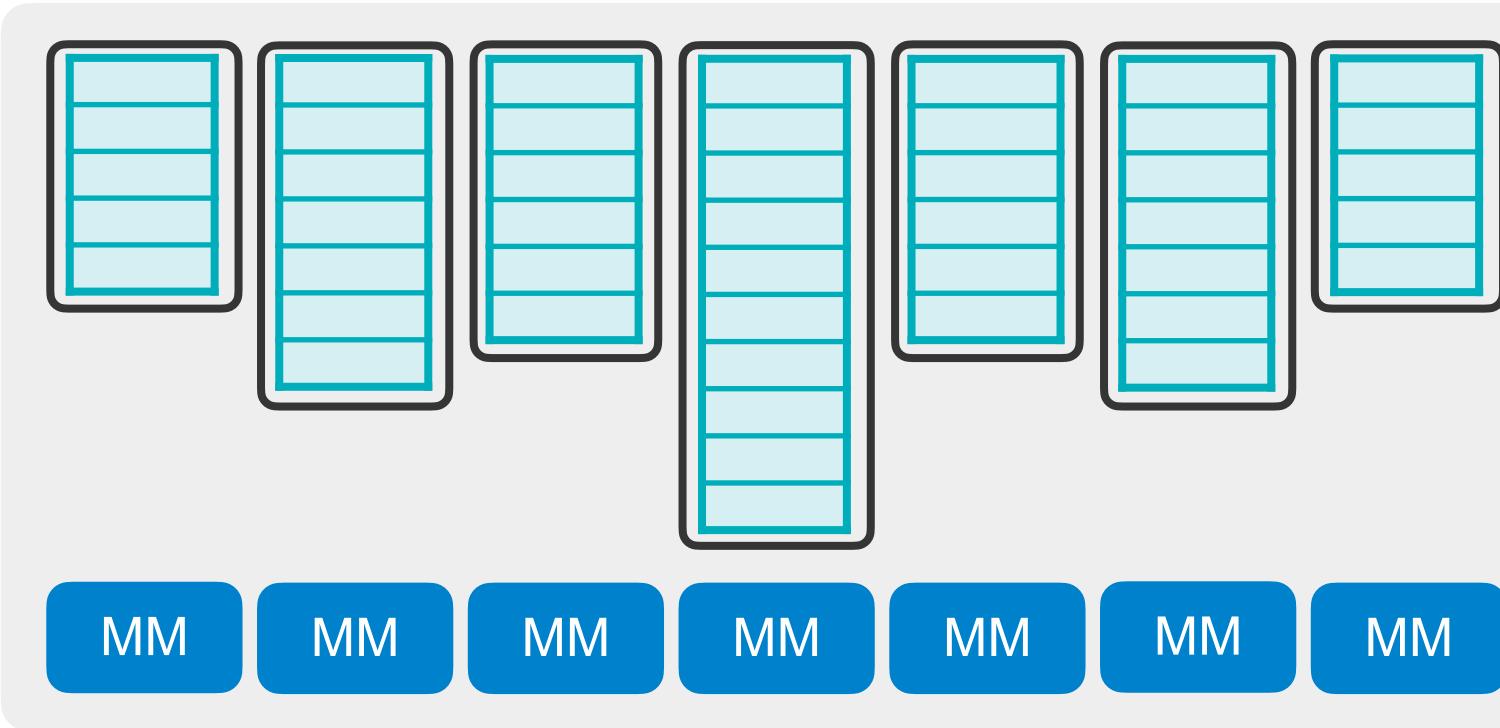
TorchSparse optimization overview



TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Trading computation for regularity

Separate computation (baseline) : many kernel calls, low device utilization



Separate Computation

Worst

Best



Computation overhead

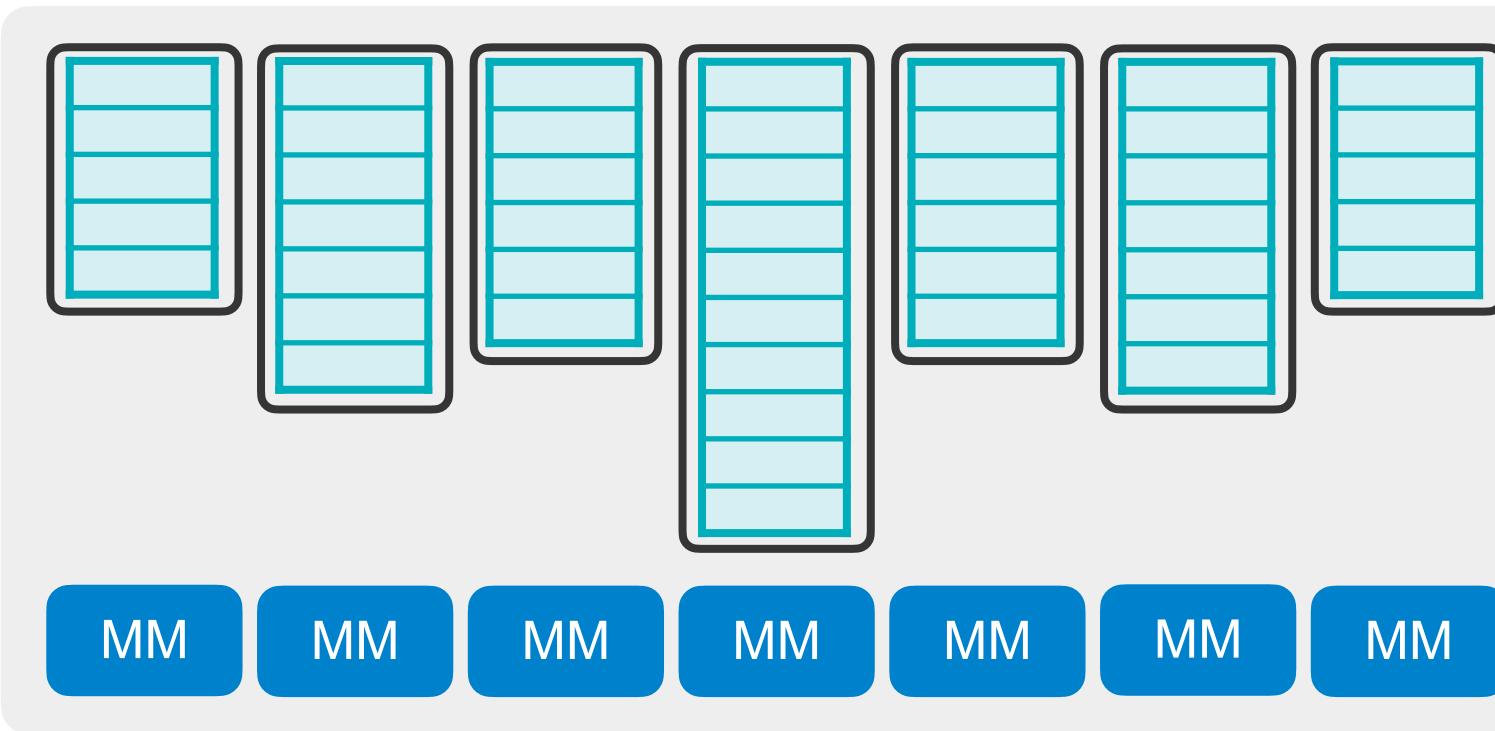


Computation regularity

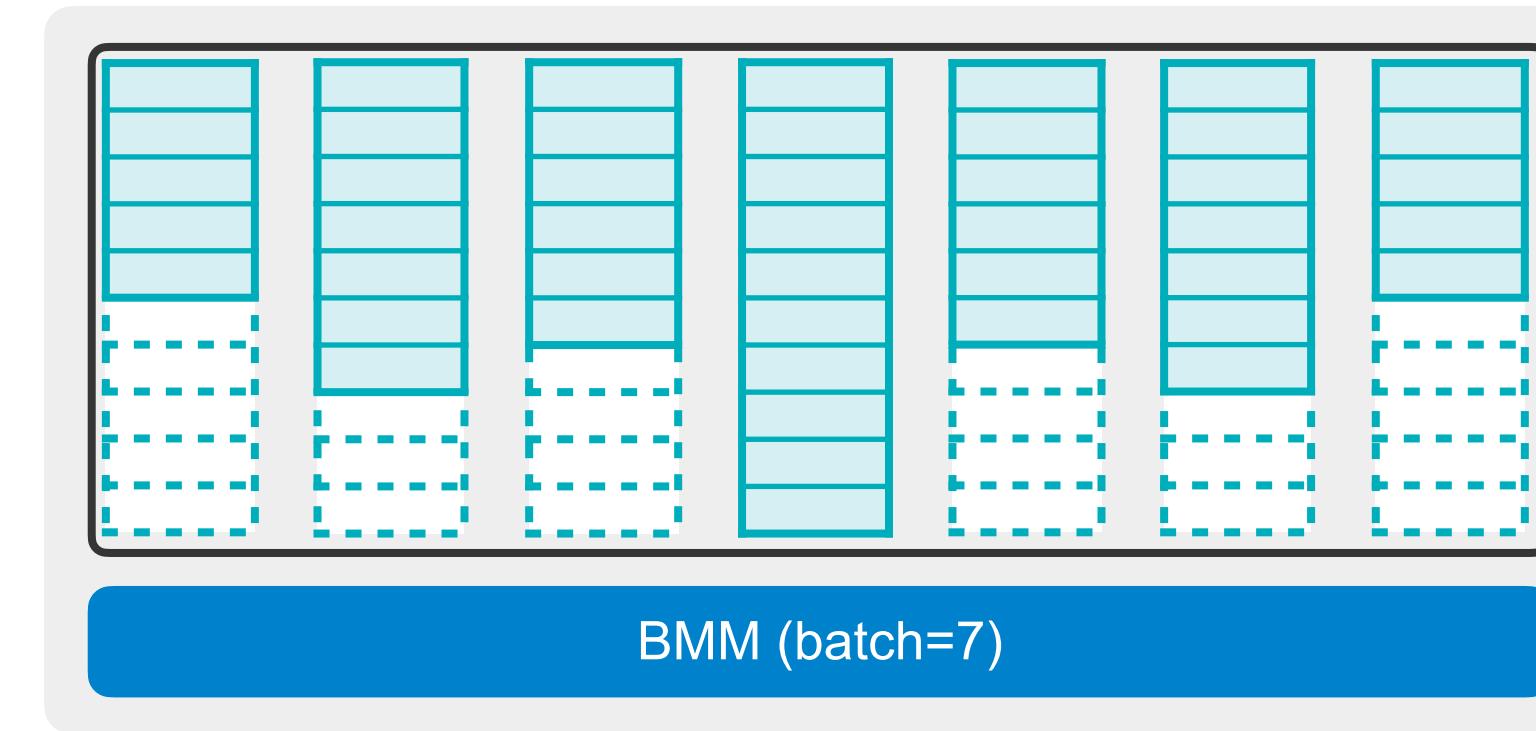
TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Trading computation for regularity

