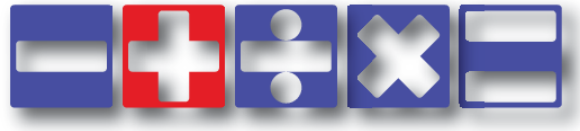
ARITH 2025



Hardware-Aware Training for Multiplierless Convolutional Neural Networks

Rémi Garcia, **Léo Pradels**, Silviu Filip, Olivier Sentieys

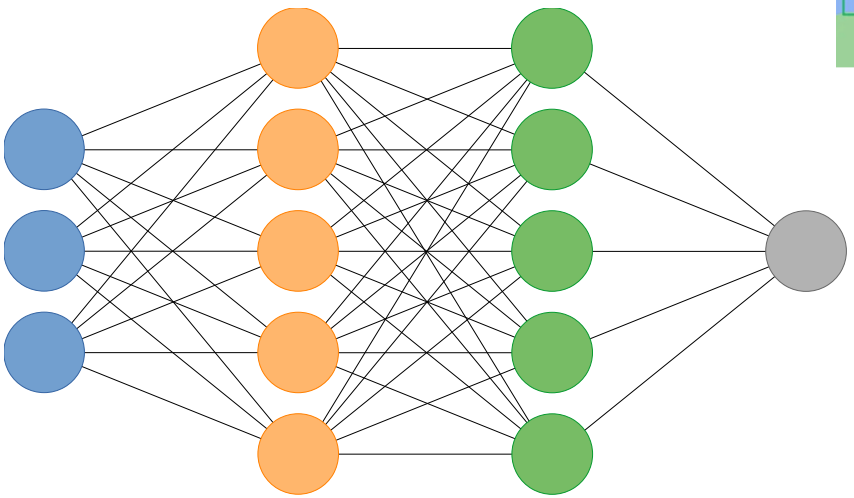
May, 5th 2025

Université de Rennes, IRISA, Inria

Strong need to optimize AI and CNN computations

Many applications:

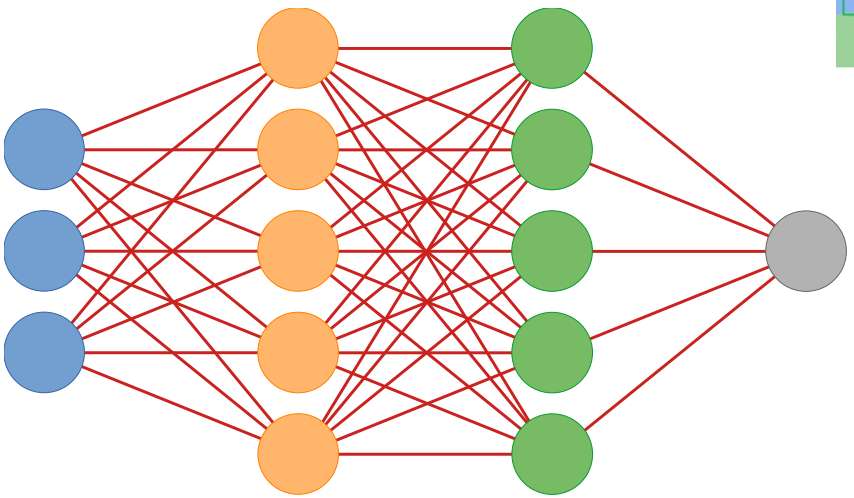
- Medical Imaging
- Audio Processing
- Object Detection
- Synthetic Data Generation



Strong need to optimize AI and CNN computations

Many applications:

- Medical Imaging
- Audio Processing
- Object Detection
- Synthetic Data Generation

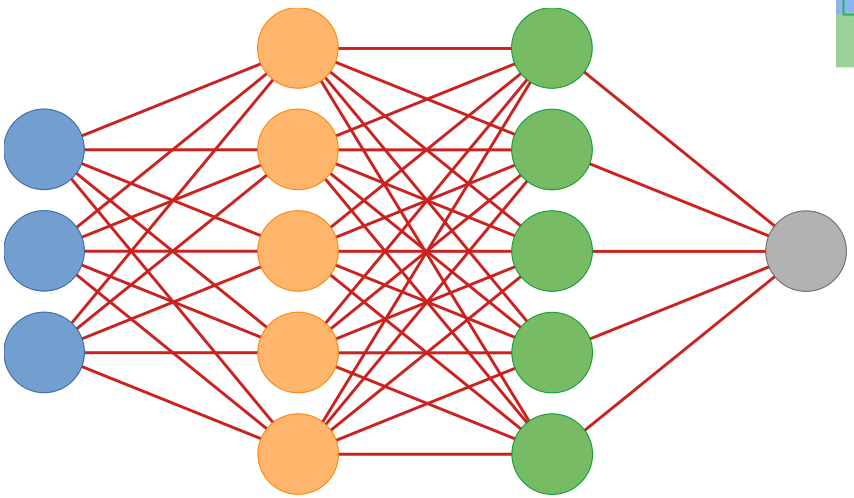


- More and more trainable parameters
  - → ResNet-18: >11M
  - → ResNet-152: >60M
  - → GPT-3: 175B!

Strong need to optimize AI and CNN computations

Many applications:

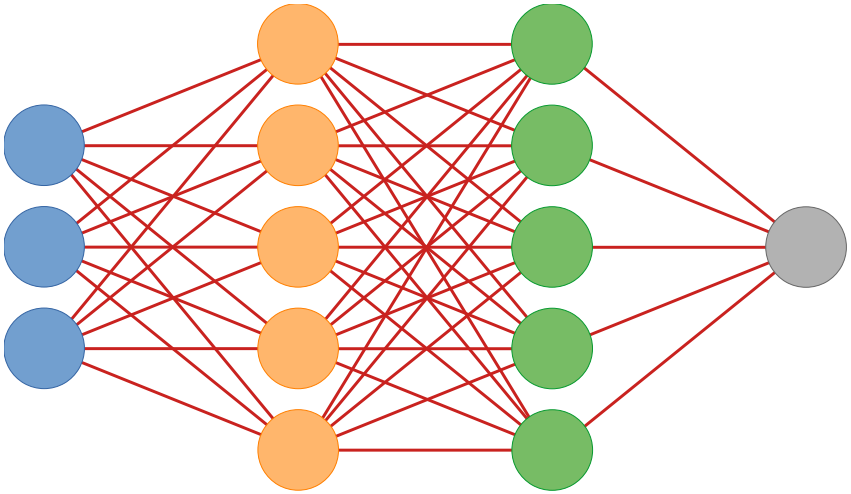
- Medical Imaging
- Audio Processing
- Object Detection
- Synthetic Data Generation



- More and more trainable parameters
  - → ResNet-18: >11M
  - → ResNet-152: >60M
  - → GPT-3: 175B!

... used in **multiplications**

Strong need to optimize AI and CNN computations

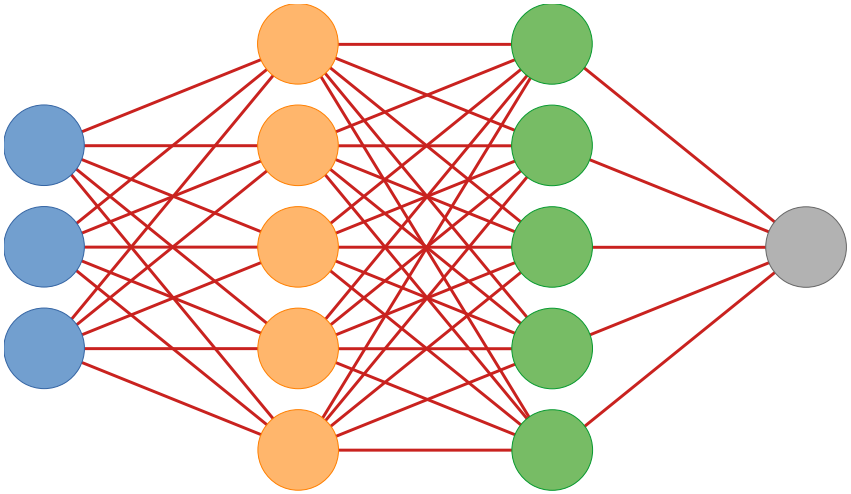


- More and more trainable parameters
  - → ResNet-18: $>$11M
  - → ResNet-152: $>$60M
  - → GPT-3: 175B!

... used in **multiplications**

FP32 multiplications are costly, in particular for **hardware implementation**
→ We target **FPGA**

Strong need to optimize AI and CNN computations



- More and more trainable parameters
  - → ResNet-18: $>$11M
  - → ResNet-152: $>$60M
  - → GPT-3: 175B!

... used in **multiplications**

FP32 multiplications are costly, in particular for **hardware implementation**
→ We target **FPGA**

A possible solution: Reducing precision through **quantization**
→ single precision (FP32) to smaller formats (FP16, INT8, INT4, etc.)

What is quantization?

Scaling and rounding to get quantized weights:

$$W_{\text{FxP}} = \lceil S \times W_{\text{FP}} \rfloor$$

Let b be the target precision and $[\alpha, \beta]$ the range of W_{FP}
The scaling factor S is a **parameter**:

$$S = \frac{\beta - \alpha}{2^b - 1}$$

To simplify it, we perform **symmetric** quantization

$$S = \frac{\max\left(|\alpha|, |\beta|\right)}{2^b - 1}$$

What is quantization?

Scaling and rounding to get quantized weights:

$$W_{\text{FxP}} = \lceil S \times W_{\text{FP}} \rfloor$$

Let b be the target precision and $[\alpha, \beta]$ the range of W_{FP}
The scaling factor S is a **parameter**:

$$S = \frac{\beta - \alpha}{2^b - 1}$$

To simplify it, we perform **symmetric** quantization

$$S = \frac{\max\left(|\alpha|, |\beta|\right)}{2^b - 1}$$

S is a floating-point value! This is expensive.
We use a **power of two** scaling factor instead:

$$S = \frac{2^{\lceil \log_2(\max(|\alpha|, |\beta|)) \rceil}}{2^b}$$

What is quantization?

Moreover, some quantization approaches **apply a function** on W_{FP}:

$$W_{\text{FxP}} = \lceil S \times f(W_{\text{FP}}) \rfloor$$

where

$$f(x) = \tanh(x) \qquad \text{for DoReFa-Net [ZWN+16]}$$

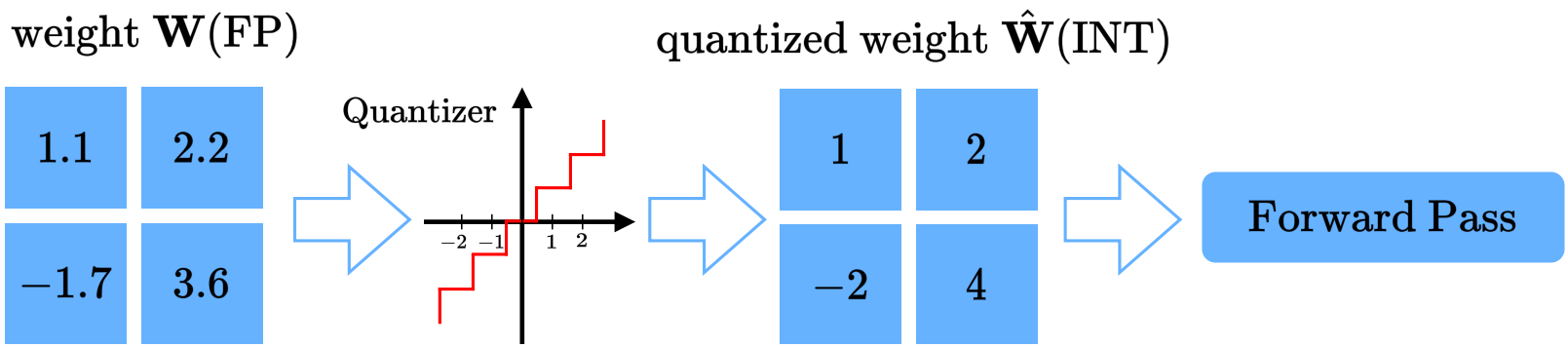
or

$$f(x) = \text{sign}(x) \times \log(|x|) \qquad \text{for LogQuant [GAGN15]}$$

Two main families of methods:

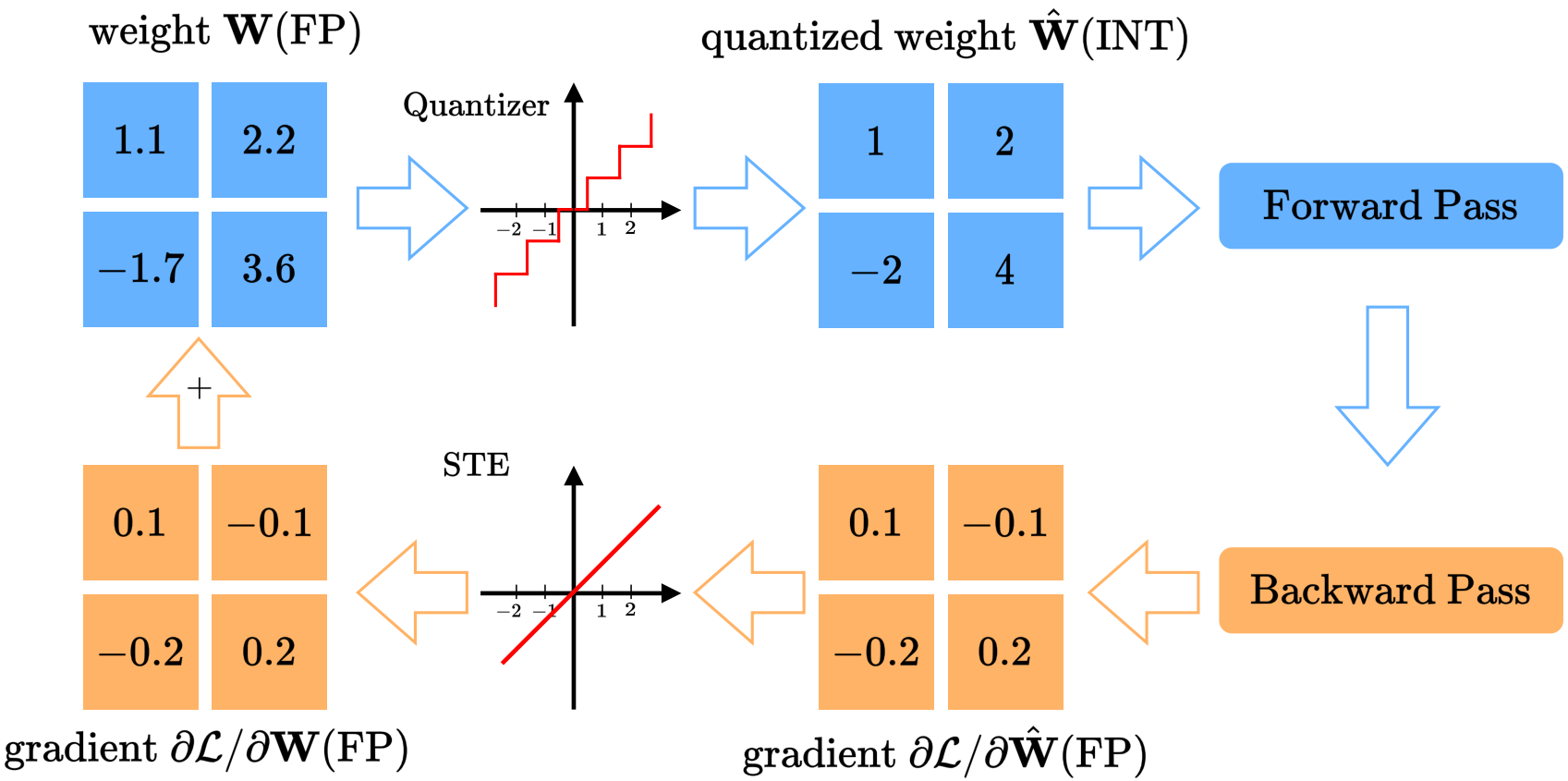
- **PTQ:** Post-Training Quantization, *i. e.*, FP32 training, then quantize
- **QAT:** Quantization-Aware Training, *i. e.*, update quantized signals during training

What is Quantization-Aware Training?



