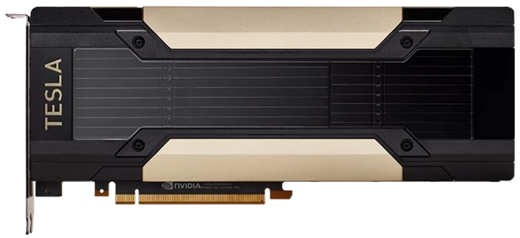
On-device Deep Learning

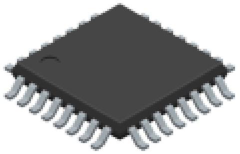
Zhaojun Ni
School of Information Science and Technology

Key Reference

- Han Song’s Group @ MIT
- Yoo Hoi-Jun’s Group @ KAIST
- Shi Yuanming’s Group @ ShanghaiTech

Deep Learning Going “Tiny”





	Cloud AI Data centers	**Mobile AI** Smartphones	**Tiny AI** IoT Devices/Microcontrollers
Memory (Activation)	32GB	4GB	320kB
Storage (Weights)	~TB/PB	512GB	1MB

On-device Deep Learning is Essential

- AI systems need to adapt to new sensor data for **customization** and **continual learning**
- To protect user's **private** data or achieve higher performance in **real-world** applications
- However, tiny devices contain only **limited** computation capability

Outline: From Design to Inference

□(Design) Efficient Neural Architecture

- Neural Architecture Search(NAS)
- Resource-constrained model specialization

□(Training) Efficient Training Hardware

- Dataflow optimizations
- External memory access reduction
- Computation optimizations

□(Inference) Efficient Inference System

- Memory-efficient inference engine
- Edge training and inference system

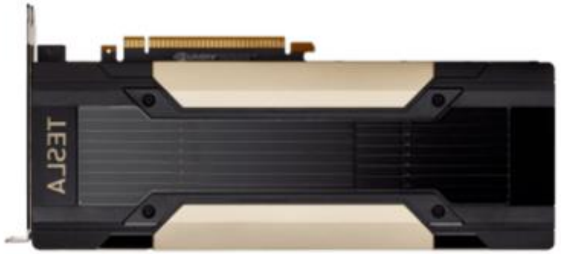
#1. Neural Architecture Search(NAS)

- not only reduce FLOPs or latency
- extend the **search space** to fit the tiny **resource constraints**

  $S' = \textit{kernel size} \times \textit{expansion ratio} \times \textit{depth} \times \textit{input resolution } \underline{R} \times \textit{width multiplier } \underline{W}$

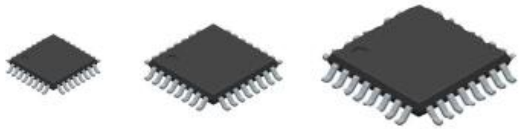
- Different R and W for different hardware capacity

R=260, W=1.4



R=224, W=1.0

R=?, W=?

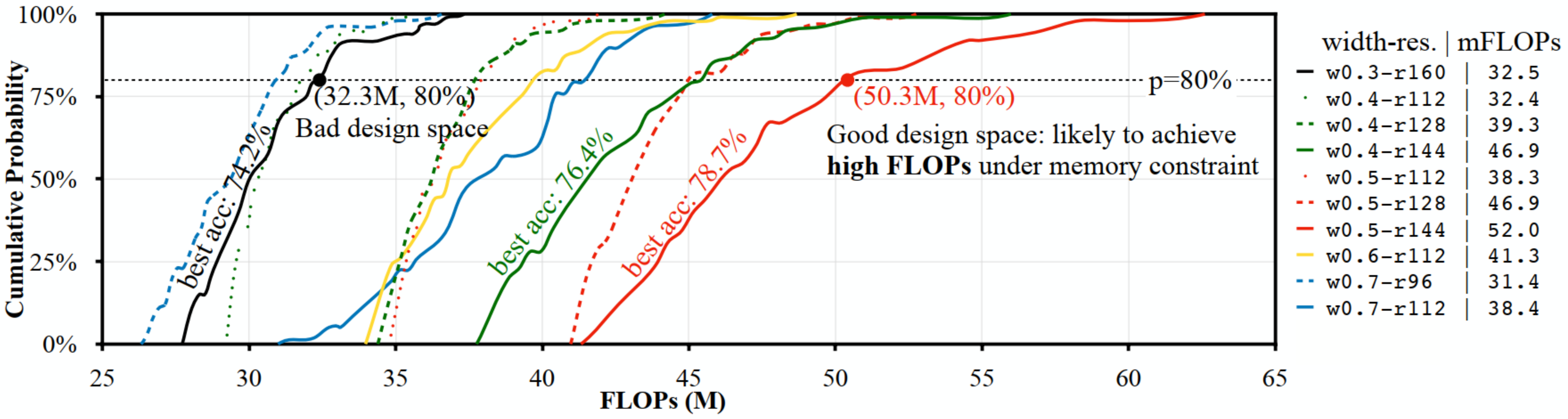


F412/F743/H746/…
256kB/320kB/512kB/…

#1. Neural Architecture Search(NAS)