Dense convolution: best regularity but large computation overhead



Separate Computation



Dense Convolution

Worst

Best



Computation overhead



Computation regularity

Worst

Best



Computation overhead

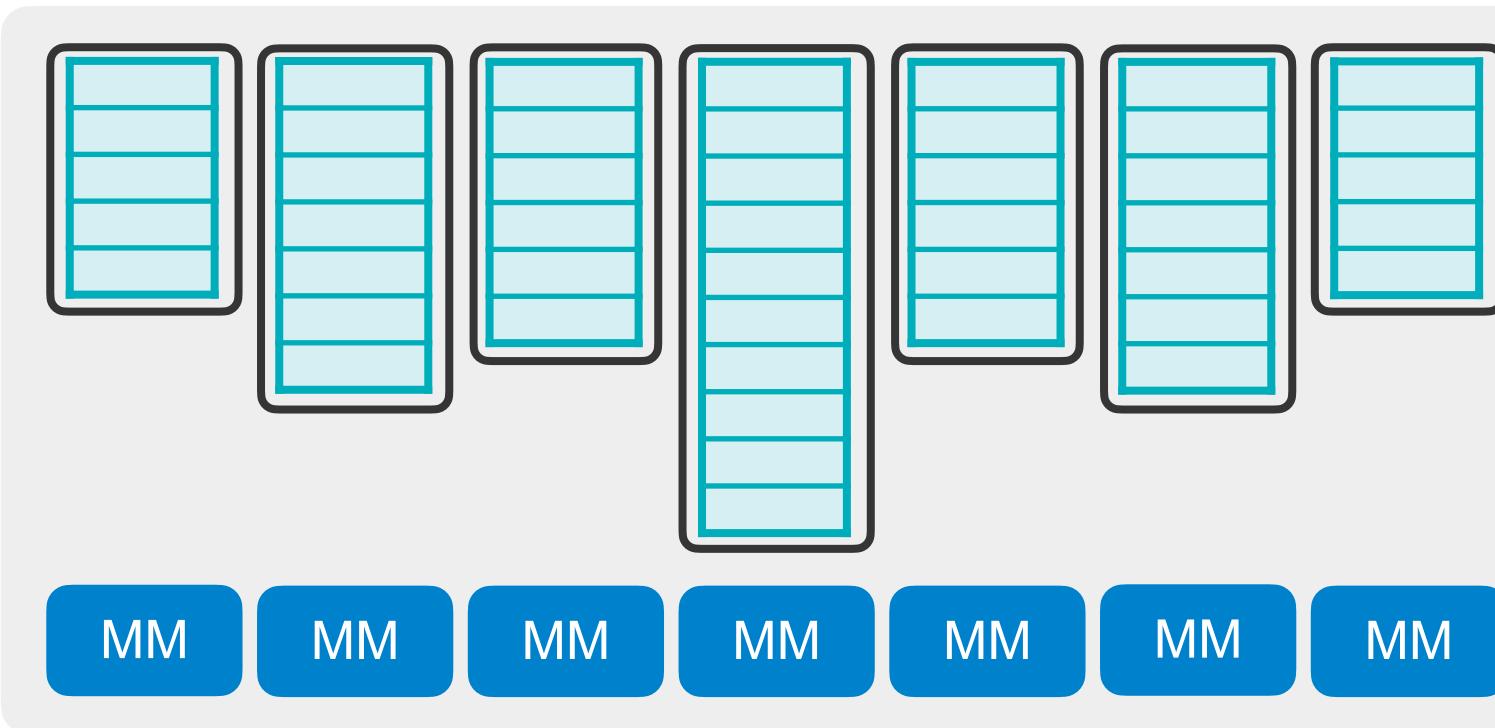


Computation regularity

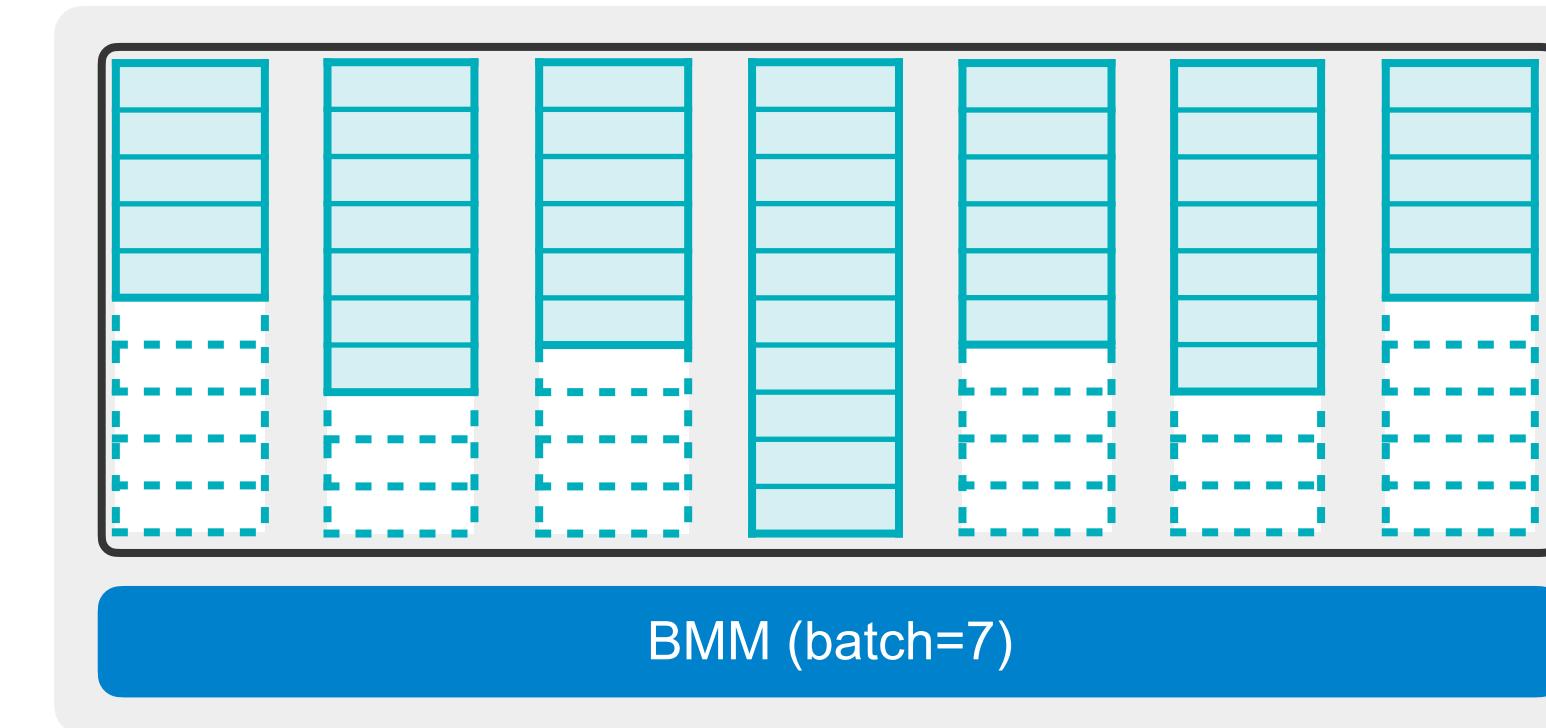
TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Trading computation for regularity