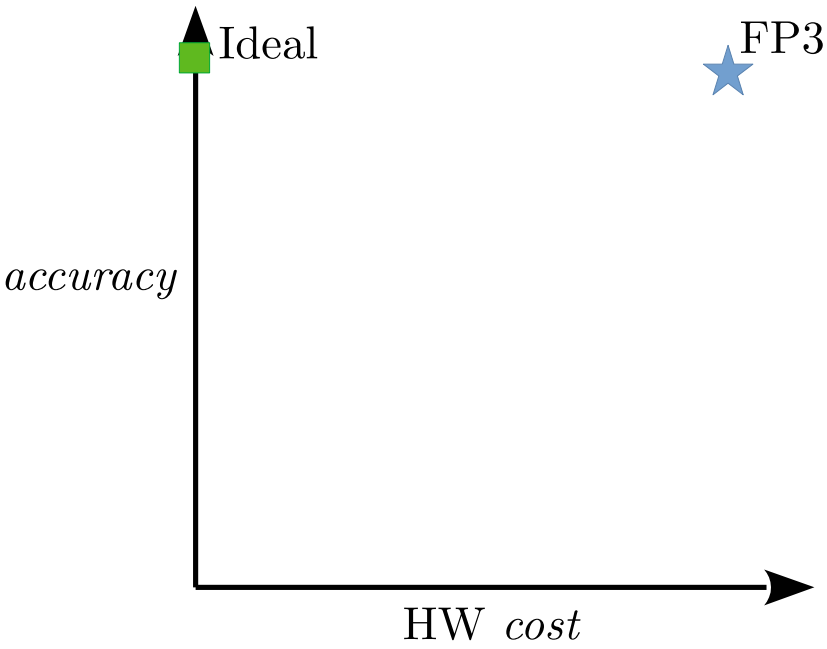
Schematic view of a QAT procedure with STE applied (adapted from [GKD+21])

What is Quantization-Aware Training?



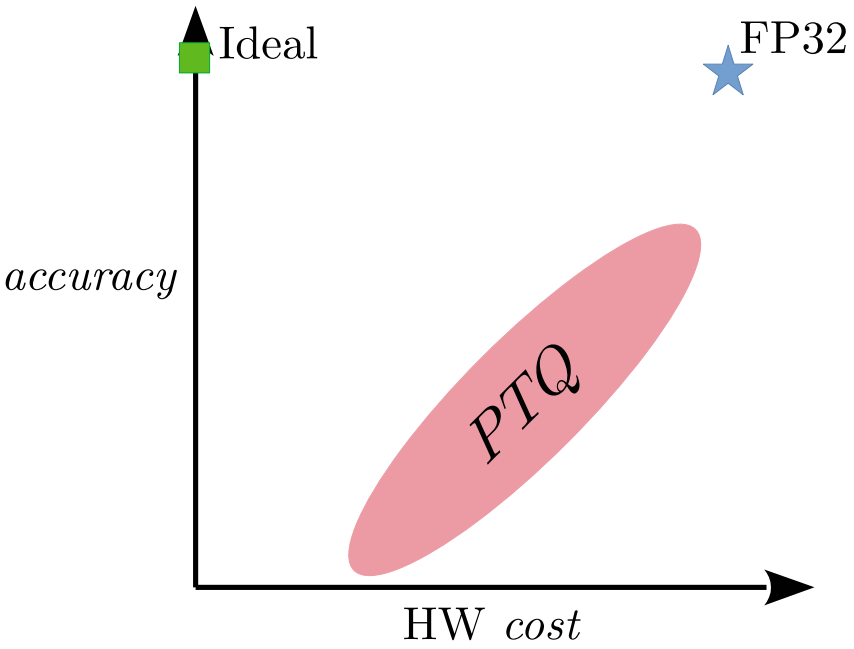
Schematic view of a QAT procedure with STE applied (adapted from [GKD+21])

How does quantization impact accuracy?



Quantization: a trade-off

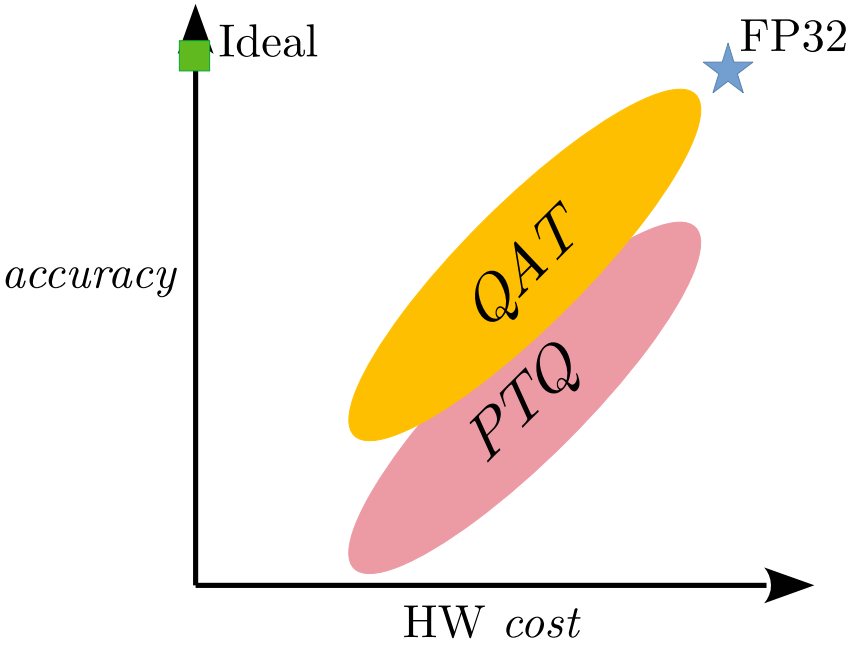
How does quantization impact accuracy?



Quantization: a trade-off

→ PTQ: smaller quantization leads to worse accuracy

How does quantization impact accuracy?

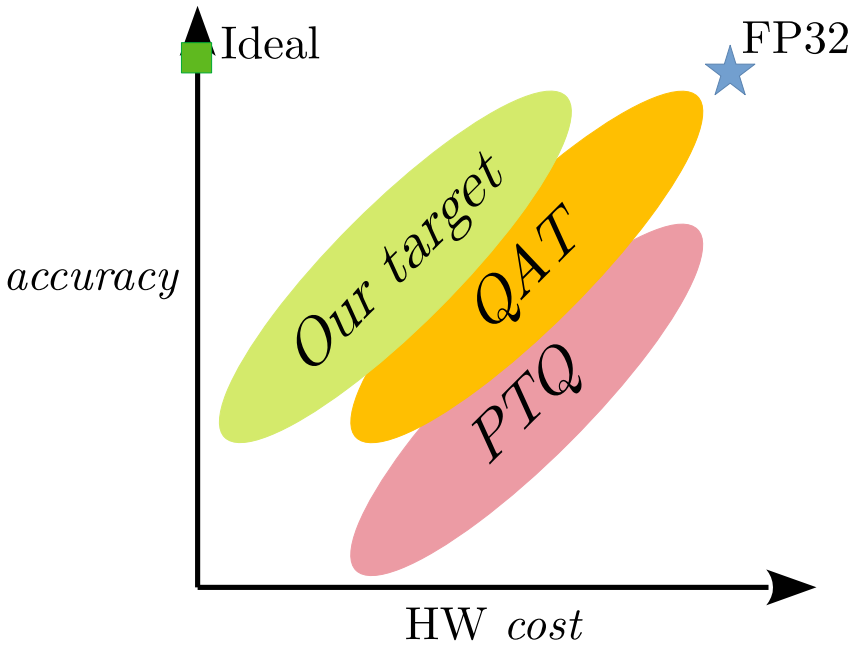


Quantization: a trade-off

→ PTQ: smaller quantization leads to worse accuracy

→ QAT: better accuracy

How does quantization impact accuracy?



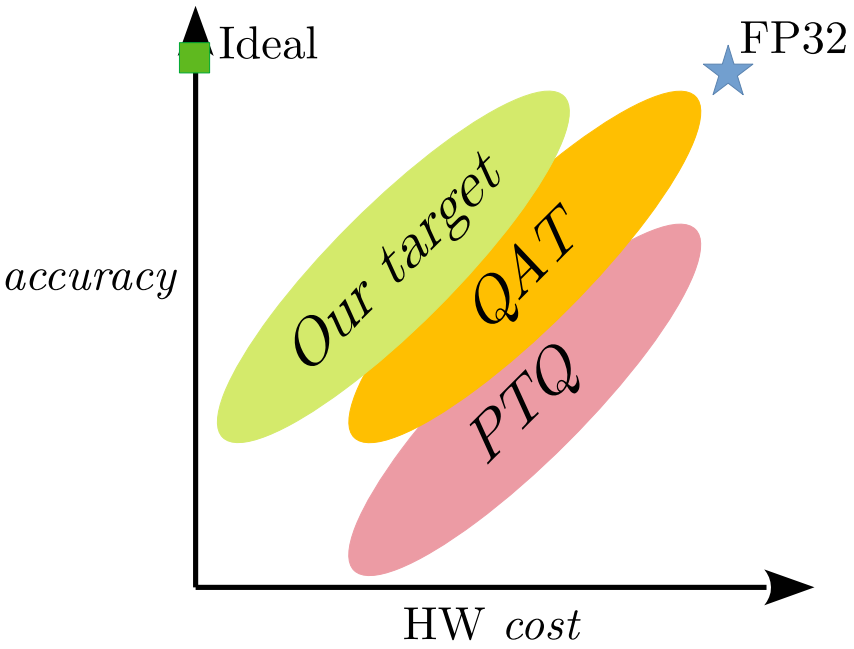
Quantization: a trade-off

→ PTQ: smaller quantization leads to worse accuracy

→ QAT: better accuracy

Our goal: taking the implementation cost into account **during training**

How does quantization impact accuracy?



Quantization: a trade-off

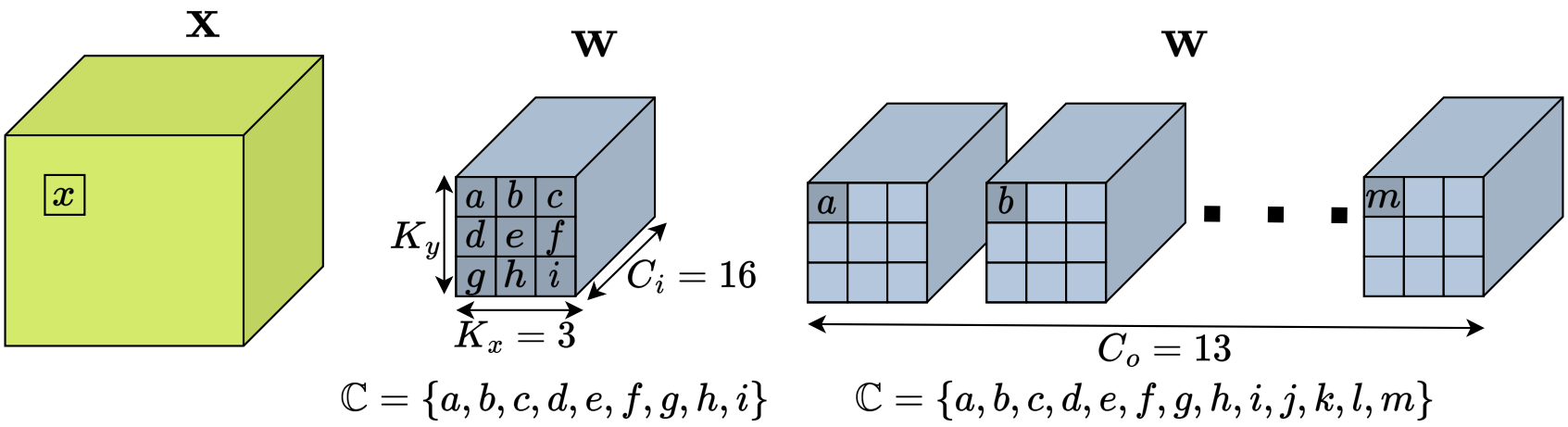
→ PTQ: smaller quantization leads to worse accuracy

→ QAT: better accuracy

Our goal: taking the implementation cost into account **during training**

We are not the first to consider hardware implementation cost during the training step [ECS+21, HKKV22, HHW+23].