- Analyzing FLOPs distribution of satisfying models in each search space:

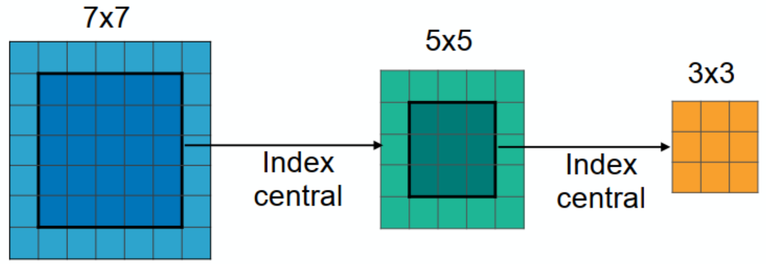
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



#2. Resource-constrained model specialization

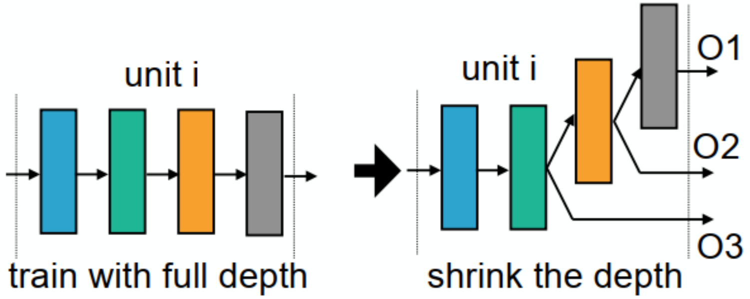
Kernel Size

Smaller kernel takes centered weights



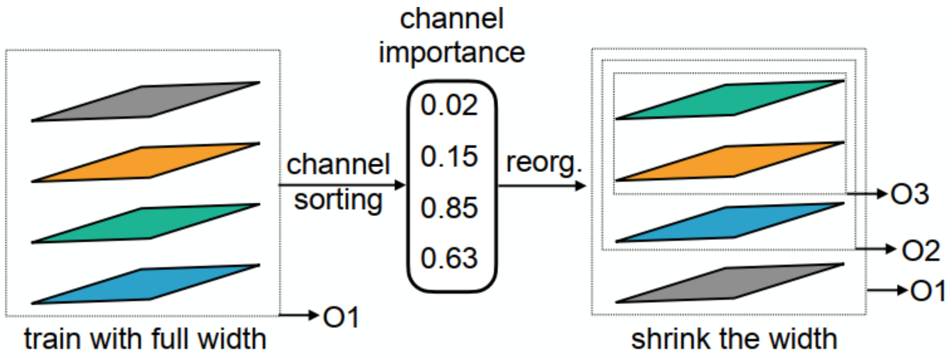
Depth

Allow later layers in each unit to be skipped to reduce the depth



Width

Keep the most important channels when shrinking via channel sorting

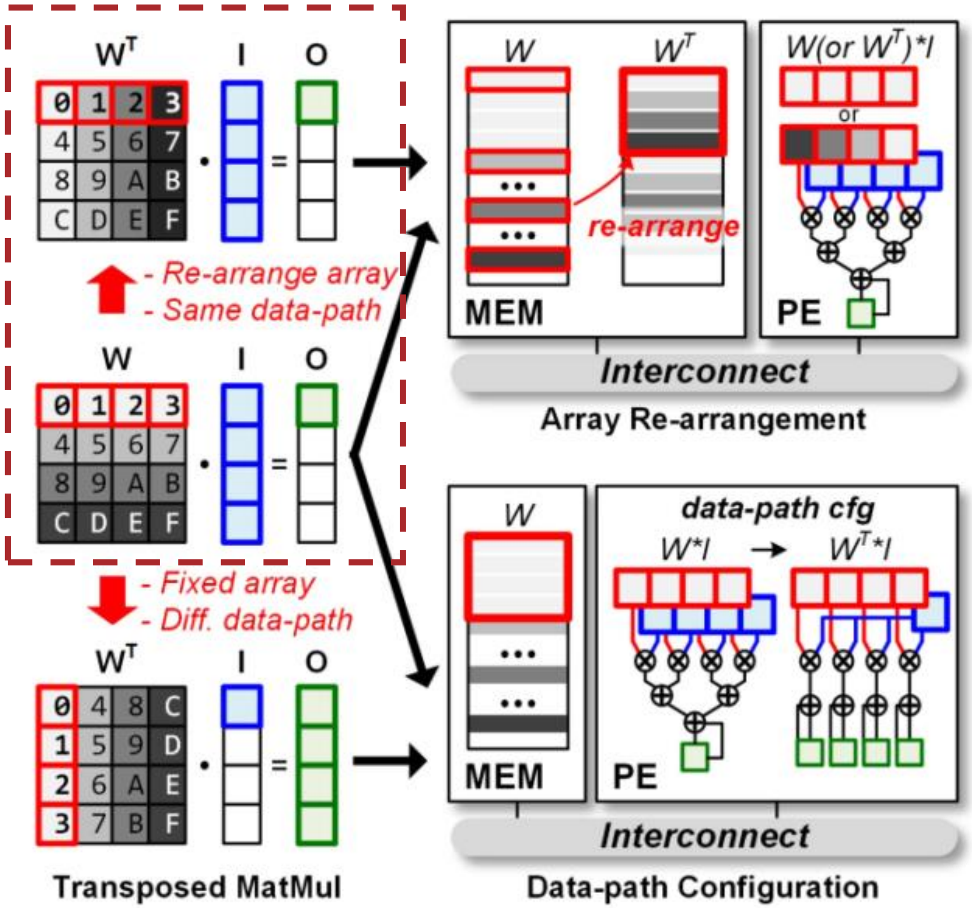


#3. Dataflow optimizations

Data array re-arrangement

Modify the **memory** layout