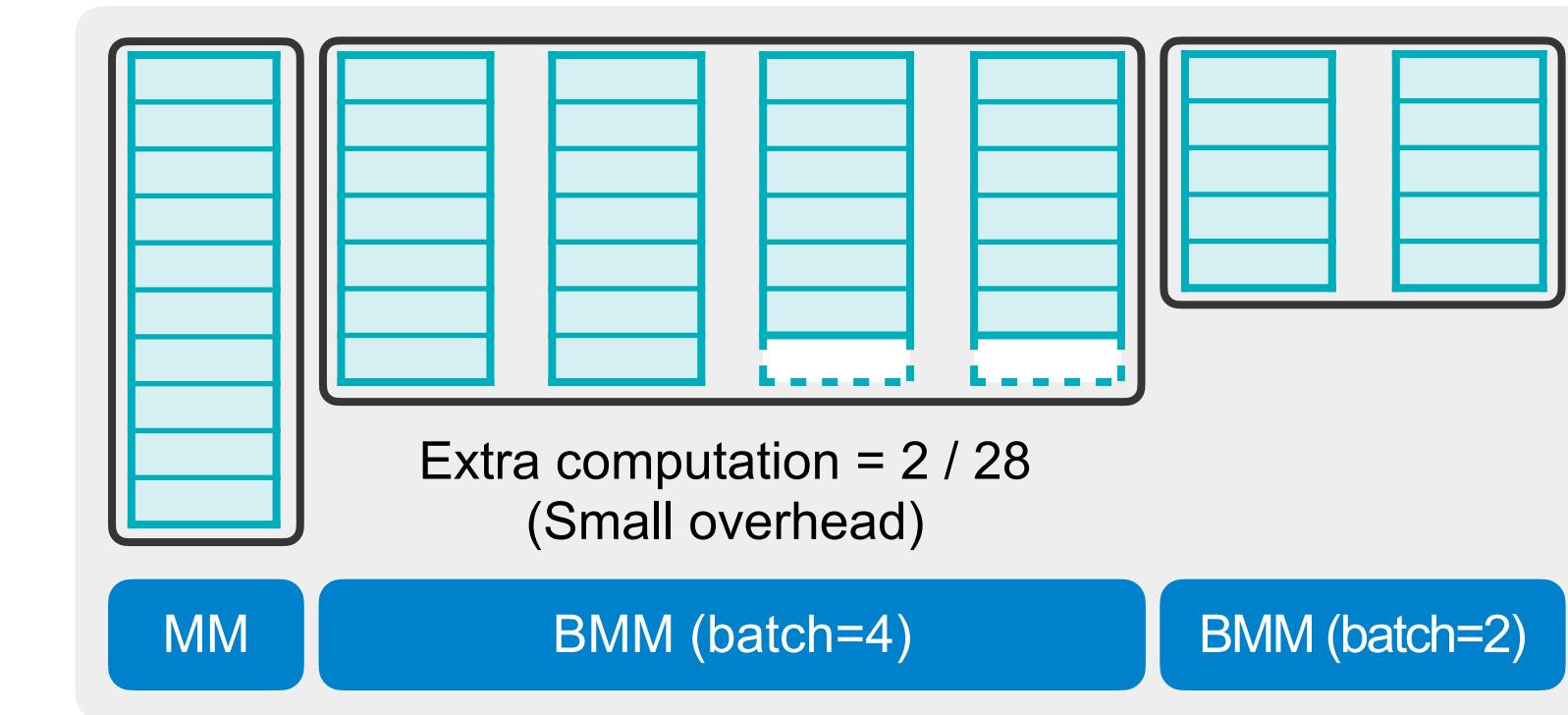
Computation with grouping: balancing overhead and regularity