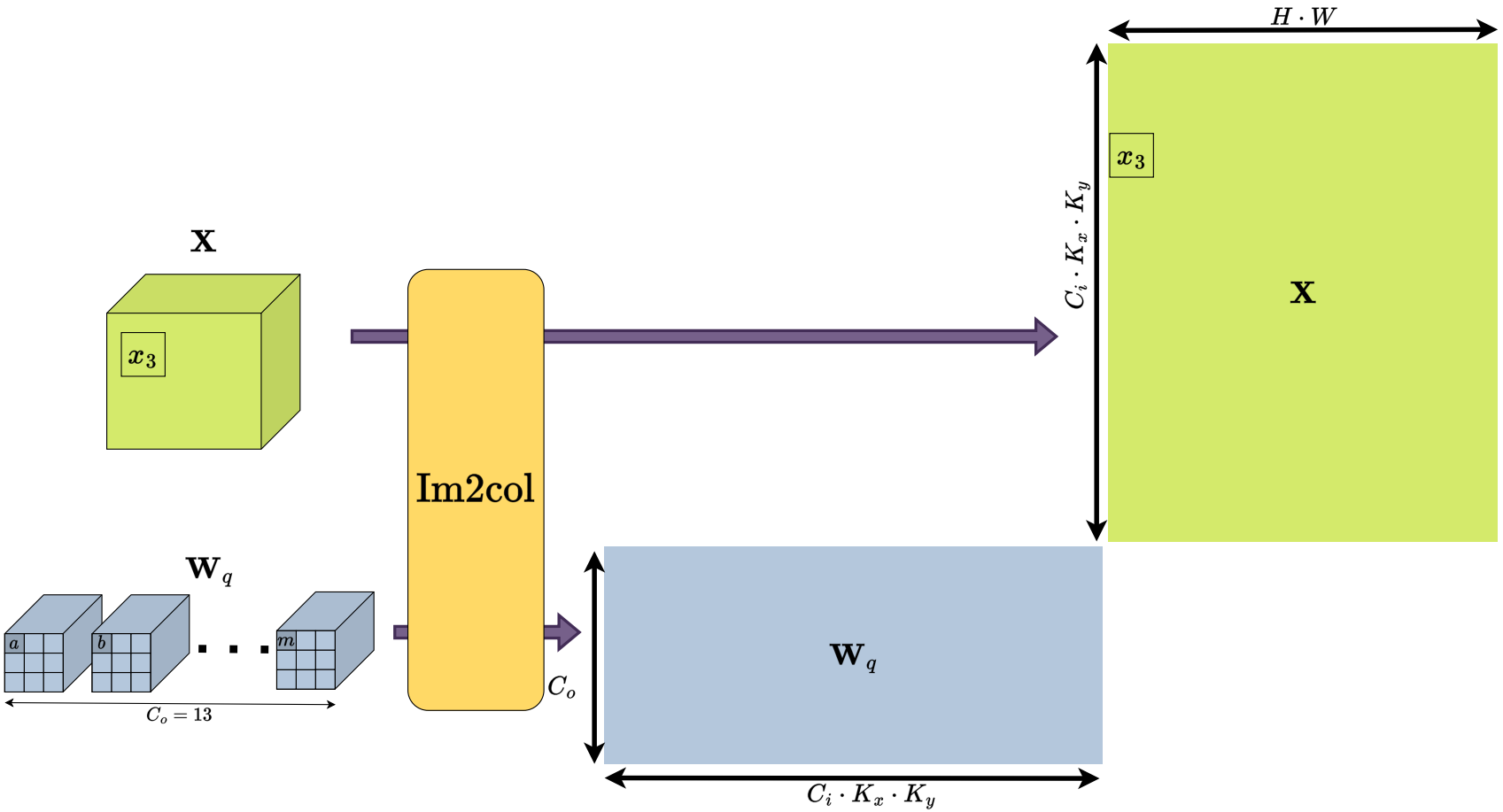
After we quantize: the implementation



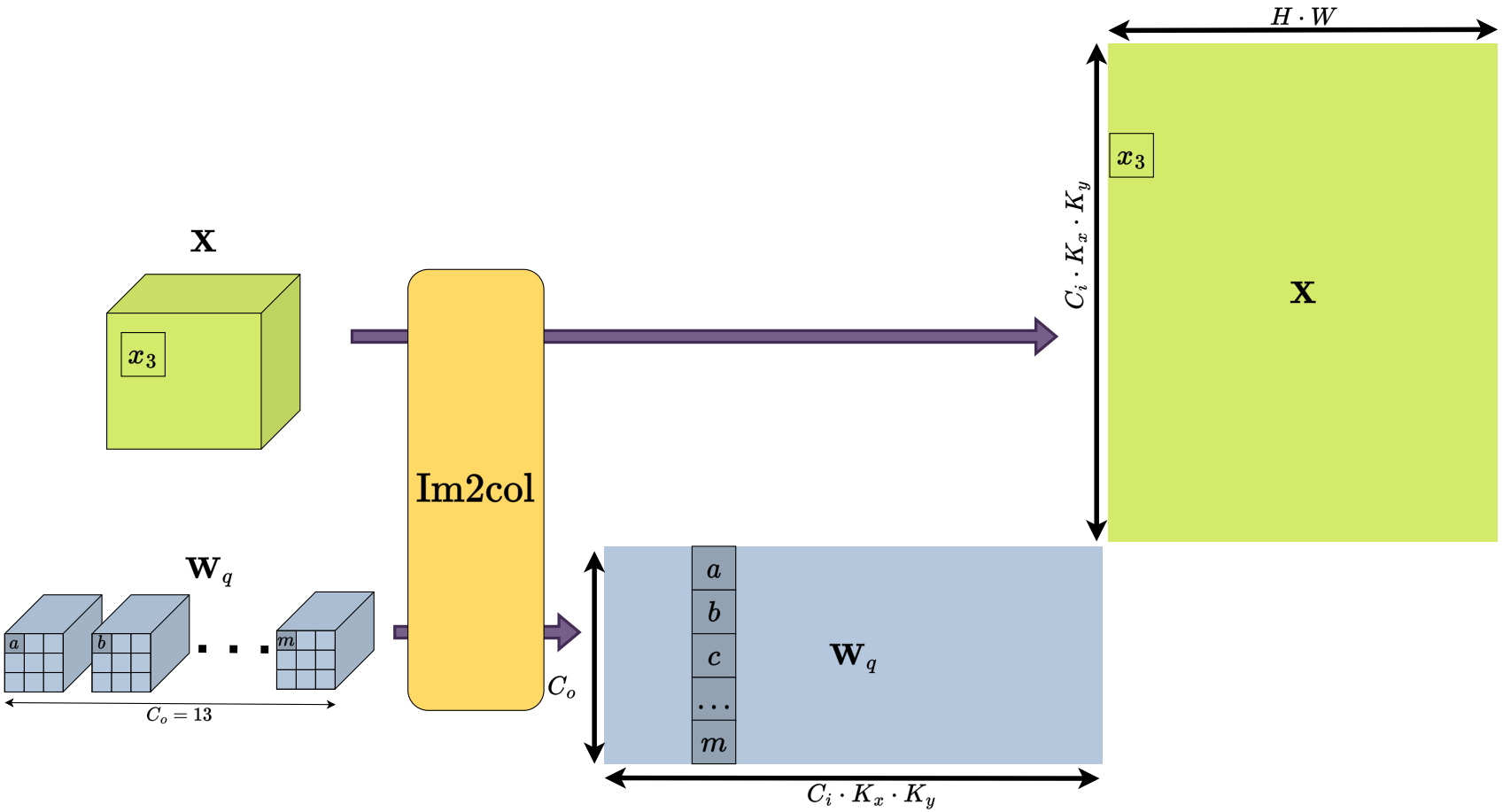
(a) Kernel Parallelization

(b) Filter Parallelization

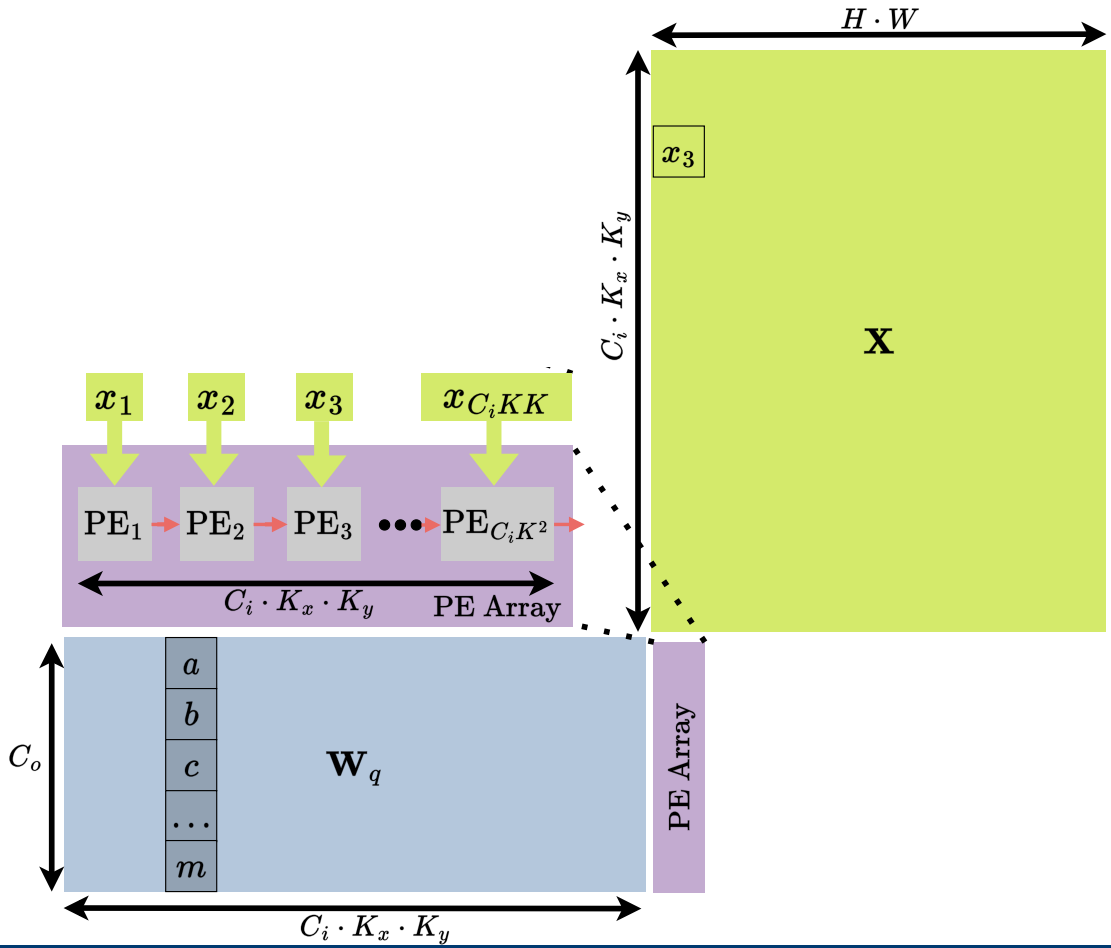
After we quantize: the implementation



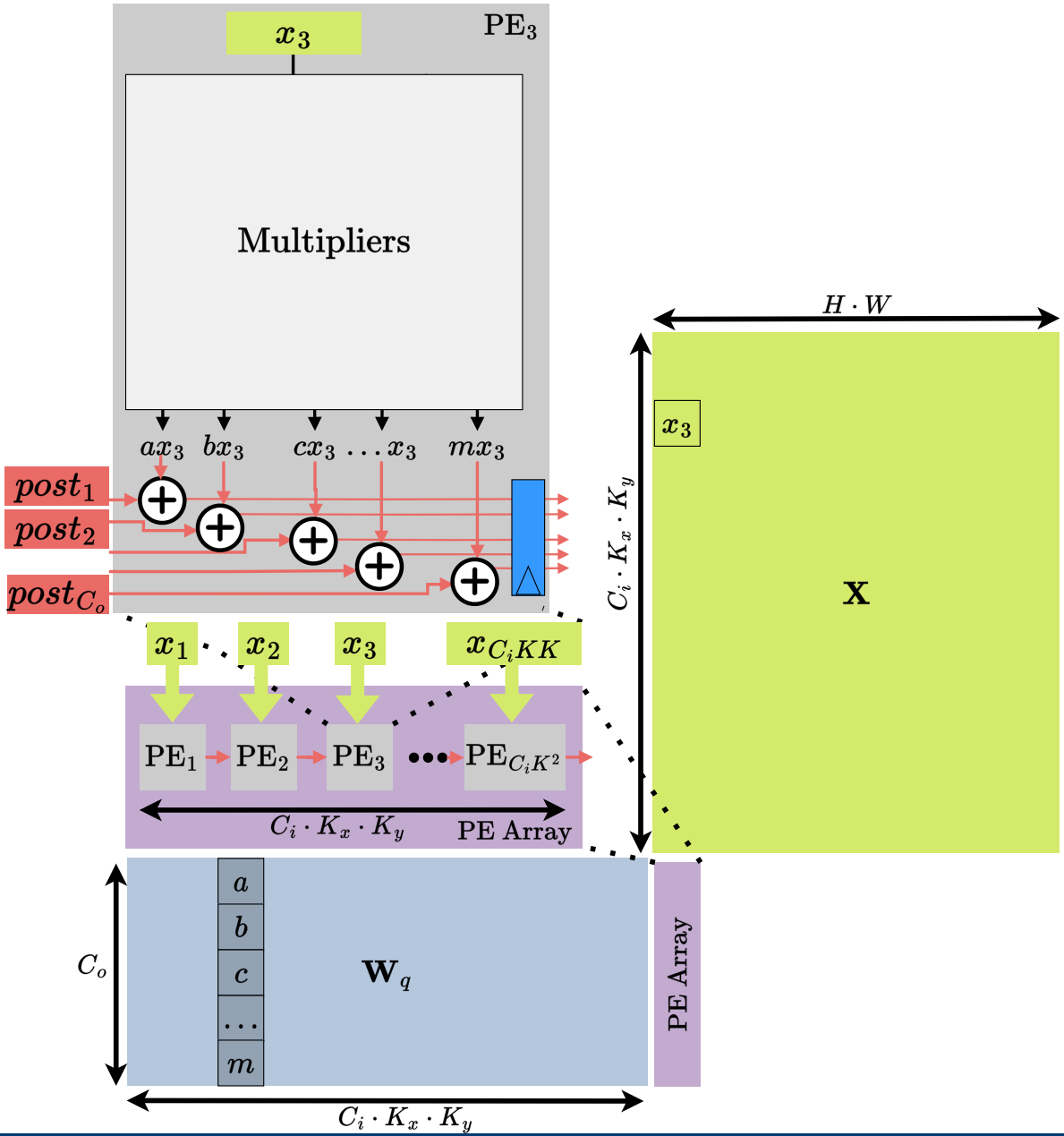
After we quantize: the implementation



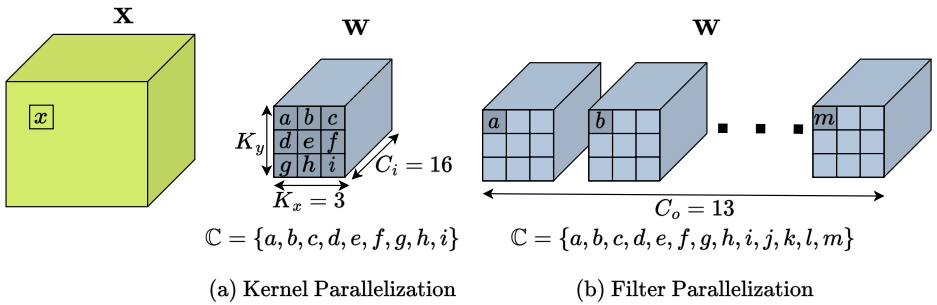
After we quantize: the implementation



After we quantize: the implementation

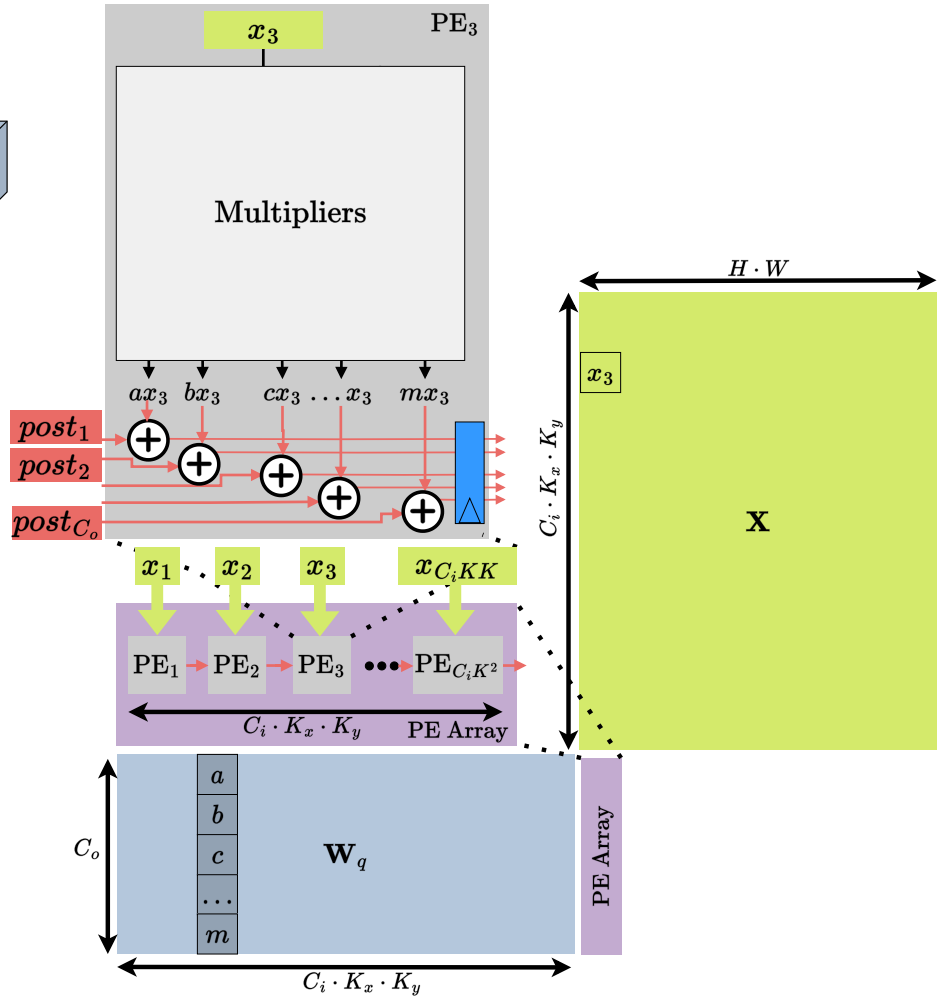


After we quantize: the implementation



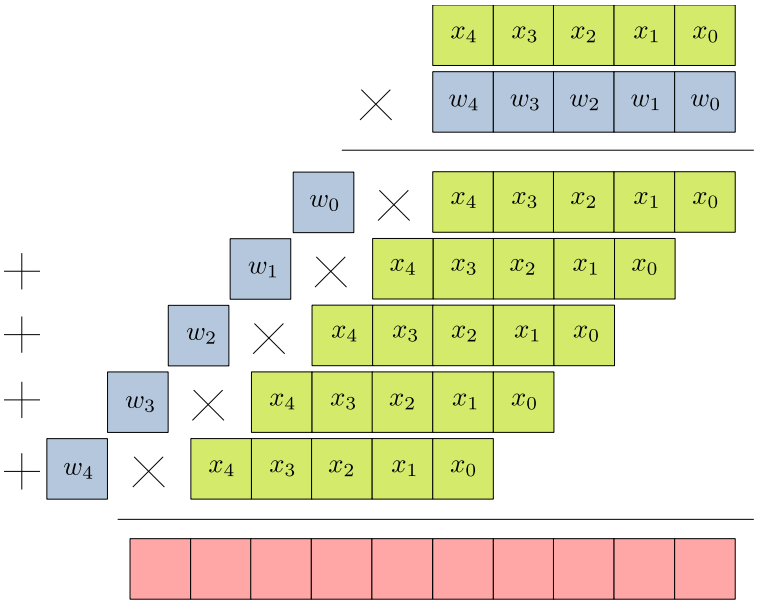
(a) Kernel Parallelization (b) Filter Parallelization

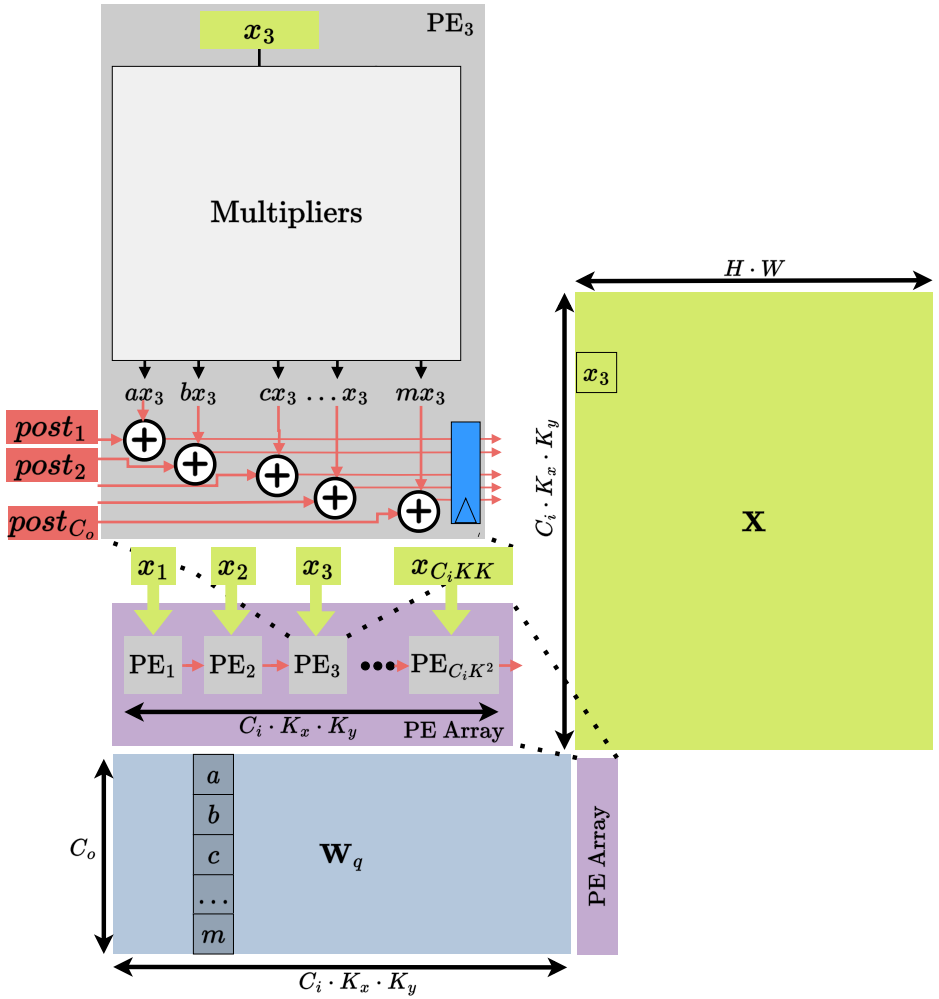
- Systolic Array with a **weight stationary** dataflow design: a trade-off between parallelism and hardware cost



After we quantize: the implementation

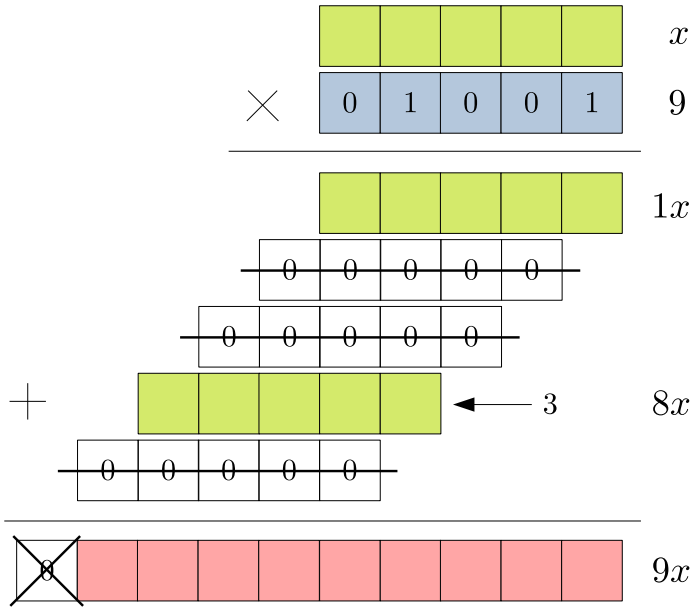