- loads tiled weights from external memory and stores to on-chip SRAM or regfiles in transposed order
- read array for both transposed and non-transposed manner with a bit-rotator
- custom on-chip SRAM which can read data as transposed

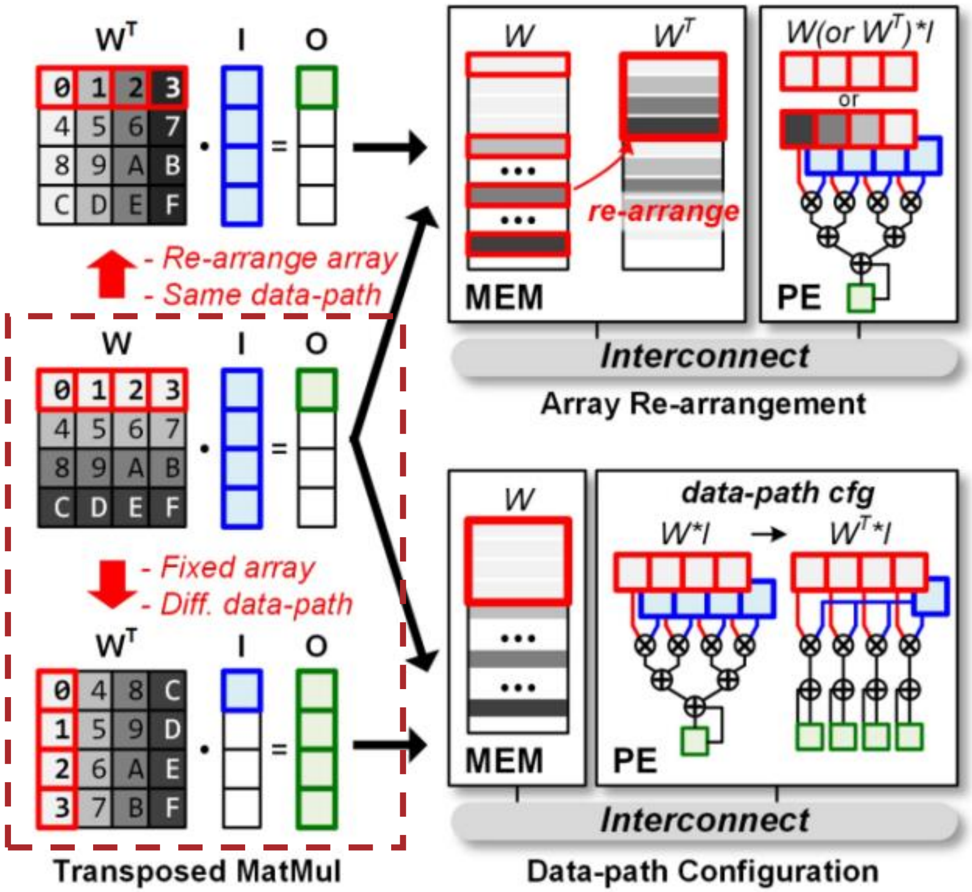


#3. Dataflow optimizations

Data-path configuration

Modify data-path of the **PE array**

- configurable and multiple data-paths in a PE array
- transposed and non-transposed MatMul without redundant memory access by exchangeable feeding paths of input and weight
- different dataflow of DNN training steps with heterogeneous architectures