Separate Computation



Dense Convolution



Computation with grouping

Worst **Best**



Computation overhead



Computation regularity

Worst **Best**



Computation overhead



Computation regularity

Worst **Best**



Computation overhead

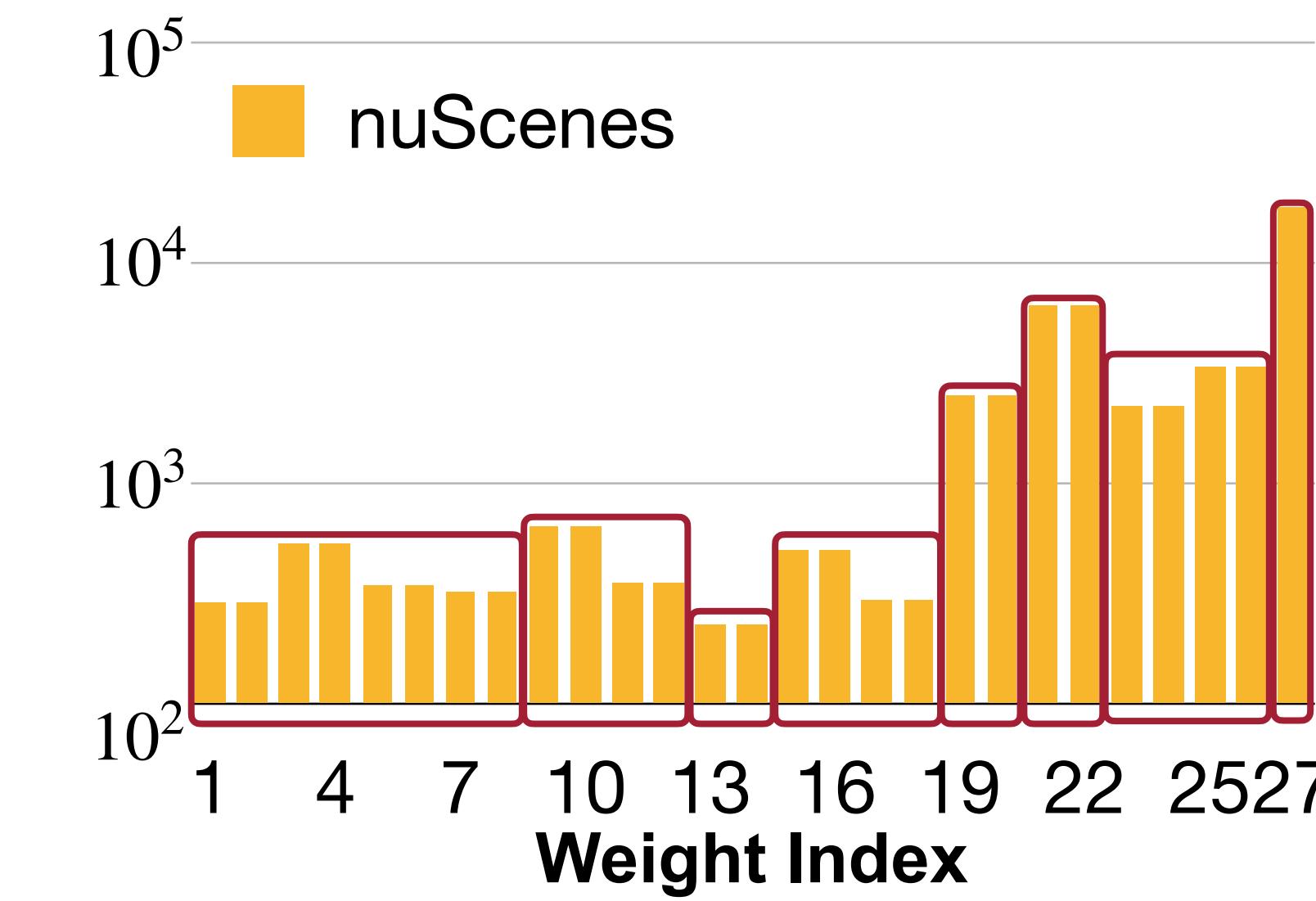
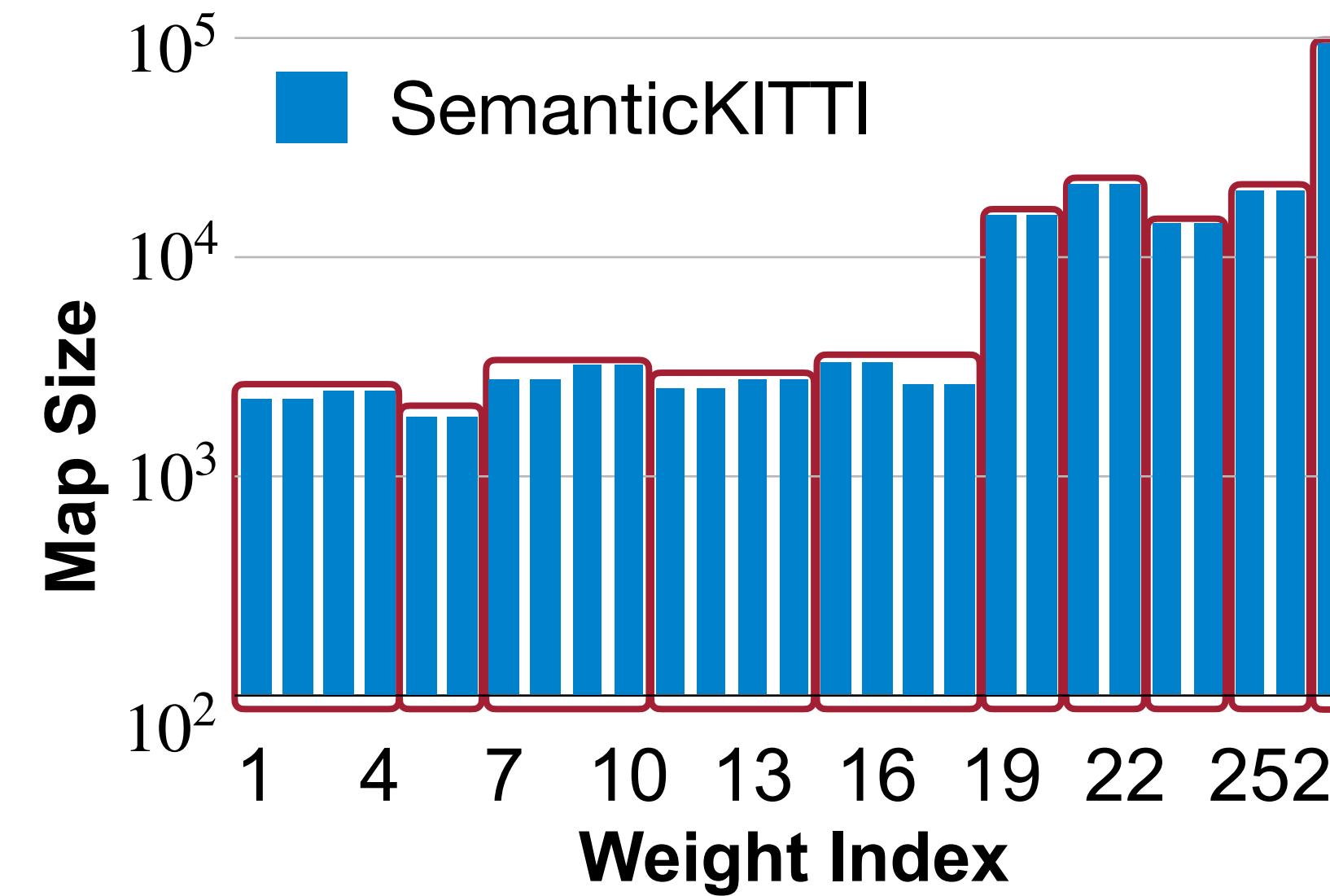
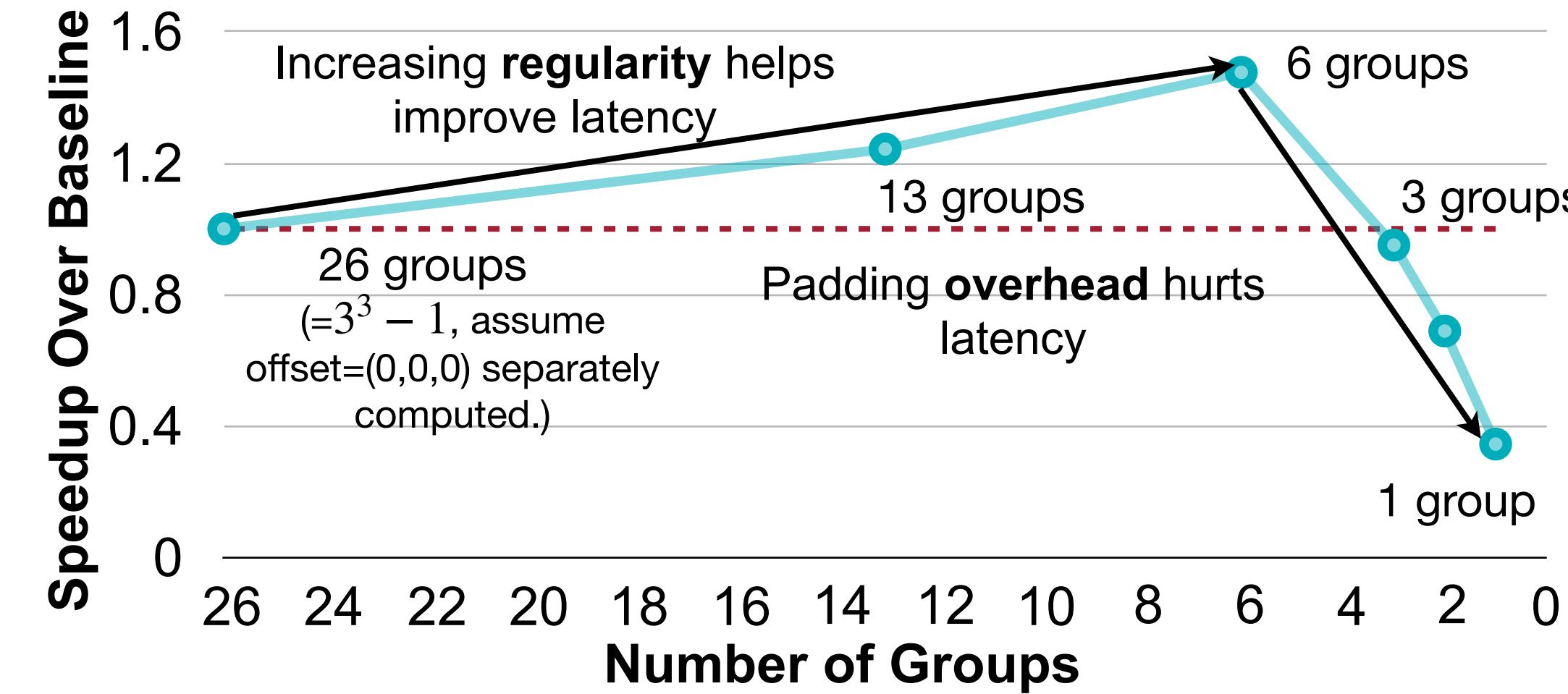