Generic multiplication during training

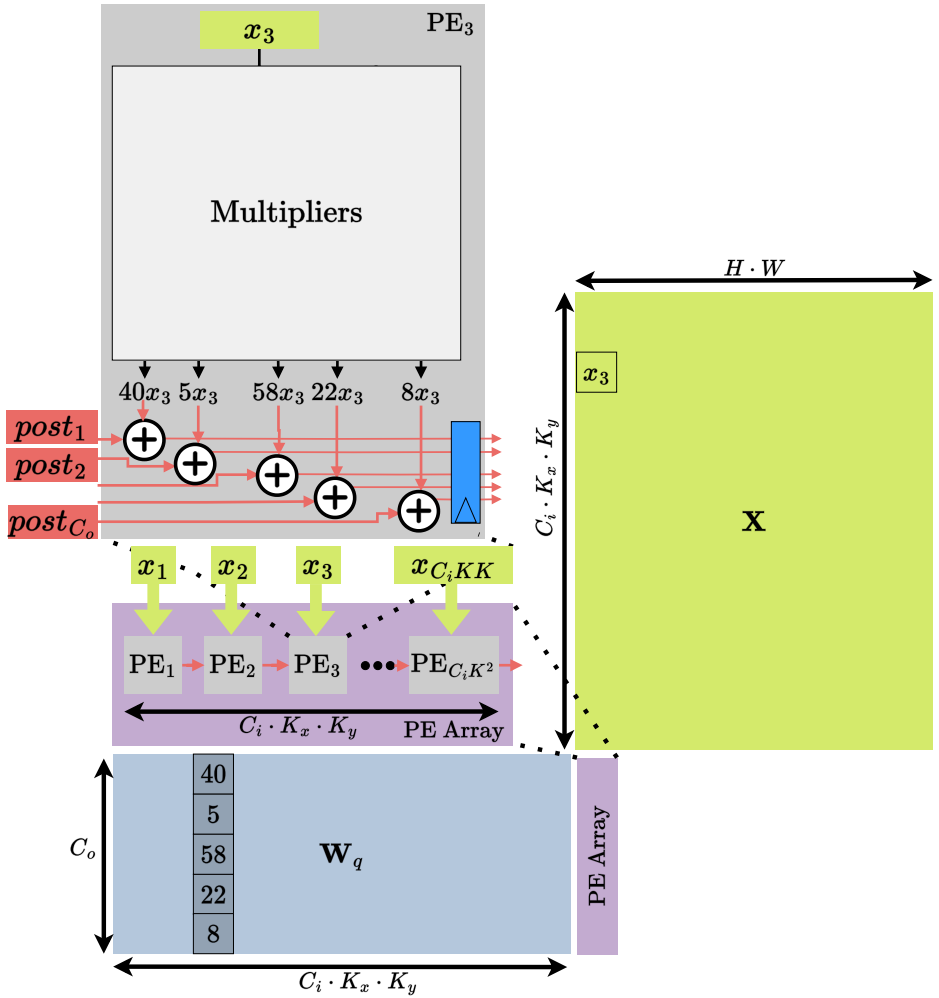




After we quantize: the implementation

Constant multiplication
for hardware implementation





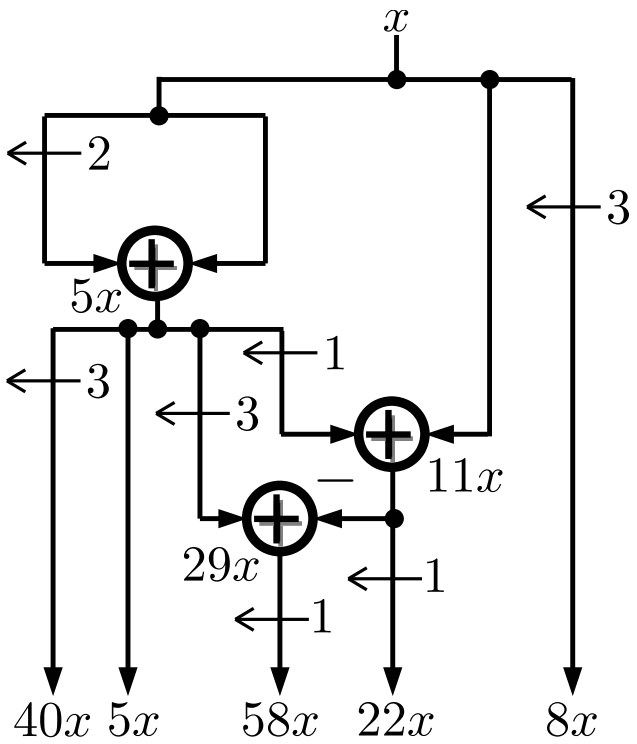
Hardware implementation: the MCM problem

Multiplying a variable with multiple constants, the so-called **Multiple Constant Multiplication** (MCM) problem

Hardware implementation: the MCM problem

Multiplying a variable with multiple constants, the so-called **Multiple Constant Multiplication** (MCM) problem

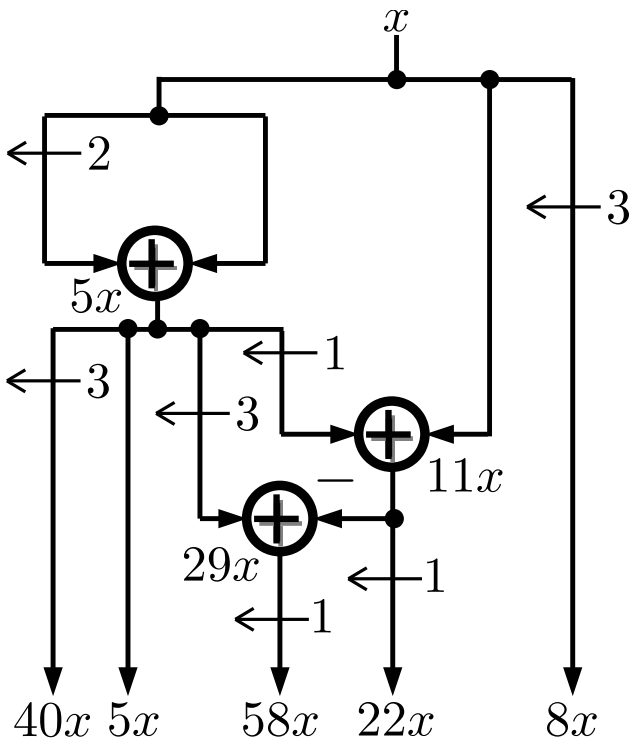
Example of an adder graph to compute $\{40, 5, 58, 22, 8\} \times x$:



Hardware implementation: the MCM problem

Multiplying a variable with multiple constants, the so-called **Multiple Constant Multiplication** (MCM) problem

Example of an adder graph to compute $\{40, 5, 58, 22, 8\} \times x$:

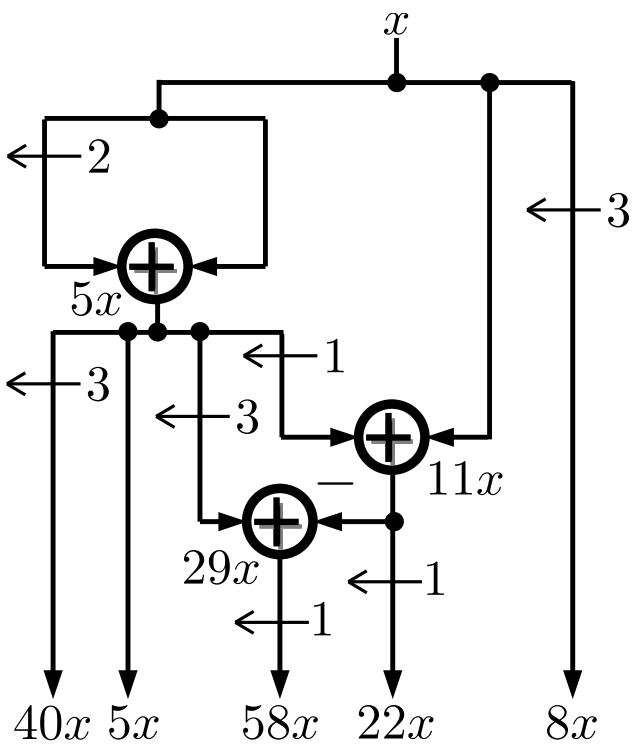


$\rightarrow$ 5 generic multiplications vs 3 adders

Hardware implementation: the MCM problem

Multiplying a variable with multiple constants, the so-called **Multiple Constant Multiplication** (MCM) problem

Example of an adder graph to compute $\{40, 5, 58, 22, 8\} \times x$:

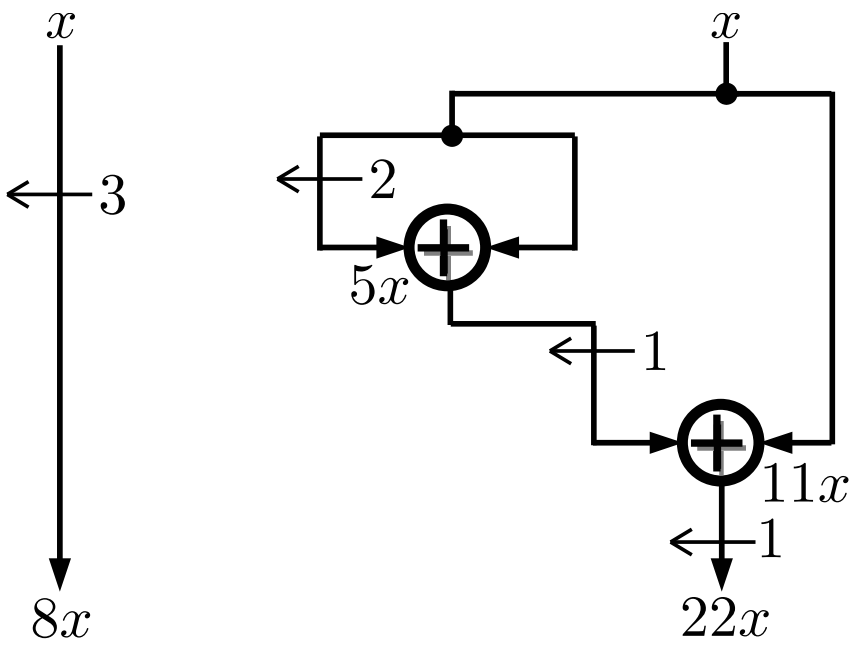


$\rightarrow$ 5 generic multiplications vs 3 adders

There is an ILP model to obtain minimal cost adder graphs [GV23]

Hardware implementation: using adder graphs

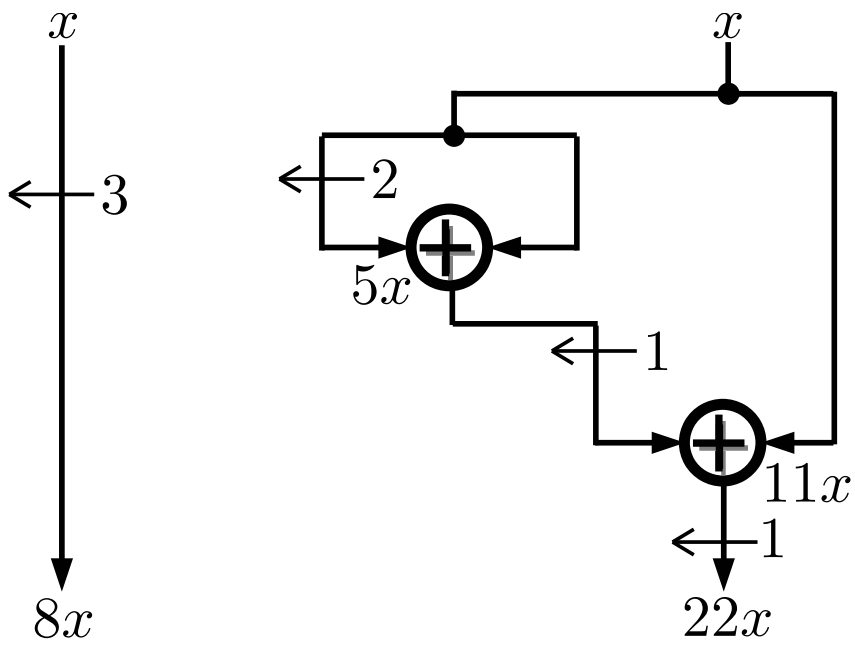
Shift-and-add implementations:



$\rightarrow$ Constant multiplication cost depends on the constant

Hardware implementation: using adder graphs

Shift-and-add implementations:

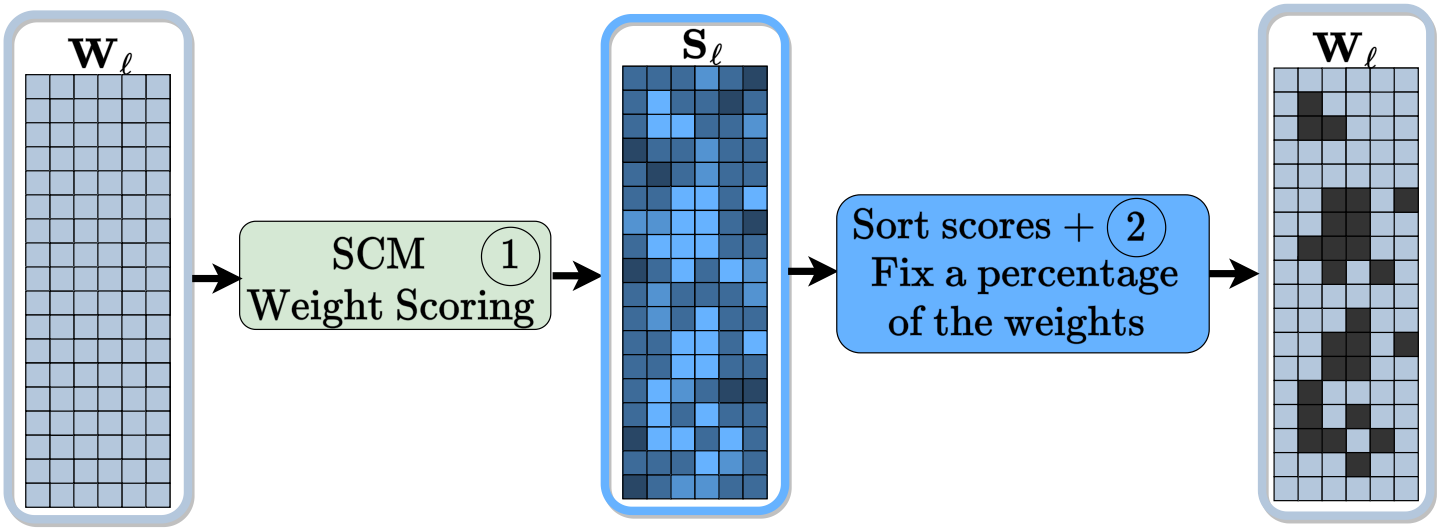


$\rightarrow$ Constant multiplication cost depends on the constant

First idea: during training, at the quantization step, prefer constants with smaller adder graph cost, *e. g.*, only power of 2 [ECS+21]

Progressive Adder-Aware Training

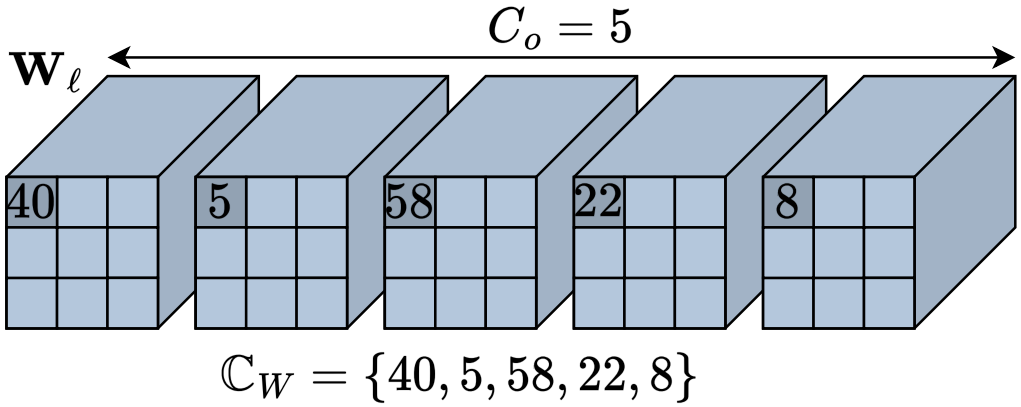
Our idea: Do not limit weight possibilities but **progressively** fix weights during training instead

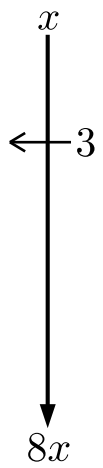


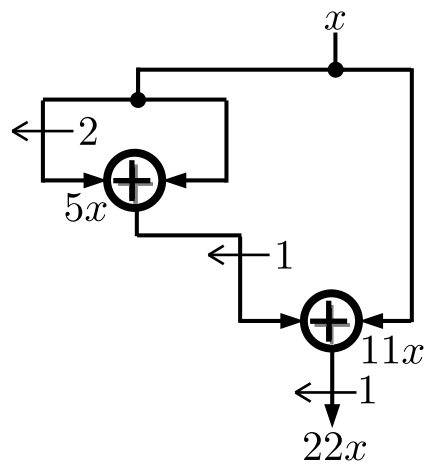
Key elements:

- Every N training steps, fix some weights
- Fixed weights are chosen from their implementation cost
  $\rightarrow$ power of two will be fixed first
- Implementation cost takes into account previously fixed weights

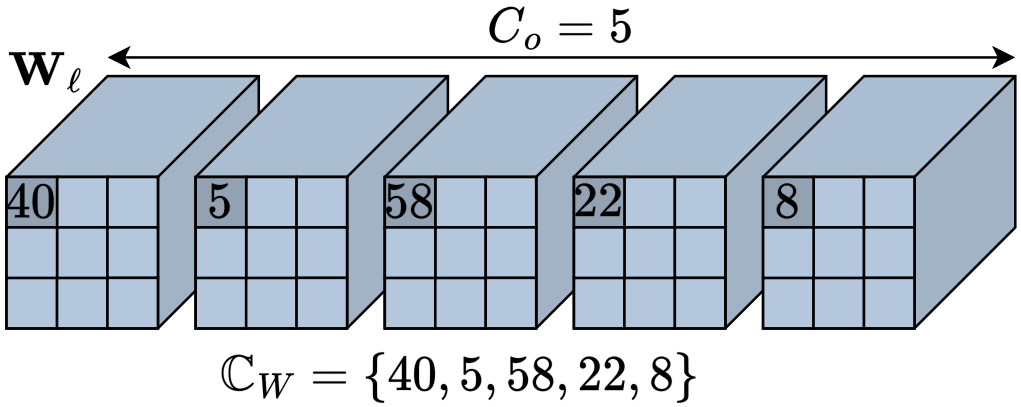
Progressive Adder-Aware Training: Computing Scores







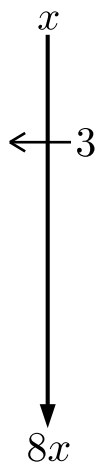
Progressive Adder-Aware Training: Computing Scores

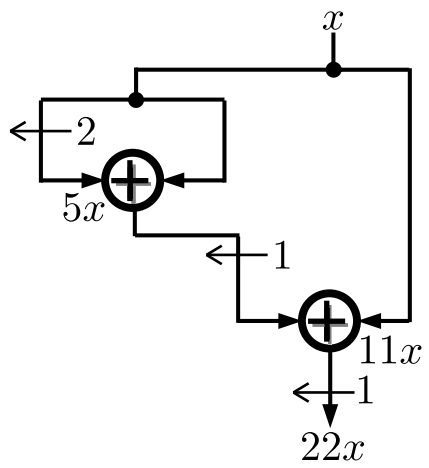


$$\mathbb{C}_W = \{40, 5, 58, 22, 8\}$$

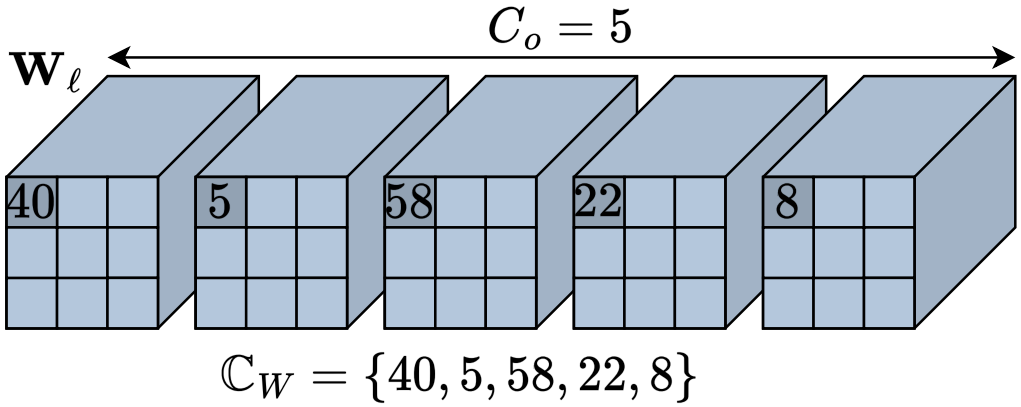
SCM Score:

$$\{1, 1, 2, 2, 0\}$$

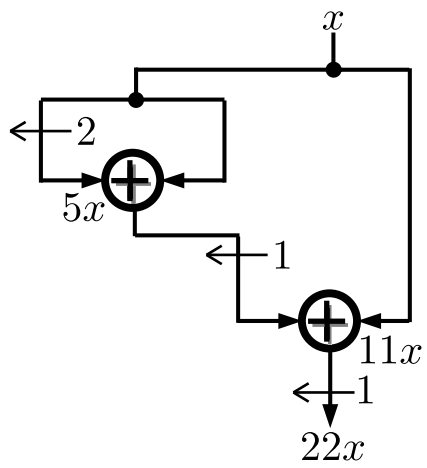




Progressive Adder-Aware Training: Computing Scores



$$\mathbb{C}_W = \{40, 5, 58, 22, 8\}$$



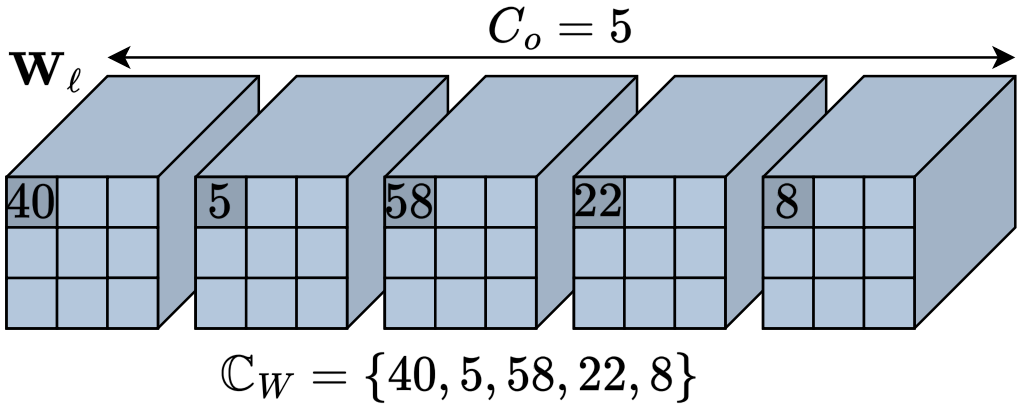
SCM Score:

$$\{1, 1, 2, 2, 0\}$$

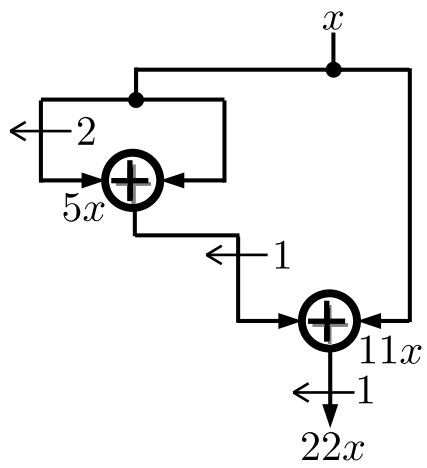
Suppose that 5 has been fixed previously:

$$\{\mathbf{0}, -, 2, \mathbf{1}, 0\}$$

Progressive Adder-Aware Training: Computing Scores



$$\mathbb{C}_W = \{40, 5, 58, 22, 8\}$$



SCM Score:

$$\{1, 1, 2, 2, 0\}$$

Suppose that 5 has been fixed previously:

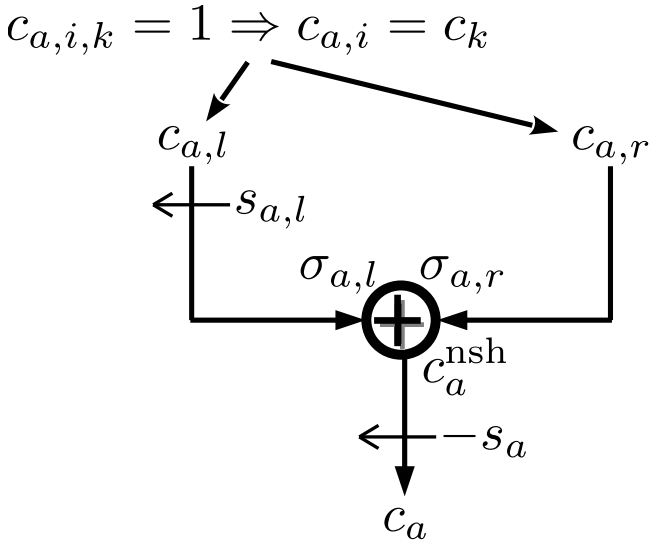
$$\{\mathbf{0}, -, 2, \mathbf{1}, 0\}$$

For basic score, we can use a standard SCM approach
$\rightarrow$ for example, an ILP model [GV23]
To consider fixed weights, we need to solve a **new problem**
$\rightarrow$ we **extend** the SCM model

Progressive Adder-Aware Training: SCM inspired model

The key of the model is "adders" modeling:

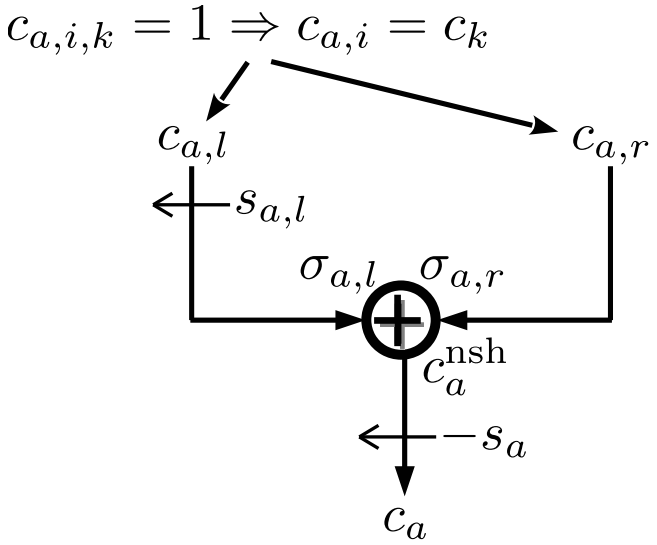
$$\underbrace{2^{s_a} c_a}_{c_a^{\text{nsh}}} = \underbrace{(-1)^{\sigma_{a,l}} \underbrace{2^{s_{a,l}} c_{a,l}}_{c_{a,l}^{\text{sh}}}}_{c_{a,l}^{\text{sh,sg}}} + \underbrace{(-1)^{\sigma_{a,r}} \underbrace{c_{a,r}}_{c_{a,r}^{\text{sh}}}}_{c_{a,r}^{\text{sh,sg}}}$$

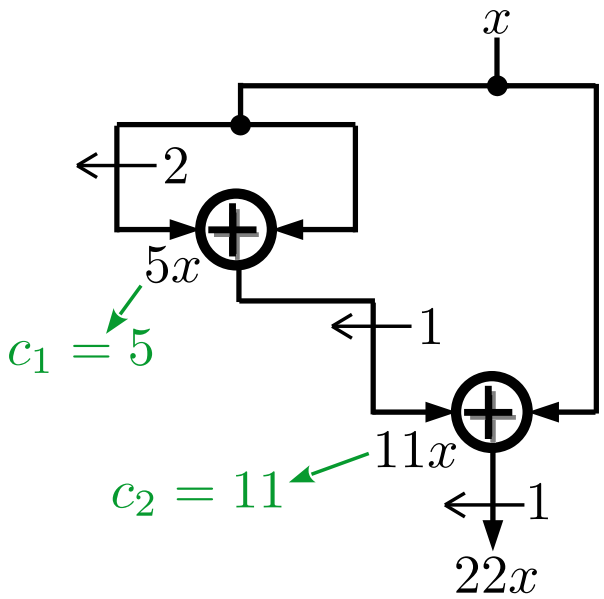


Progressive Adder-Aware Training: SCM inspired model

The key of the model is "adders" modeling:

$$\underbrace{2^{s_a} c_a}_{c_a^{\text{nsh}}} = \underbrace{\underbrace{(-1)^{\sigma_{a,l}} \underbrace{2^{s_{a,l}} c_{a,l}}_{c_{a,l}^{\text{sh}}}}_{c_{a,l}^{\text{sh,sg}}} + \underbrace{(-1)^{\sigma_{a,r}} \underbrace{c_{a,r}}_{c_{a,r}^{\text{sh}}}}_{c_{a,r}^{\text{sh,sg}}}}$$

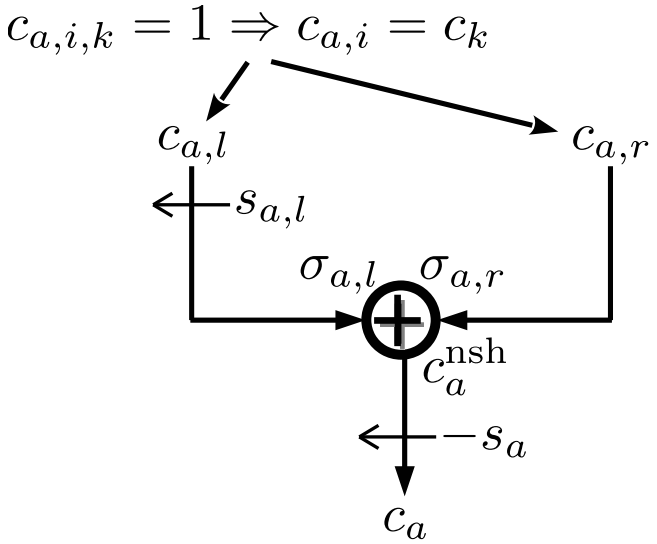


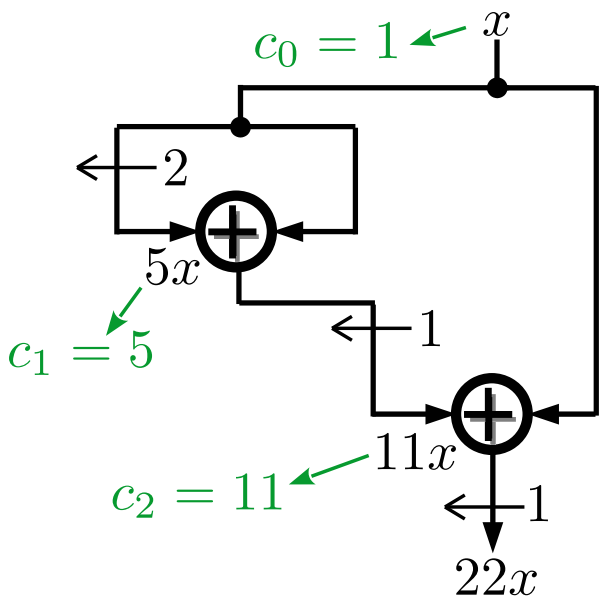


Progressive Adder-Aware Training: SCM inspired model

The key of the model is "adders" modeling:

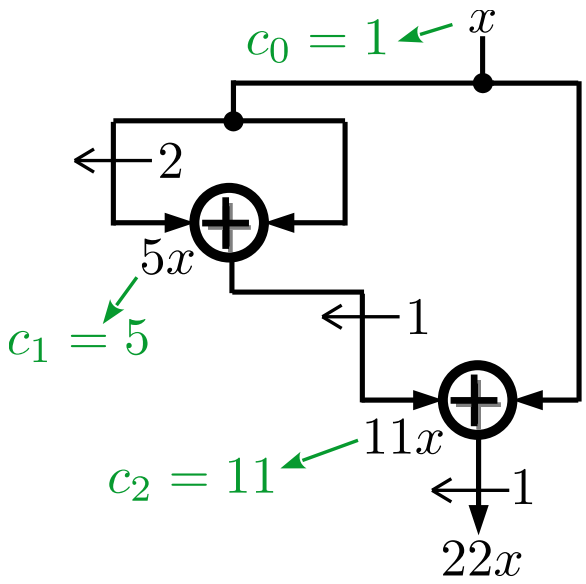
$$\underbrace{2^{s_a} c_a}_{c_a^{\text{nsh}}} = \underbrace{\underbrace{(-1)^{\sigma_{a,l}} \underbrace{2^{s_{a,l}} c_{a,l}}_{c_{a,l}^{\text{sh}}}}_{c_{a,l}^{\text{sh,sg}}} + \underbrace{(-1)^{\sigma_{a,r}} \underbrace{c_{a,r}}_{c_{a,r}^{\text{sh}}}}_{c_{a,r}^{\text{sh,sg}}}}$$

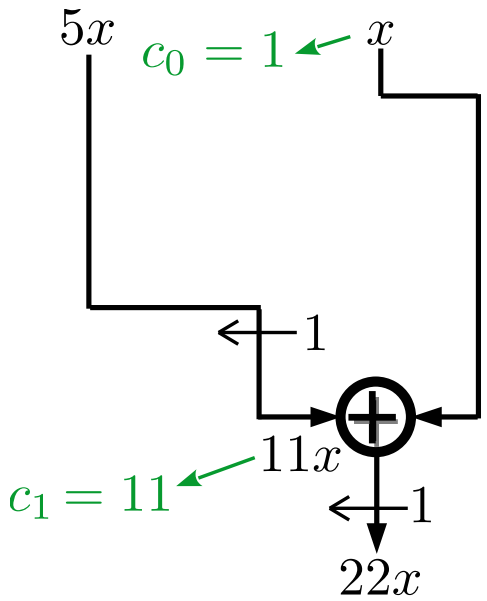




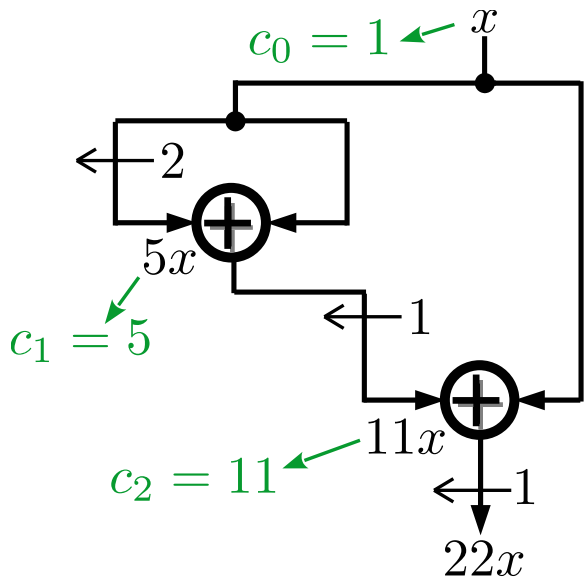
- $c_{1,l,0} = 1$ and $c_{1,r,0} = 1$;
- $c_{2,l,0} = 0$ and $c_{2,r,0} = 1$ and $c_{2,l,1} = 1$ and $c_{2,r,1} = 0$;
- $c_{1,l} = 1$ and $c_{1,r} = 1$ and $c_{2,l} = 5$ and $c_{2,r} = 1$;
- $s_{1,l} = 2$ and $s_1 = 0$ and $s_{2,l} = 1$ and $s_2 = 0$;
- $\sigma_{1,l} = 0$ and $\sigma_{1,r} = 0$ and $\sigma_{2,l} = 0$ and $\sigma_{2,r} = 0$;

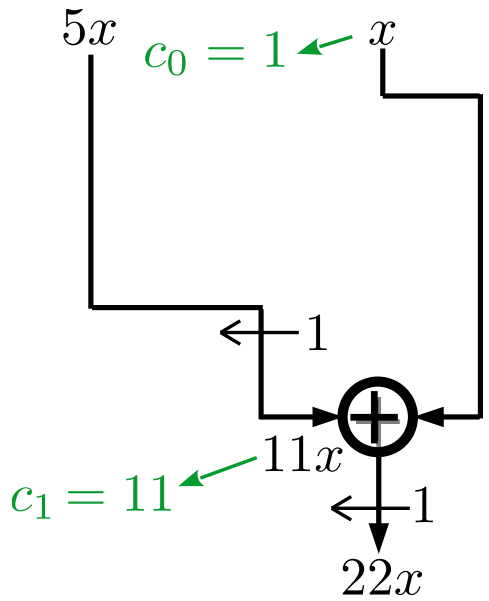
Progressive Adder-Aware Training: SCM-inspired model