#4. External memory access reduction

Sparse compression

Represent the tensors into a non-zero vector

- Zero-Value Compression (ZVC)
- Compressed Sparse Row (CSR)
- Run Length Encoding (RLE)

Probability of occurrence based compression

Encode more likely to occur values into low bit-width code

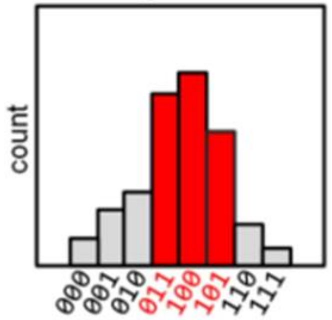
- Huffman coding
- Replaced most-frequent exponent values with low-bit code

Sparse compression

$$\begin{bmatrix} 0 & 5 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 4 & 0 \end{bmatrix}$$

	compressed vector
data	[5 1 2 4]
cidx	[1 0 0 2]
ridx	[0 1 3 3]

Occur. based compression

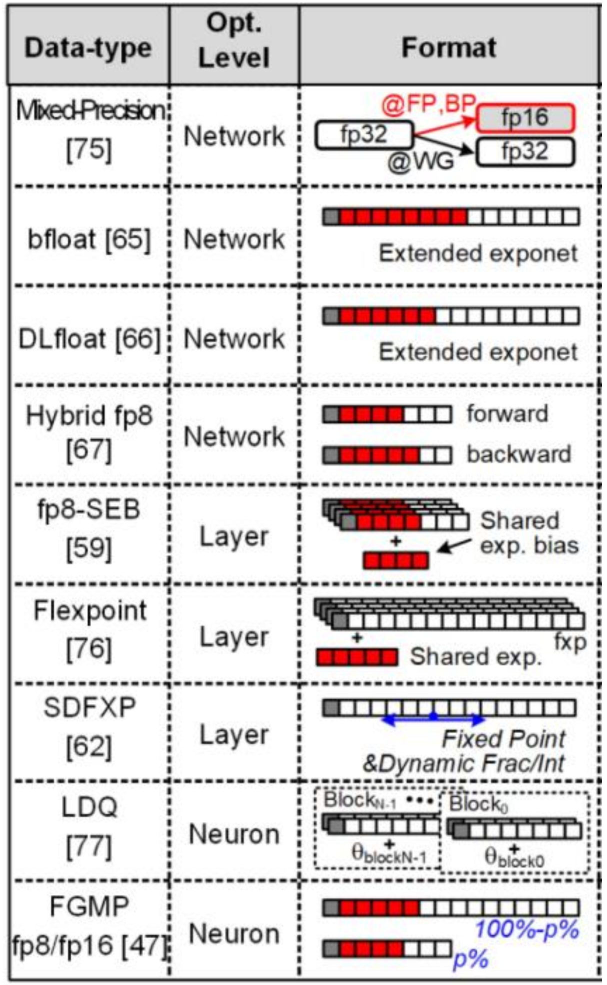


data	*p(%)*	code
011	35	01
100	32	10
101	31	11
000	0.5	00000
001	1.1	00001
...	...	...

#4. External memory access reduction

Reduced Bit-width

- custom reduced width data-type
- some hardware uses fp16 or fp8
- use low bit-width for representing the stored data
- 1 bit-width for ReLU-Pool layer pairs