Computation regularity

TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

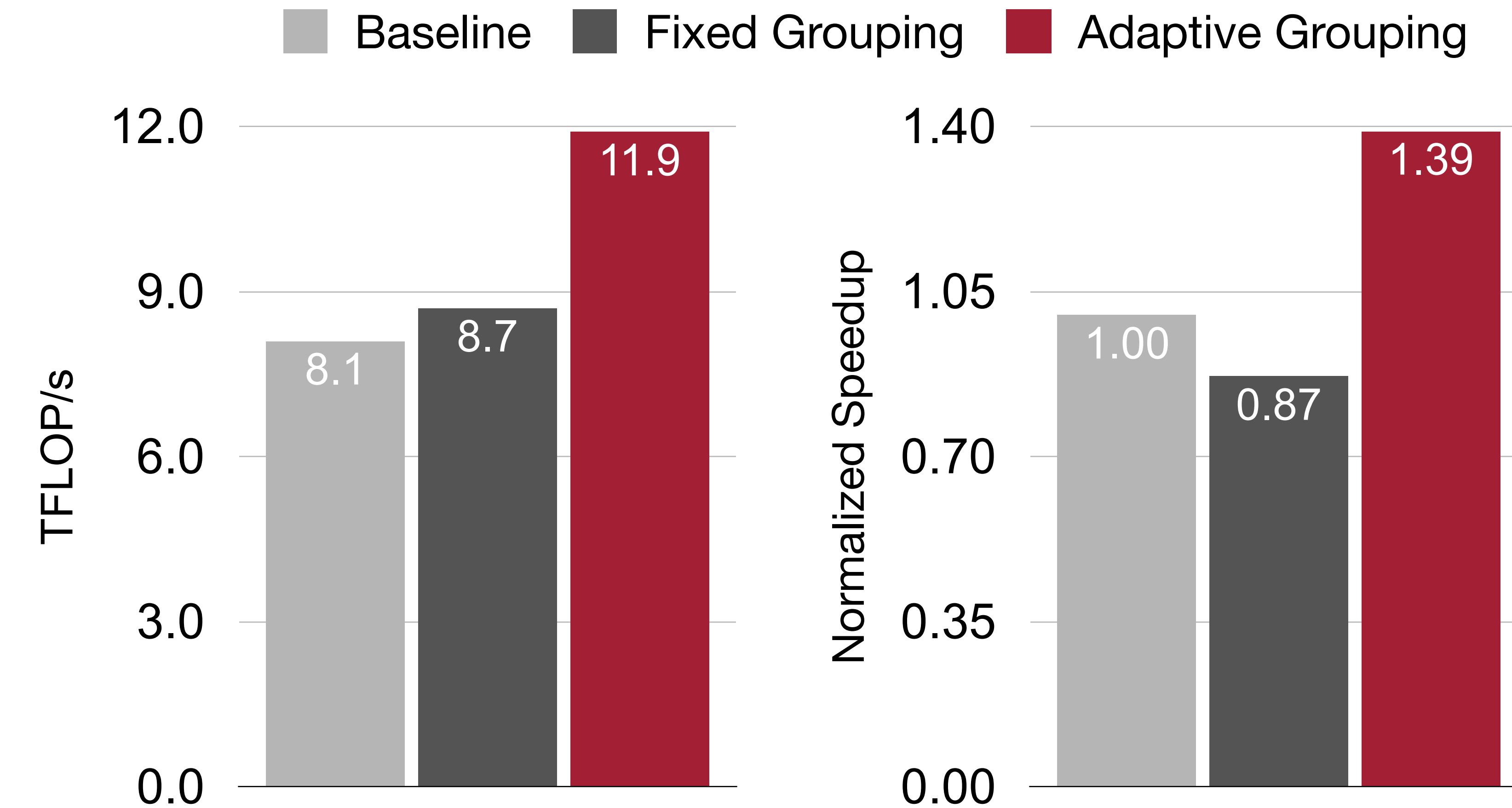
Trading computation for regularity

Searching customized strategy for different model and datasets



Results on matrix multiplication optimizations

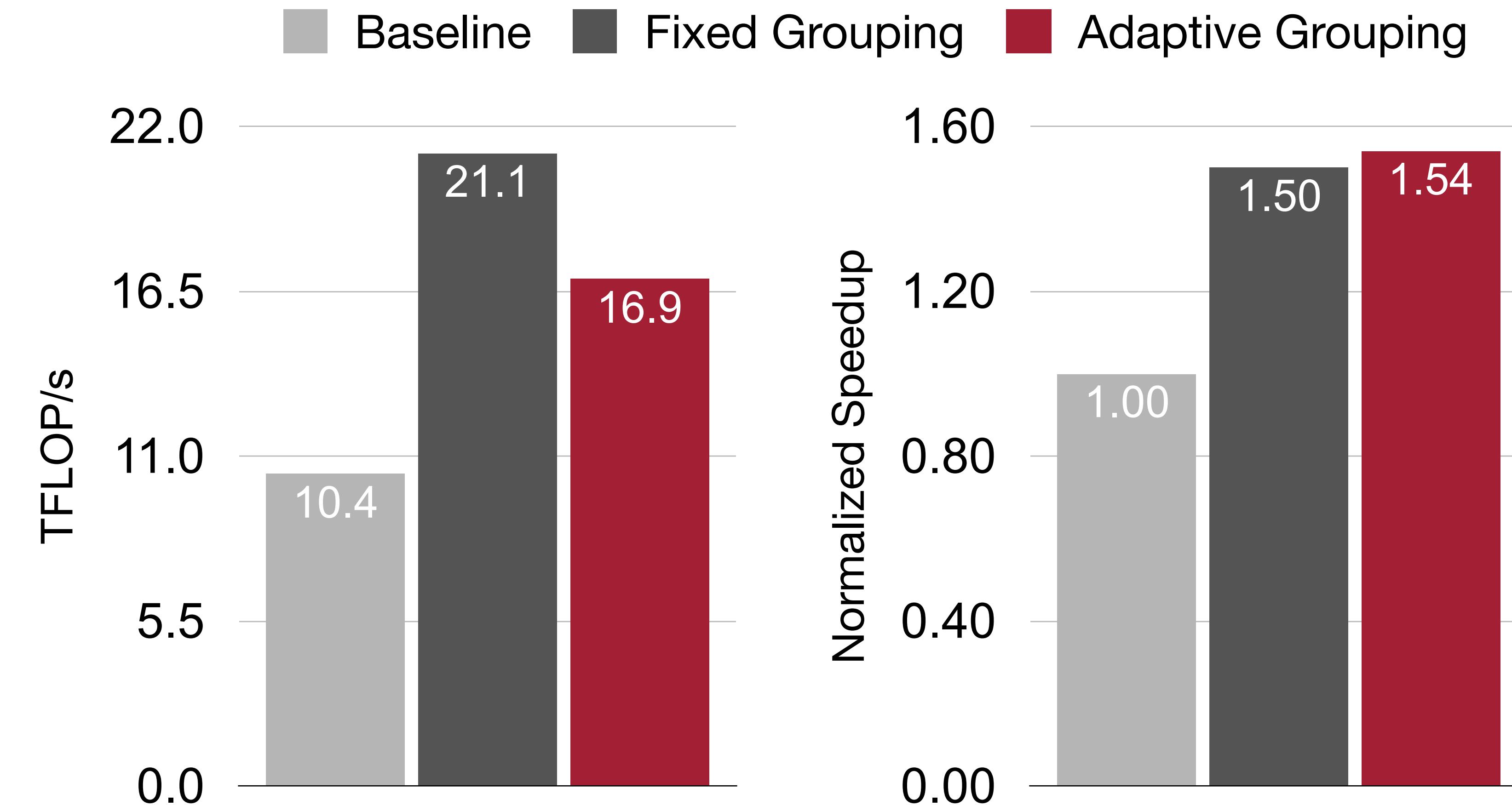
SemanticKITTI



TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

Results on matrix multiplication optimizations

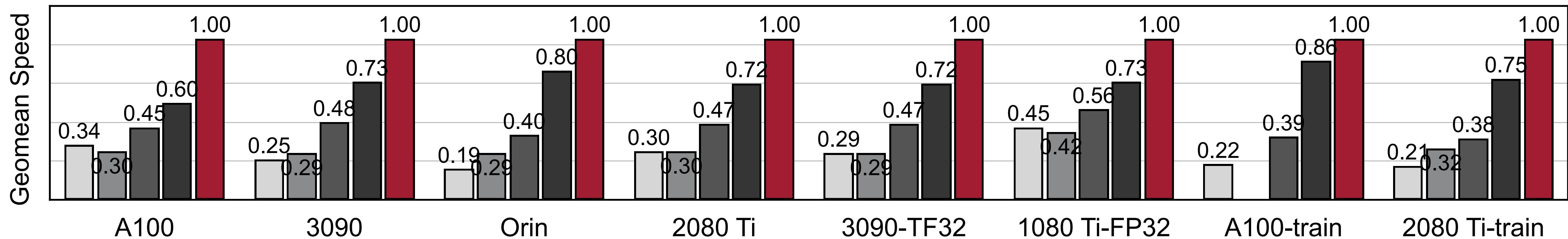
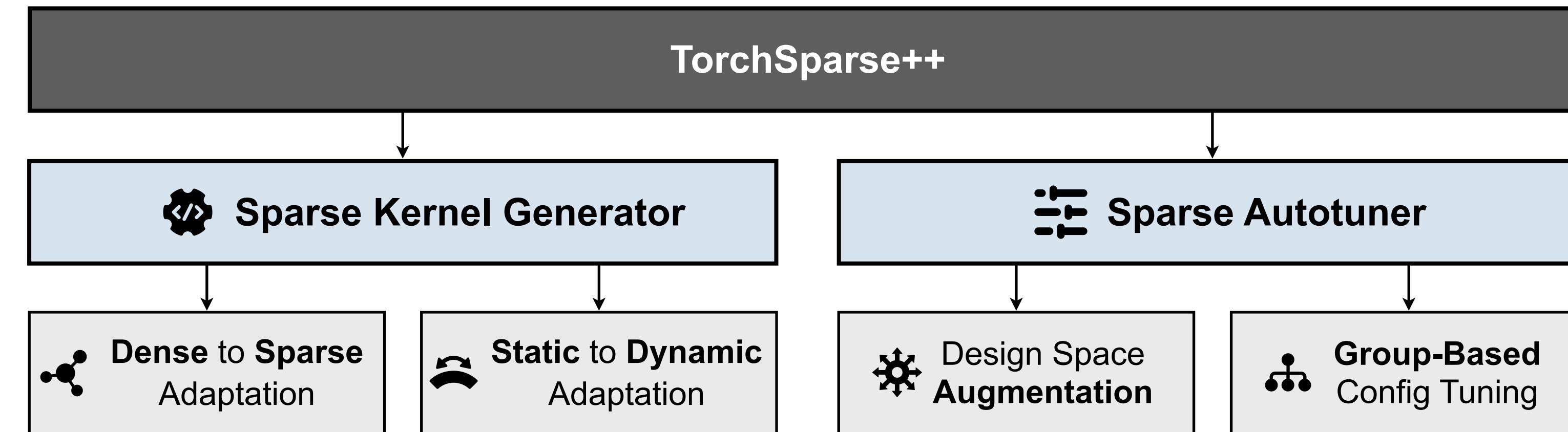
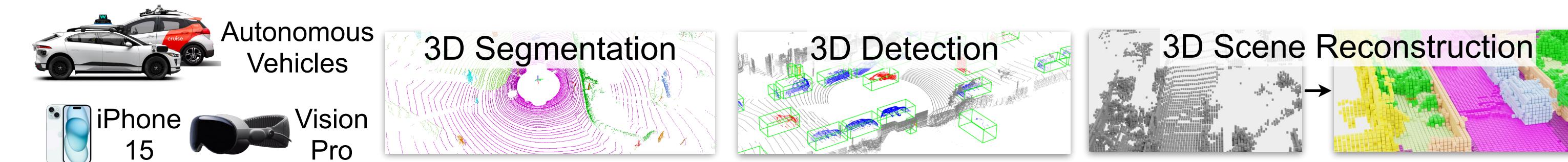
nuScenes: fixed grouping has best TFLOP/s but adaptive grouping is faster



This is because fixed grouping introduced large amount of **redundant computation**.