Progressive Adder-Aware Training: SCM-inspired model



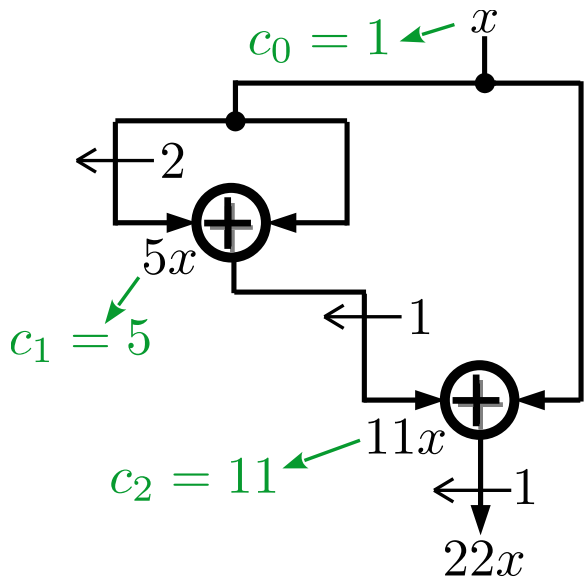


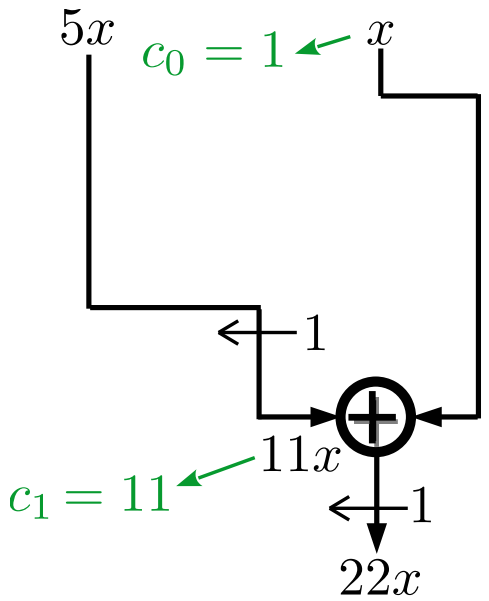
To consider fixed weights $\mathbf{F}$, we define c_a for all $a \in [\![-|\mathbf{F}|, N]\!]$

Then we fix "starting" c_a's:

$$c_a = F_{-a} \qquad \forall a \in [\![-|\mathbf{F}|, -1]\!]$$

Progressive Adder-Aware Training: SCM-inspired model





To consider fixed weights $\mathbf{F}$, we define c_a for all $a \in [\![-|\mathbf{F}|, N]\!]$

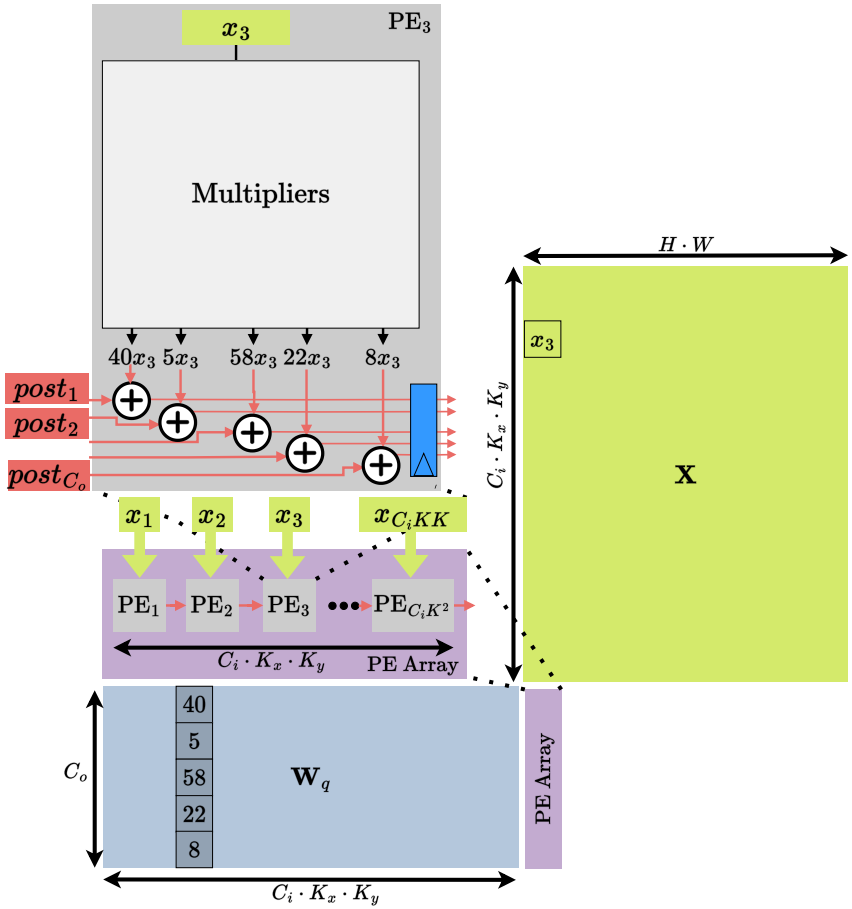
Then we fix "starting" c_a's:

$$c_a = F_{-a} \qquad \forall a \in [\![-|\mathbf{F}|, -1]\!]$$

We can **compute a score taking into account fixed weights**

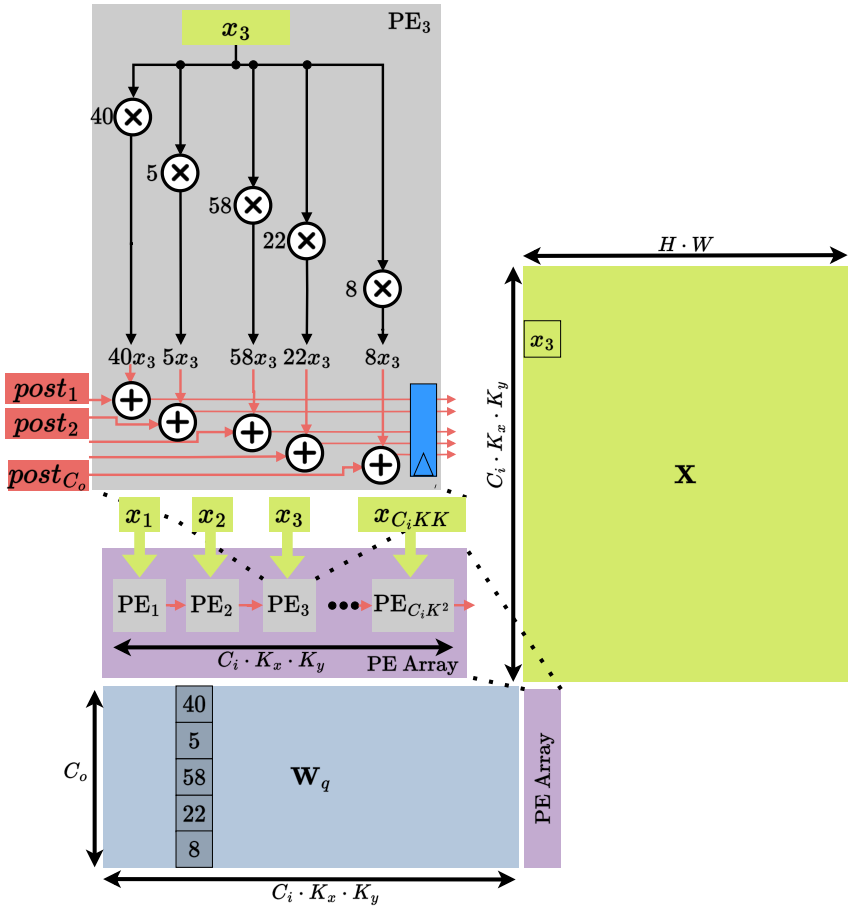
Comparing hardware implementations

- Systolic Array with a **weight stationary** dataflow design: a trade-off between parallelism and hardware cost
- Processing Elements (PE) can be **changed as needed**
- HLS code generation: **easily interchangeable** components



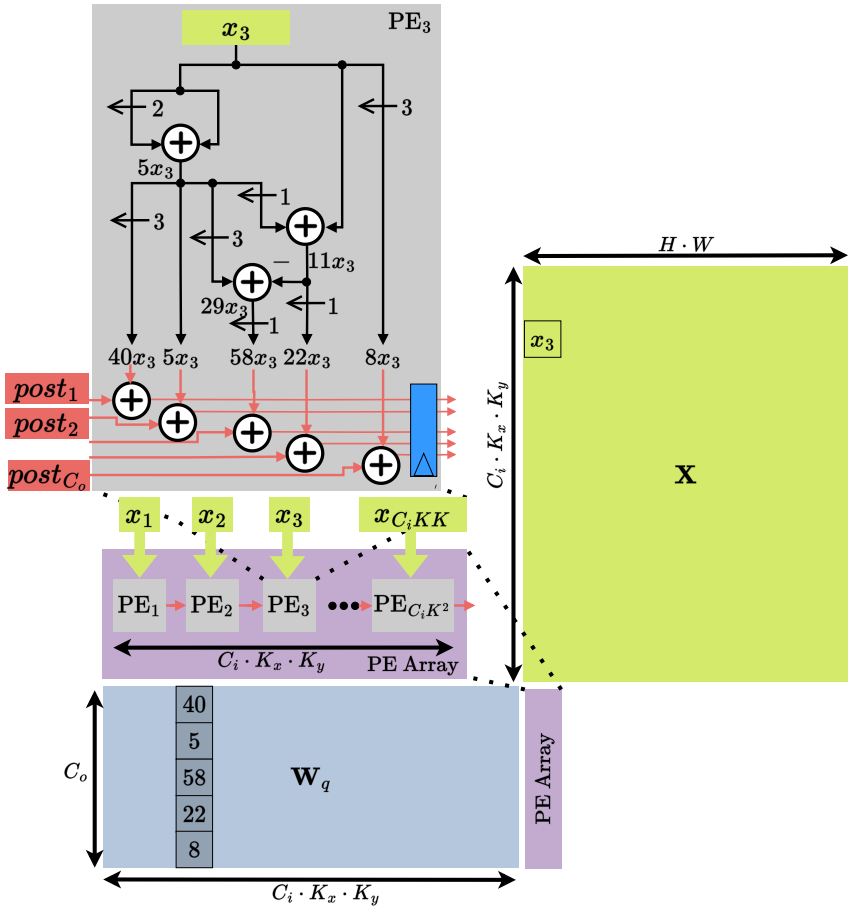
Comparing hardware implementations

- Systolic Array with a **weight stationary** dataflow design: a trade-off between parallelism and hardware cost
- Processing Elements (PE) can be **changed as needed**
- HLS code generation: **easily interchangeable** components



Comparing hardware implementations

- Systolic Array with a **weight stationary** dataflow design: a trade-off between parallelism and hardware cost
- Processing Elements (PE) can be **changed as needed**
- HLS code generation: **easily interchangeable** components



Comparing hardware implementations

CNN	Method	LUT (%)	FF (%)	DSP (%)
ResNet-20	Generic Mult.	15.17	1.90	0
	DSPs	3.91	0.90	68.06
	Adder graph	8.49	1.54	0
ResNet-18	Generic Mult.	64.94	9.44	0
	DSPs	–	–	308†
	Adder graph	41.82	7.05	0
VDSR-10	Generic Mult.	$> 100\%$	–	–
	DSPs	–	–	616†
	Adder graph	97.87	14.59	0

†indicates an estimation as place and route was not possible

Xilinx ZCU104 board

- Using only DSPs is not reasonable
- Using adder graphs leads to lower resource consumption than letting the tool decide

Comparing hardware implementations

CNN	Method	LUT (%)	FF (%)	DSP (%)
ResNet-20	Generic Mult.	15.17	1.90	0
	DSPs	3.91	0.90	68.06
	Adder graph	**8.49**	1.54	0
ResNet-18	Generic Mult.	64.94	9.44	0
	DSPs	–	–	308†
	Adder graph	**41.82**	7.05	0
VDSR-10	Generic Mult.	$> 100\%$	–	–
	DSPs	–	–	616†
	Adder graph	**97.87**	14.59	0

†indicates an estimation as place and route was not possible

Xilinx ZCU104 board

- Using only DSPs is not reasonable
- Using adder graphs leads to lower resource consumption than letting the tool decide

→ **Using adder graphs in CNNs looks interesting**

Adder-Aware Training vs Quantization-Aware Training

CNN	Method	LUT (%)	FF (%)
ResNet-20	Classic QAT	8.49	1.54
	Our AAT	6.19	1.20
ResNet-18	Classic QAT	41.82	7.05
	Our AAT	23.46	4.23
VDSR-10	Classic QAT	97.87	14.59
	Our AAT	81.54	12.15

- HW cost reduced by $\sim$25% for a layer of ResNet-20
- HW cost reduced by $\sim$40% for a layer of ResNet-18
- HW cost reduced by $\sim$16% for a layer of VDSR-10