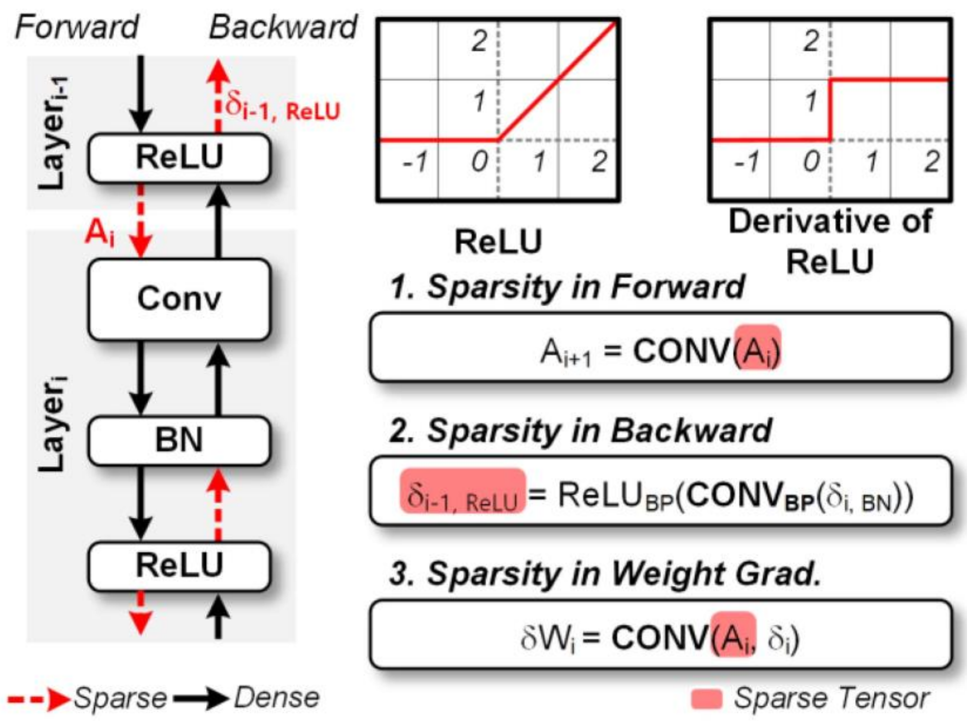


Data-type	Opt. Level	Format
Mixed-Precision [75]	Network	fp32 @FP,BP fp16 @WG fp32
bfloat [65]	Network	Extended exponet
DLfloat [66]	Network	Extended exponet
Hybrid fp8 [67]	Network	forward backward
fp8-SEB [59]	Layer	Shared exp. bias
Flexpoint [76]	Layer	Shared exp. fxp
SDFXP [62]	Layer	Fixed Point &Dynamic Frac/Int
LDQ [77]	Neuron	Block_{N-1} ••• Block_0 $\theta_{\mathrm{blockN-1}}$ θ_{block0}
FGMP fp8/fp16 [47]	Neuron	100%-p% p%

#5. Computation optimization method

Sparsity

- replaces fp16 operations with zero input MAC operations and compensates with fp8 operations
- zero output prediction and speculative skipping
- determine the weight to be pruned in the training phase and skip all related operations
- neglect the partial sum of small values