TorchSparse: Efficient Point Cloud Inference Engine [Tang et al., MLSys 2022]

TorchSparse++: Overlapped memory and computation



TorchSparse++ [Tang and Yang et. al, MICRO 2023]

TorchSparse++: Overlapped memory and computation

Vanilla

	W _{-1,-1}	W _{-1,0}	W _{-1,1}	W _{0,-1}	W _{0,0}	W _{0,1}	W _{1,-1}	W _{1,0}	W _{1,1}
B ₀	0	0	0	0	A ₀	A ₁	0	0	A ₃
B ₁	0	0	0	A ₀	A ₁	A ₂	0	A ₃	A ₄
B ₂	0	0	0	A ₁	A ₂	0	A ₃	A ₄	0
B ₃	A ₀	A ₁	A ₂	0	A ₃	A ₄	0	0	0
B ₄	A ₁	A ₂	0	A ₃	A ₄	0	0	0	A ₅
B ₅	A ₄	0	0	0	A ₅	0	0	0	A ₇
B ₆	0	0	0	0	A ₆	0	0	0	0
B ₇	A ₅	0	0	0	A ₇	0	0	0	0

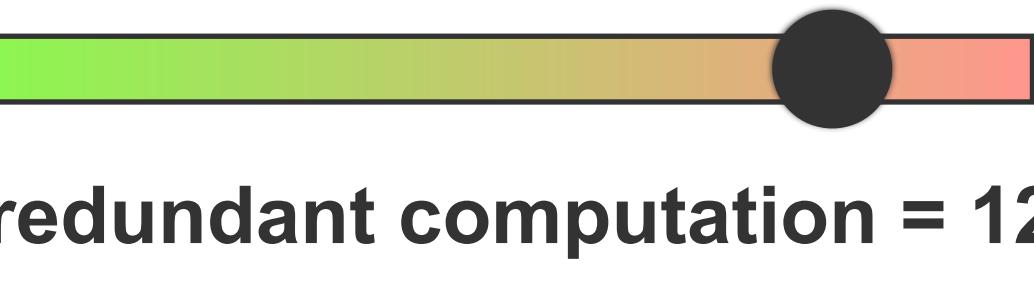
+ Row Reordering

	W _{-1,-1}	W _{-1,0}	W _{-1,1}	W _{0,-1}	W _{0,0}	W _{0,1}	W _{1,-1}	W _{1,0}	W _{1,1}
B ₆	0	0	0	0	A ₆	0	0	0	0
B ₀	0	0	0	0	A ₀	A ₁	0	0	A ₃
B ₂	0	0	0	A ₁	A ₂	0	A ₃	A ₄	0
B ₁	0	0	0	A ₀	A ₁	A ₂	0	A ₃	A ₄
B ₇	A ₅	0	0	0	A ₇	0	0	0	0
B ₅	A ₄	0	0	0	A ₅	0	0	0	A ₇
B ₄	A ₁	A ₂	0	A ₃	A ₄	0	0	0	A ₅
B ₃	A ₀	A ₁	A ₂	0	A ₃	A ₄	0	0	0

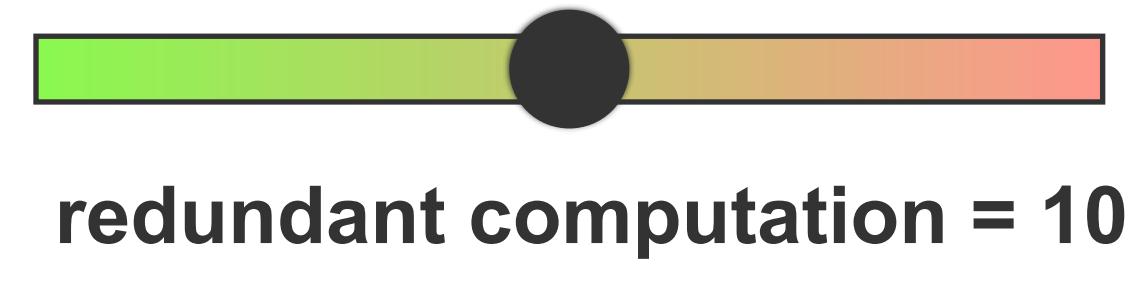
+ Column Splitting

	W _{-1,-1}	W _{-1,0}	W _{-1,1}		W _{0,-1}	W _{0,0}	W _{0,1}		W _{1,-1}	W _{1,0}	W _{1,1}
B ₀	0	0	0		B ₅	0	A ₅	0	B ₃	0	0
B ₁	0	0	0		B ₆	0	A ₆	0	B ₆	0	0
B ₂	0	0	0		B ₇	0	A ₇	0	B ₇	0	0
B ₆	0	0	0		B ₀	0	A ₀	A ₁	B ₀	0	A ₃
B ₅	A ₄	0	0		B ₃	0	A ₃	A ₄	B ₄	0	A ₅
B ₇	A ₅	0	0		B ₂	A ₁	A ₂	0	B ₅	0	A ₇
B ₄	A ₁	A ₂	0		B ₄	A ₃	A ₄	0	B ₁	0	A ₃
B ₃	A ₀	A ₁	A ₂		B ₁	A ₀	A ₁	A ₂	B ₂	A ₃	A ₄

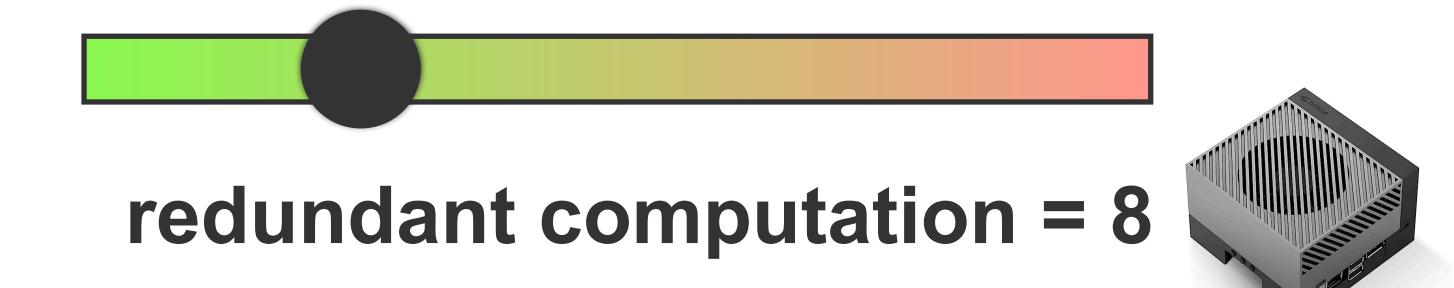
Redundancy



Redundancy



Redundancy



TorchSparse++ [Tang and Yang et. al, MICRO 2023]

System & Hardware Support for Sparsity

- EIE: Weight Sparsity + Activation Sparsity
- NVIDIA Tensor Core: M:N Weight Sparsity
- **TorchSparse & PointAcc: Activation Sparsity**
 - **TorchSparse: Sparse Convolution Library**
 - **PointAcc: Hardware Accelerator for Sparse Convolution**