$\rightarrow$ **Hypothesis:** more **parallelism** to exploit in ResNet-18

Adder-Aware Training vs Quantization-Aware Training

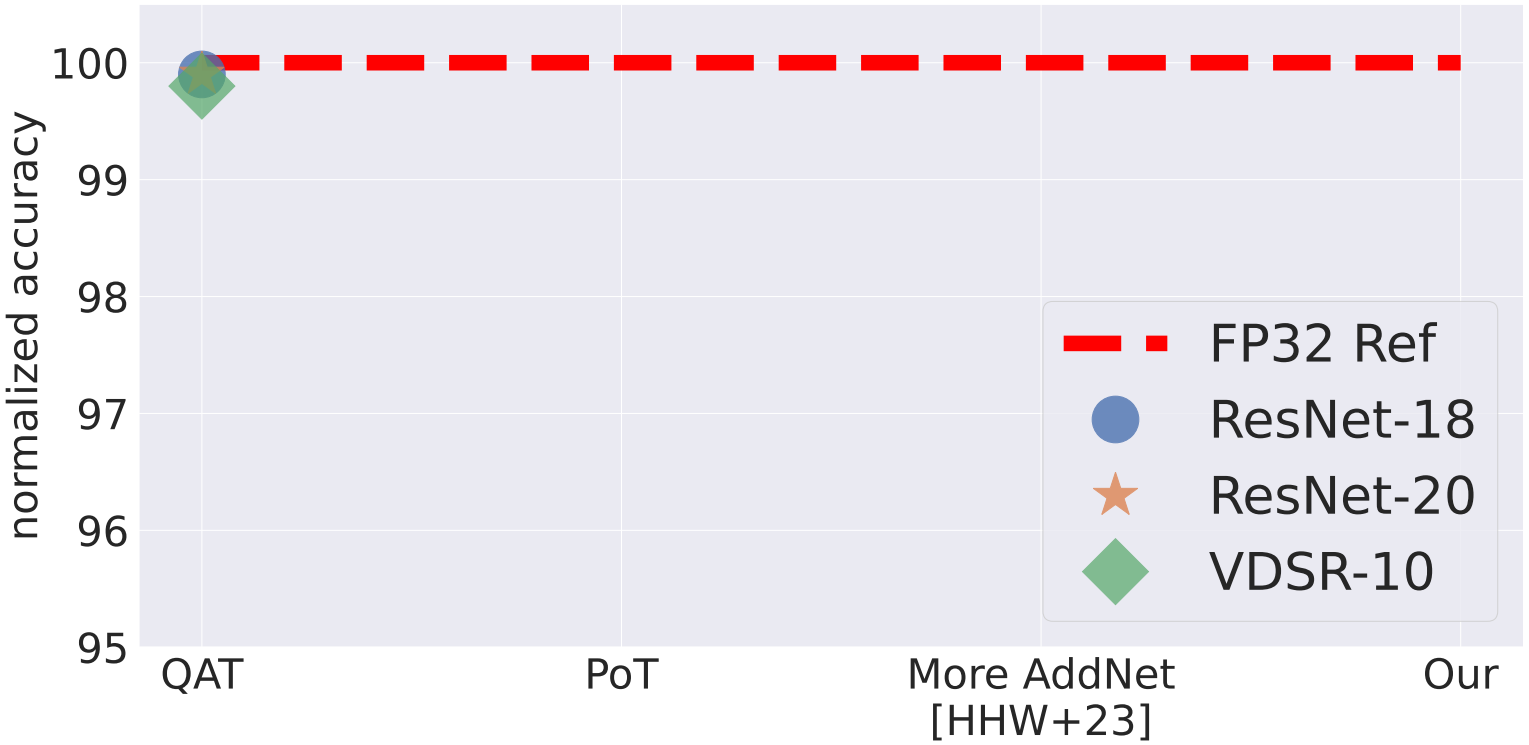
CNN	Method	LUT (%)	FF (%)
ResNet-20	Classic QAT	8.49	1.54
	Our AAT	6.19	1.20
ResNet-18	Classic QAT	41.82	7.05
	Our AAT	23.46	4.23
VDSR-10	Classic QAT	97.87	14.59
	Our AAT	81.54	12.15

- HW cost reduced by $\sim$25% for a layer of ResNet-20
- HW cost reduced by $\sim$40% for a layer of ResNet-18
- HW cost reduced by $\sim$16% for a layer of VDSR-10

$\rightarrow$ **Hypothesis:** more **parallelism** to exploit in ResNet-18

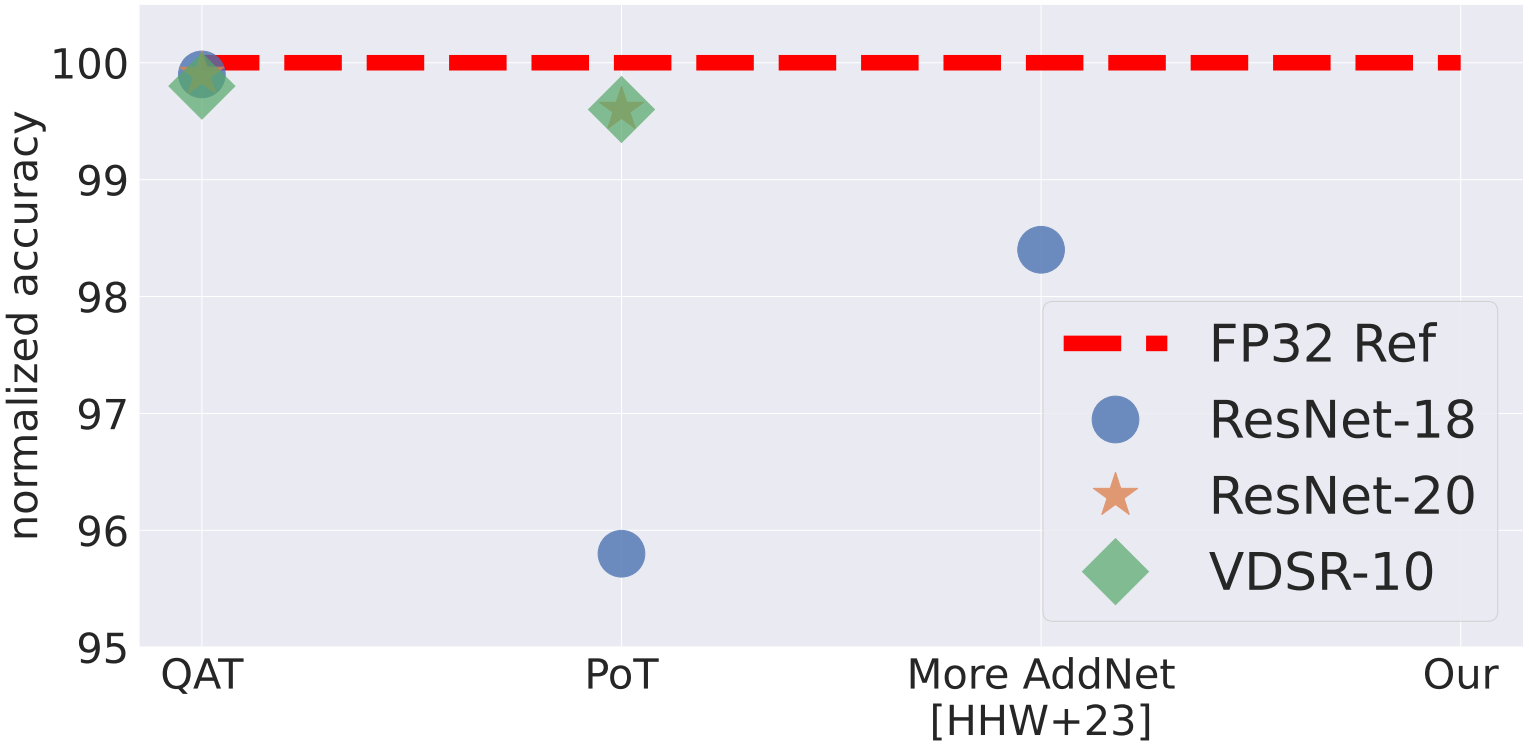
Disclaimer: no HW comparison with SotA AAT approaches

Adder-Aware Training vs Quantization-Aware Training



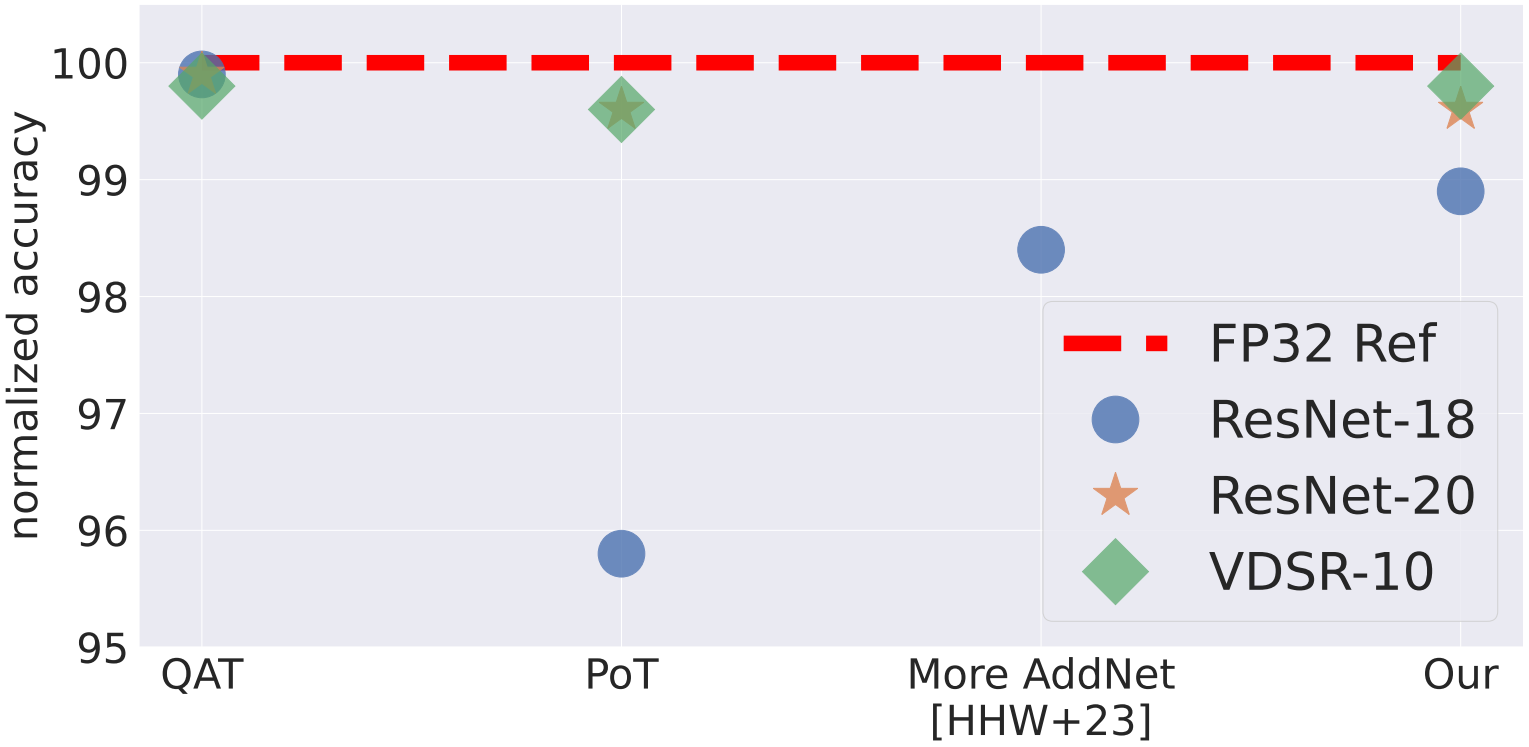
FP32 Ref: ResNet-18: 73.1%, ResNet-20: 92.8%, VDSR: 34.03dB

Adder-Aware Training vs Quantization-Aware Training



FP32 Ref: ResNet-18: 73.1%, ResNet-20: 92.8%, VDSR: 34.03dB

Adder-Aware Training vs Quantization-Aware Training



FP32 Ref: ResNet-18: 73.1%, ResNet-20: 92.8%, VDSR: 34.03dB

- AAT accuracy $<$ QAT accuracy
  $\rightarrow$ the **AAT problem is more constrained**
- Progressive AAT accuracy $>$ Precomputed sets AAT accuracy
  $\rightarrow$ not all weight values are allowed

Conclusion and perspectives

- We propose a new Adder-Aware Training approach
  - → Key idea: **progressively** fix weights
  - → This approach relies on **solving an optimization problem**
- Multiple experiments to have accuracy *and* hardware results
  - → **More constraints** on the problem leads to **accuracy loss**
  - → A **new flexible hardware accelerator**
  - → Using adder graphs for neural networks implementation is promising
  - → Adder-Aware Training leads to **significant hardware cost reduction** w. r. t. a vanilla QAT approach

Conclusion and perspectives

Next steps:

- Accuracy and hardware comparison with existing AAT approaches
- Hardware comparison over a complete CNN
- Speeding up the score computation
- Applying Adder-Aware Training approaches to other models (LLMs?)
- Free weights according to their gradient
- Find a balance between adder graphs and DSPs

[ECS+21] Mostafa Elhoushi, Zihao Chen, Farhan Shafiq, Ye Henry Tian, and Joey Yiwei Li, *DeepShift: Towards Multiplication-Less Neural Networks*, Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2021, pp. 2359–2368.

[GAGN15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, *Deep Learning with Limited Numerical Precision*, Proceedings of the 32nd International Conference on Machine Learning (Lille, France) (Francis Bach and David Blei, eds.), Proceedings of Machine Learning Research, vol. 37, PMLR, July 2015, pp. 1737–1746.

[GKD+21] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer, *A survey of quantization methods for efficient neural network inference. corr abs/2103.13630 (2021)*, arXiv preprint arXiv:2103.13630 (2021).

[GV23] Rémi Garcia and Anastasia Volkova, *Toward the Multiple Constant Multiplication at Minimal Hardware Cost*, IEEE Transactions on Circuits and Systems I: Regular Papers **70** (2023), no. 5, 1976–1988.

[HHW+23] Martin Hardieck, Tobias Habermann, Fabian Wagner, Michael Mecik, Martin Kumm, and Peter Zipf, *More AddNet: A deeper insight into DNNs using FPGA-optimized multipliers*, 2023 IEEE International Symposium on Circuits and Systems (ISCAS), vol. abs/1308.3432, IEEE, May 2023, pp. 1–5.

[HKKV22] Tobias Habermann, Jonas Kühle, Martin Kumm, and Anastasia Volkova, *Hardware-Aware Quantization for Multiplierless Neural Network Controllers*, 2022 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), IEEE, November 2022.

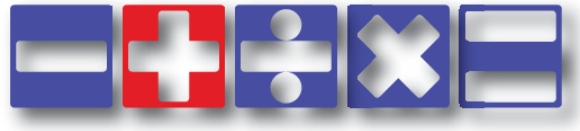
[ZWN+16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou, *DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients*, 2016.

Conclusion and perspectives

- We propose a new Adder-Aware Training approach
  - → Key idea: **progressively** fix weights
  - → This approach relies on **solving an optimization problem**
- Multiple experiments to have accuracy *and* hardware results
  - → **More constraints** on the problem leads to **accuracy loss**
  - → A **new flexible hardware accelerator**
  - → Using adder graphs for neural networks implementation is promising
  - → Adder-Aware Training leads to **significant hardware cost reduction** w. r. t. a vanilla QAT approach

Thank you! Questions?

ARITH 2025



Hardware-Aware Training for Multiplierless Convolutional Neural Networks

Rémi Garcia, **Léo Pradels**, Silviu Filip, Olivier Sentieys

May, 5th 2025

Université de Rennes, IRISA, Inria

Comparing hardware implementations

Comparing different PE implementations:

- Using adder graphs
- Using DSPs
- Letting the synthesis tool decide (using $*$ operator)

Context:

- ResNet-20, ResNet-18 and VDSR-10 networks
- CIFAR-10, ImageNet-1K and Set291/Set5 datasets
- 8-bit Quantization-Aware Training
- Training calls an MILP solver for scores: Gurobi through Pyomo
- Xilinx ZCU104 board
- Results for a single layer, including the systolic array cost

Progressive Adder-Aware Training: Solving the ILP model

- Models are small
  - $\rightarrow \sim 30 - 200$ variables
  - $\rightarrow \sim 50 - 250$ constraints
- Solving times are negligible
  - $\rightarrow < 1$s

- ResNet-18 on ImageNet-1K: Training time $\sim$24h ($\sim$21h + $\sim$3h)

$\rightarrow$ **The ILP model is not a bottleneck**

Training on abacus 17 Grid5K (RTX 2080TI)