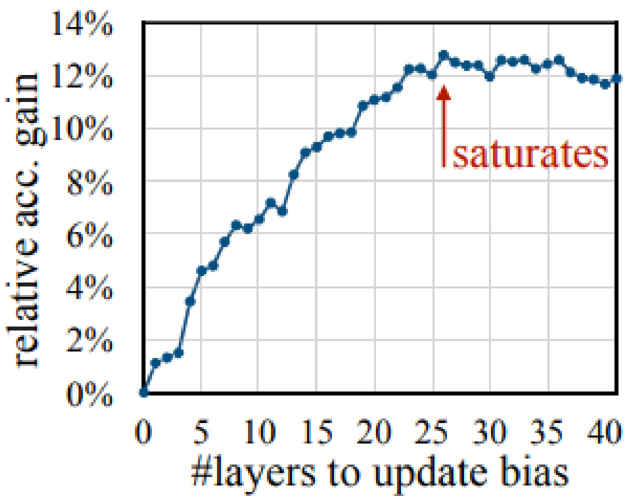


#5. Computation optimizations

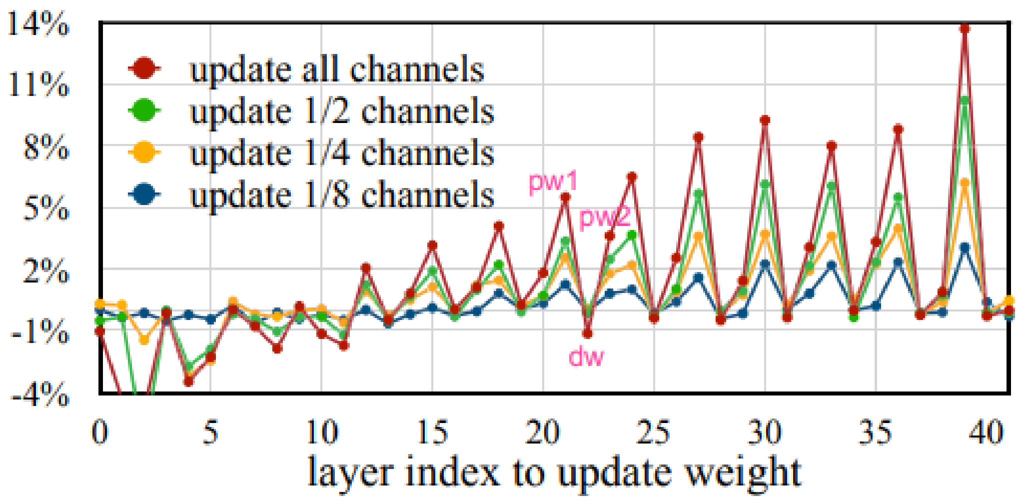
Sparse layer/tensor update

Prune the gradient during backpropagation and update the model sparsely

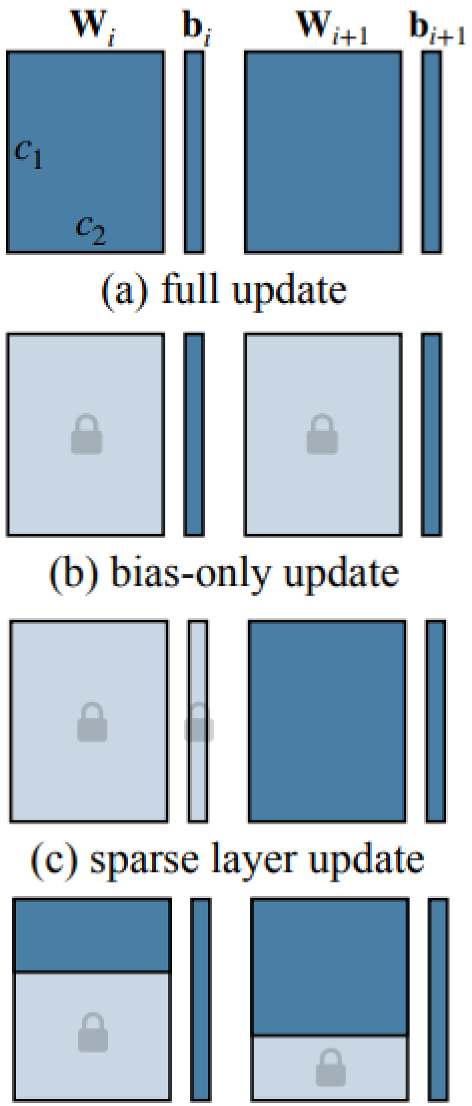
Find the right sparse update scheme



(a) Contribution of last k biases $\Delta\text{acc}_{b[:k]}$



(b) Contribution of a certain weight $\Delta\text{acc}_{W_{i,r}}$



(a) full update

(b) bias-only update

(c) sparse layer update

(d) sparse tensor update

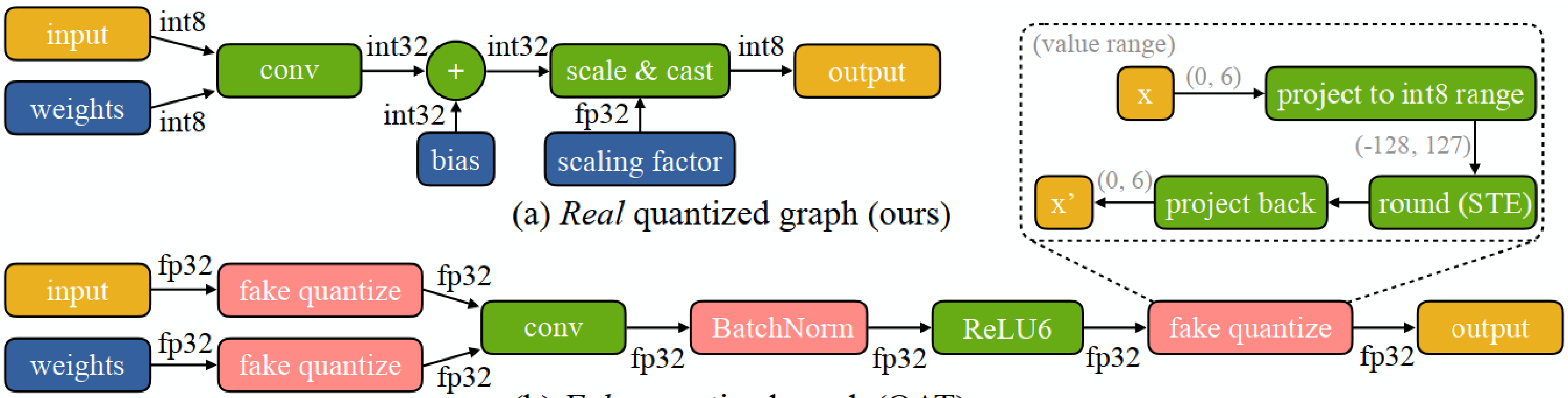
#5. Computation optimizatios

Quantization

- custom bit-precisions
- configurable PE design for various bit-width

- Quantization-Aware Training(QAT)
- Quantization-Aware Scaling(QAS)

$$\tilde{\mathbf{G}}_{\bar{\mathbf{W}}} = \mathbf{G}_{\bar{\mathbf{W}}} \cdot s_{\mathbf{W}}^{-2}, \quad \tilde{\mathbf{G}}_{\bar{\mathbf{b}}} = \mathbf{G}_{\bar{\mathbf{b}}} \cdot s_{\mathbf{W}}^{-2} \cdot s_{\mathbf{x}}^{-2} = \mathbf{G}_{\bar{\mathbf{b}}} \cdot s^{-2}$$