Mapping Unit

Merge sort can be used to find mappings in sparse convolution

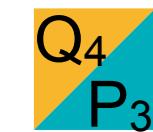
Input Point Cloud

P ₀				
	P ₁		P ₂	
		P ₃		
			P ₄	

↓ stride = 1



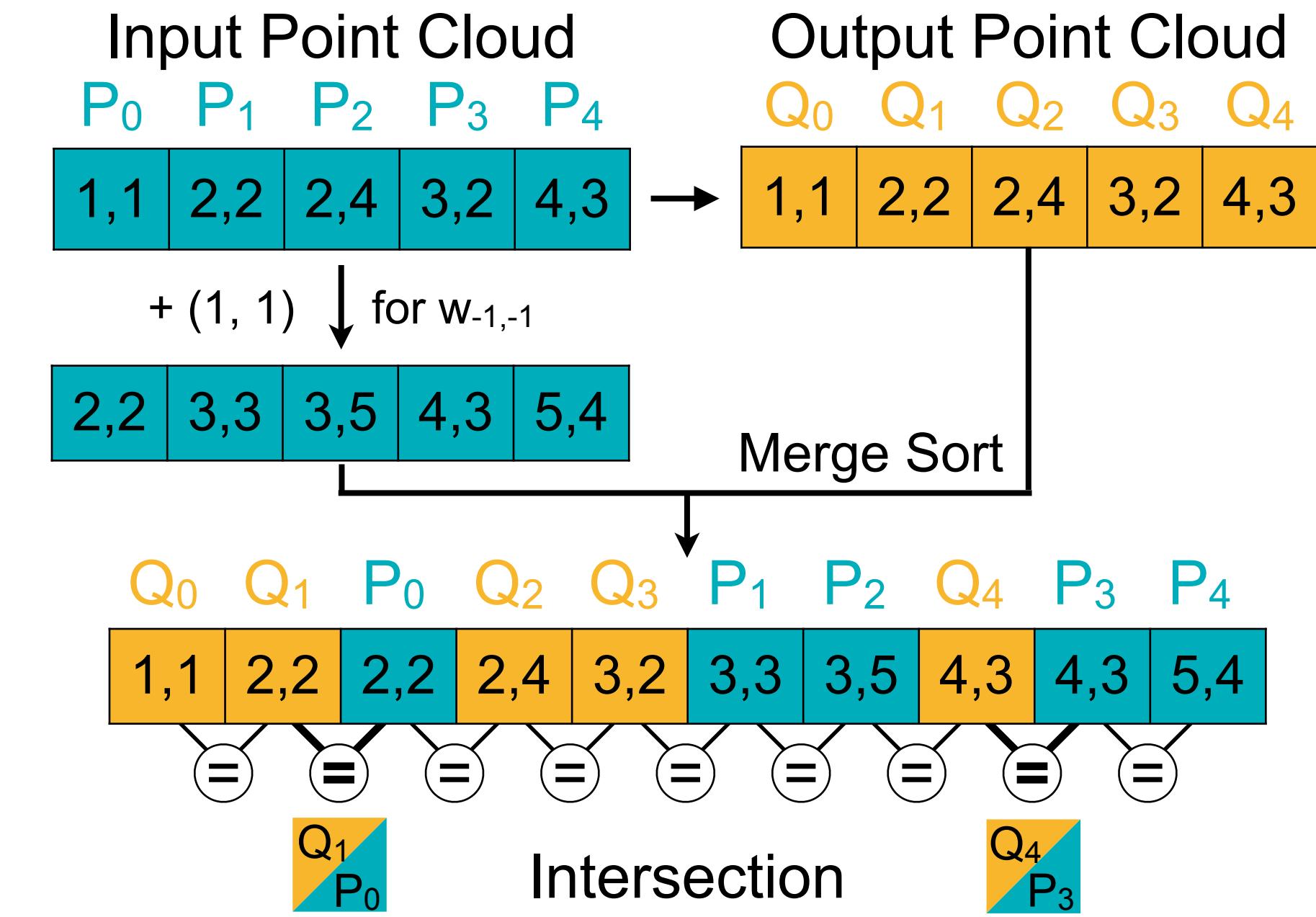
Q ₀				
	Q ₁		Q ₂	
		Q ₃		
			Q ₄	



Shift Input for W_{-1,-1}

W _{-1,-1}	W _{-1,0}	W _{-1,1}
W _{0,-1}	W _{0,0}	W _{0,1}
W _{1,-1}	W _{1,0}	W _{1,1}

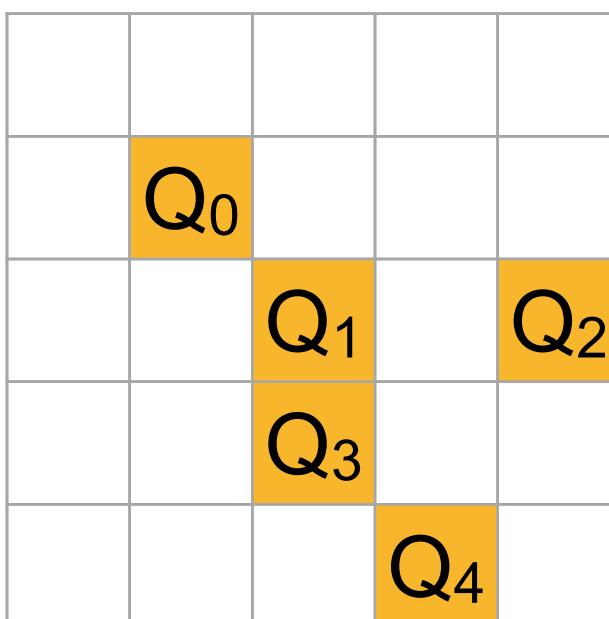
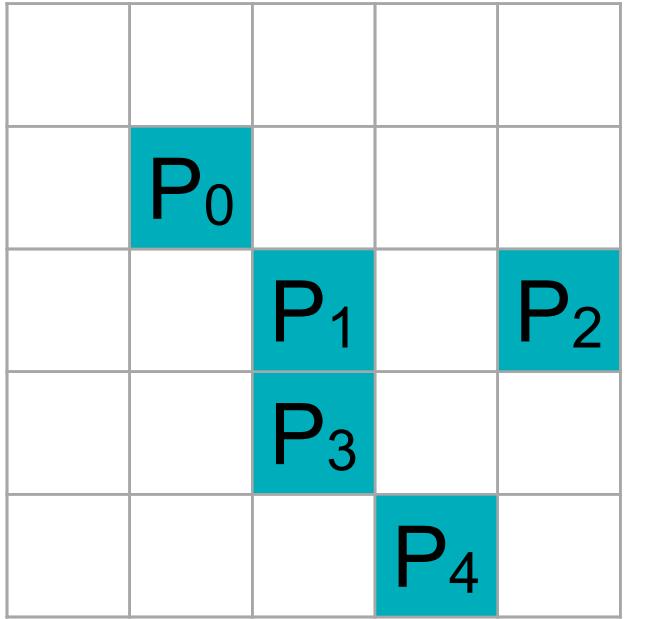
Output Point Cloud



Mapping Unit

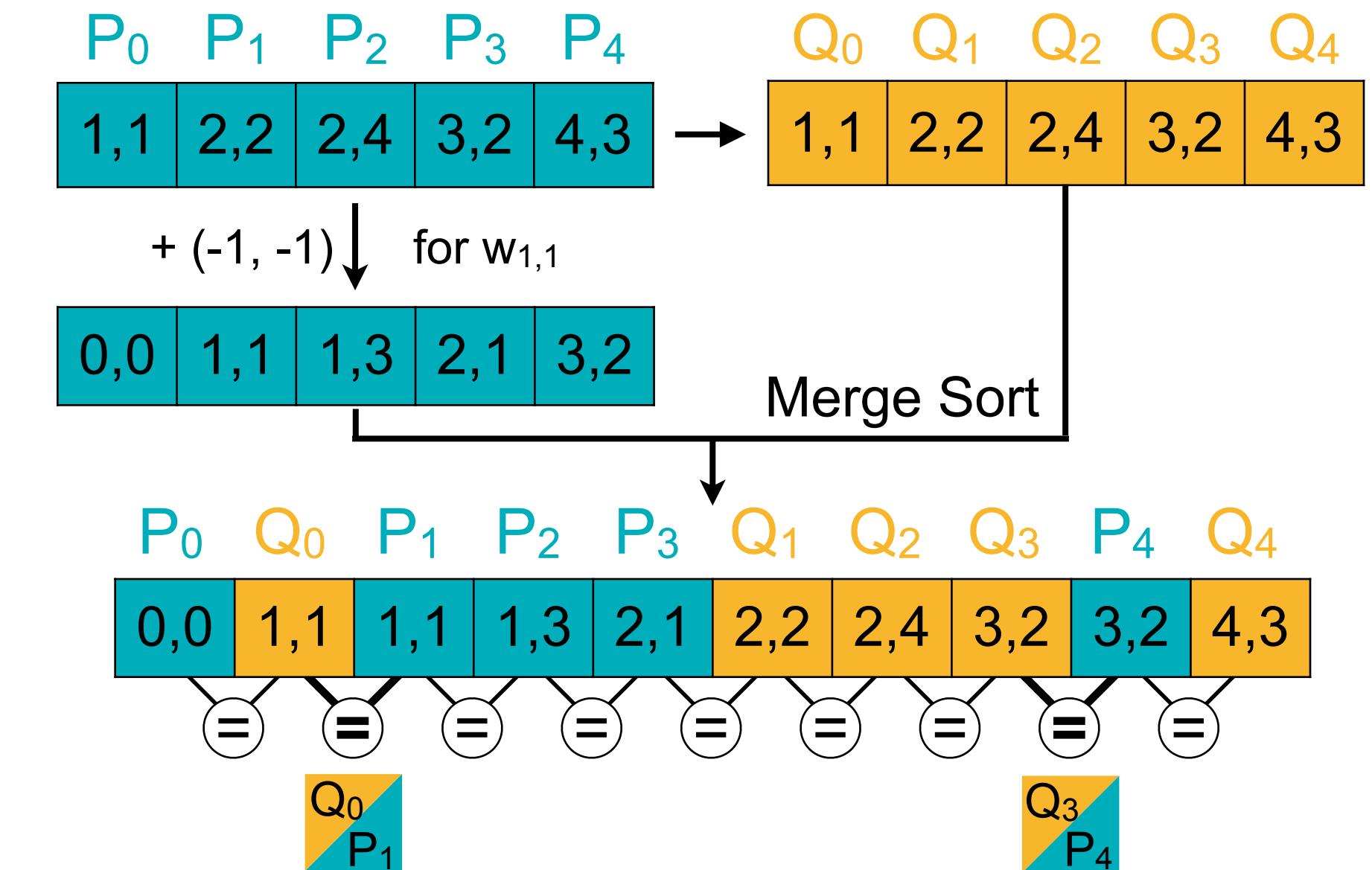
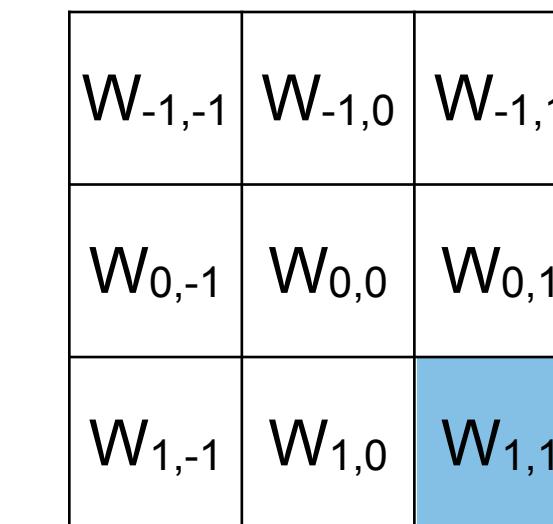
Merge sort can be used to find mappings in sparse convolution