(a) *Real* quantized graph (ours)

(b) *Fake* quantized graph (QAT)

#6. Memory-efficient inference engine

Separated compilation & runtime

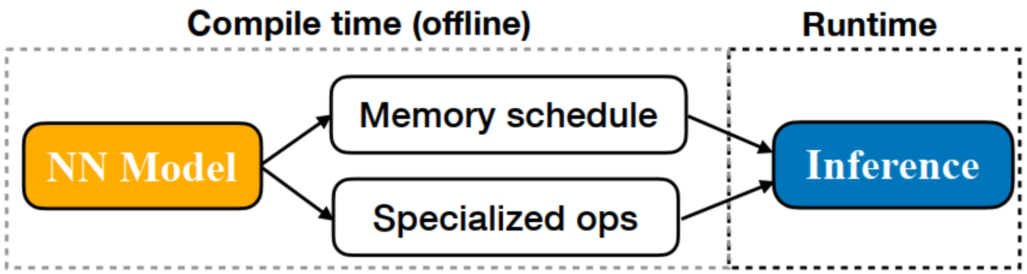
- Reduce overhead

Code generator-based compilation

- Eliminate overheads of runtime interpretation

Model-adaptive memory scheduling

- Increase data reuse for each layer



(a) Model-level memory scheduling

$$M = \max\left(\text{kernel size}_{L_i}^2 \cdot \text{in channels}_{L_i}; \forall L_i \in \boldsymbol{L}\right)$$

(b) Tile size configuration for Im2col

$$\text{tiling size of feature map width}_{L_j} = \lfloor M / \left(\text{kernel size}_{L_j}^2 \cdot \text{in channels}_{L_j}\right) \rfloor$$

#6. Memory-Efficient inference engine

Patch-based inference

- Reduce peak memory
- Reduce the receptive field of the patch-based initial stage
- Increase the receptive field of the later stage

Graph-level optimization

- Minimize memory footprint
- Optimize the overall computation

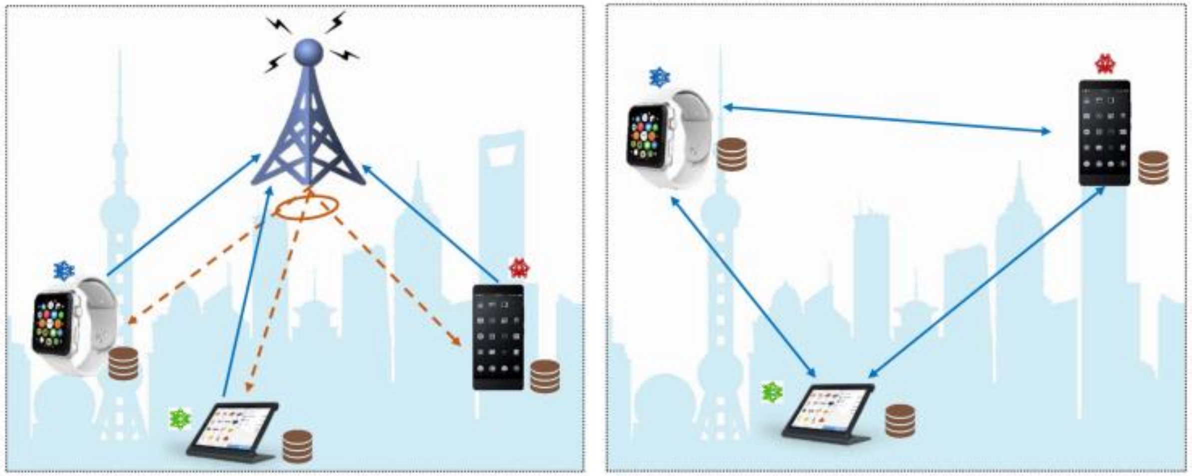
Re-order and in-place update

- Gradient updates are immediately applied once calculated
- Intermediate buffers can be released

#7. Edge training and inference system

Data partition based edge training systems

- Data is massively distributed over a number of edge devices, and each edge device has only a subset of the whole dataset
- During training, each edge device holds a replica of the complete AI model to compute a local update



(a) Distributed mode (b) Decentralized mode

#7. Edge training and inference system

Model partition based edge training systems

- Each node holds part of the model parameters with small storage size
- Accomplish the model training task or the inference task collaboratively
- Data privacy at each node belongs to different parties
- Heavy communication overhead between edge devices