Input Point Cloud



Output Point Cloud

Shift Input for $W_{1,1}$



(In, Out, Wgt)
(P₀, Q₁, W_{-1,-1})
(P₃, Q₄, W_{-1,-1})
...
(P₁, Q₀, W_{1,1})
(P₄, Q₃, W_{1,1})

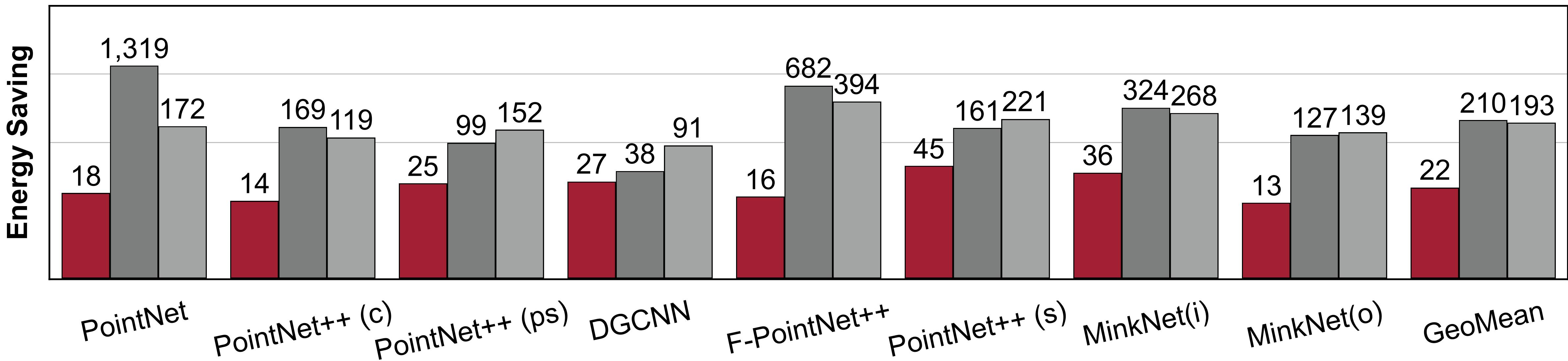
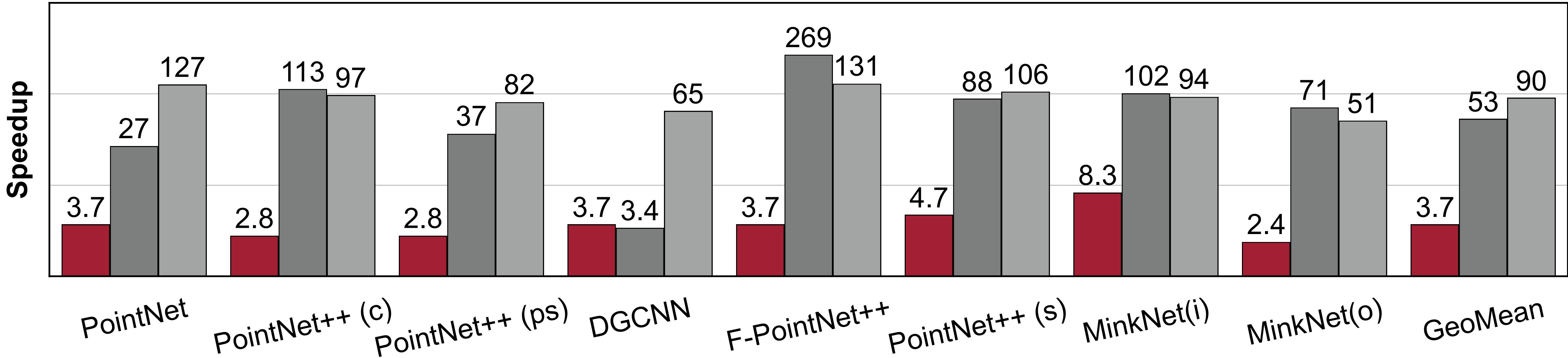
PointAcc: Efficient Point Cloud Accelerator [Lin et al., MICRO 2021]

PointAcc: Speedup and Energy Saving

over NVIDIA RTX 2080Ti

over Intel Xeon Skylake + TPU V3

over Intel Xeon Gold 6130

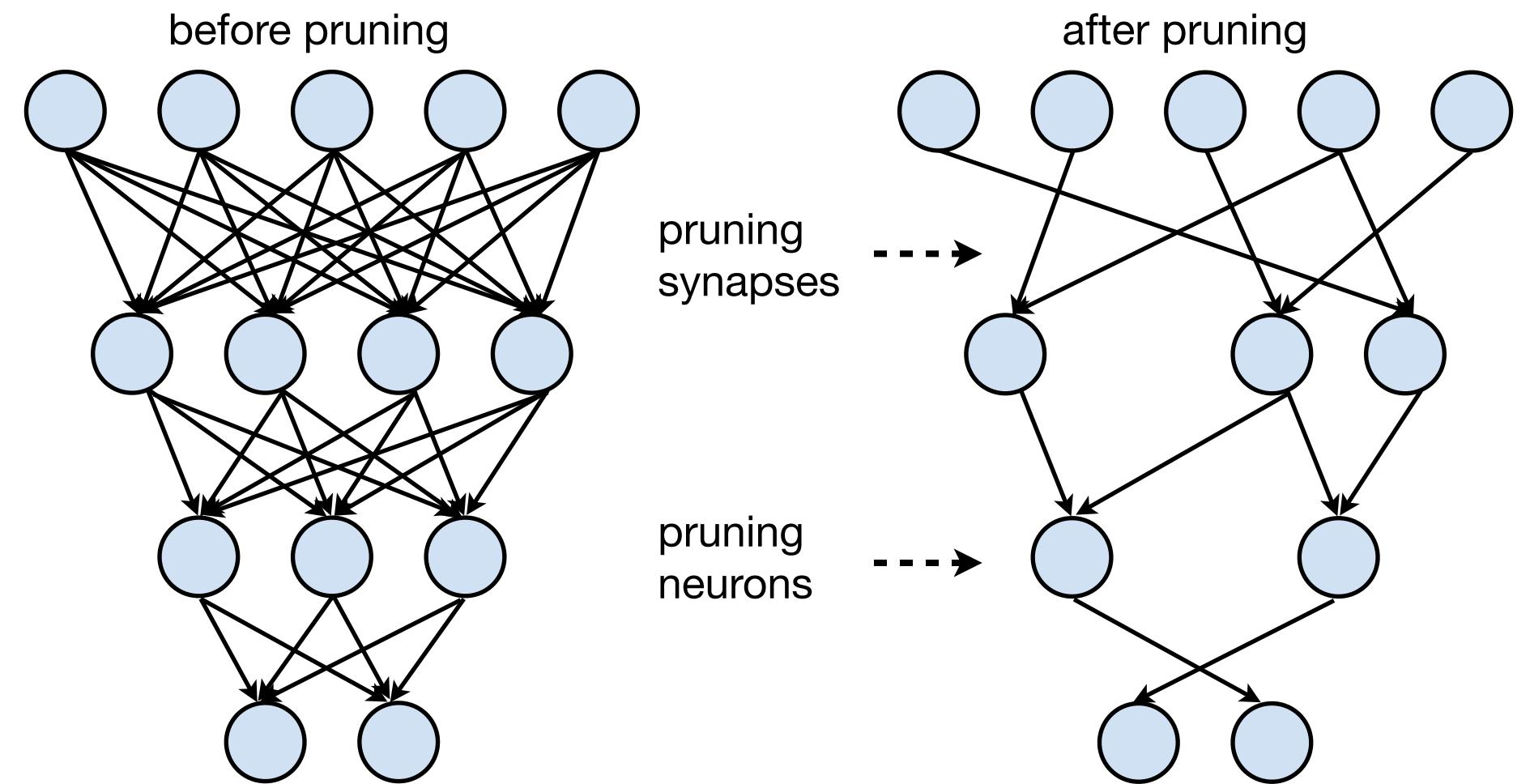


PointAcc: Efficient Point Cloud Accelerator [Lin et al., MICRO 2021]

Summary of Today's Lecture

In this lecture, we introduced:

- Automated ways to find pruning ratios
- System and hardware support for different granularities
- **We will cover in the next lecture:**
 - Numeric data types in modern computer systems
 - Basic concept of neural network quantization
 - Common neural network quantization methods



References

1. Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]
2. Exploring the granularity of sparsity in convolutional neural networks [Mao et al., CVPR-W]
3. Learning Structured Sparsity in Deep Neural Networks [Wen et al., NeurIPS 2016]
4. Learning Efficient Convolutional Networks through Network Slimming [Liu et al., ICCV 2017]
5. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang et al., ECCV 2018]
6. AMC: Automl for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]
7. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT
8. EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]
9. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han et al., FPGA 2017]
10. Block Sparse Format [NVIDIA, 2021]
11. Accelerating Sparse Deep Neural Networks [Mishra et al., arXiv 2021]