#7. Edge training and inference system

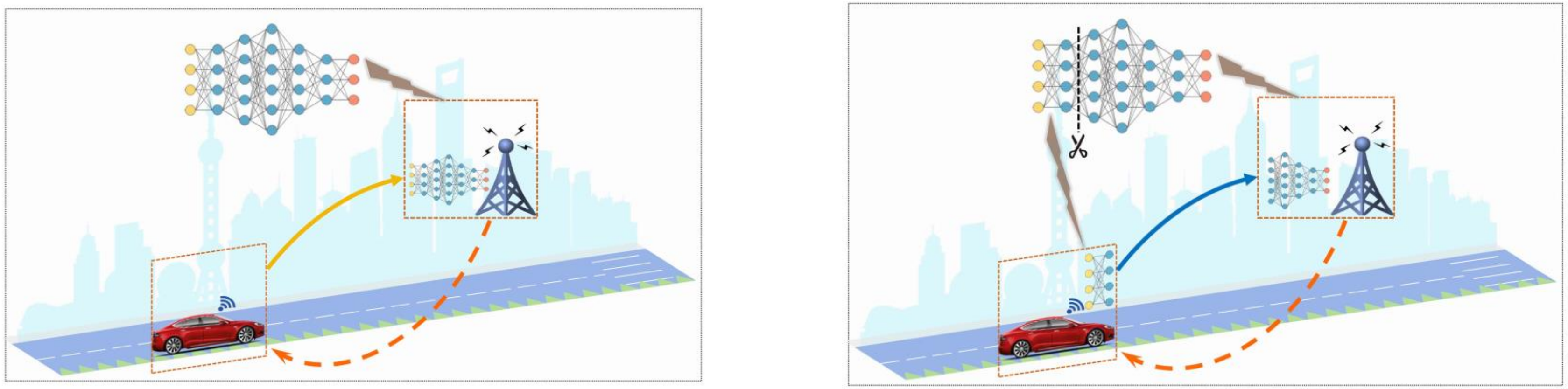
Computation offloading based edge inference systems

- Offload the entire inference task to an edge server

Edge devices should upload their input data to edge servers for inference

- Offload only a part of the task to the edge server

Edge server computes the inference result based on the intermediate value computed by the edge device



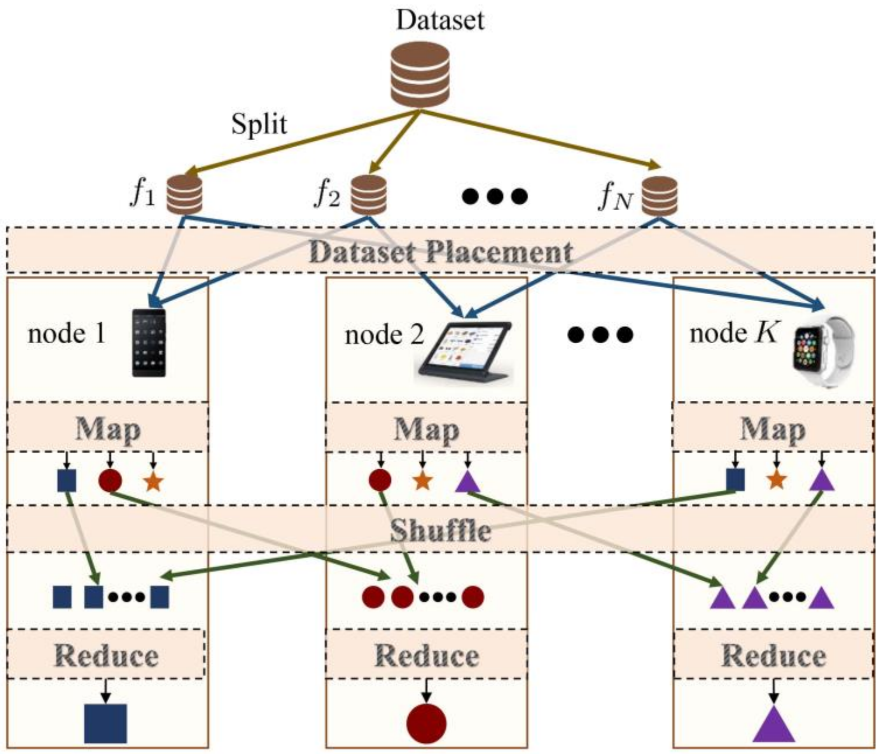
(a) Server-based edge inference

(b) Device-edge joint inference

#7. Edge training and inference system

General edge computing systems

- MapReduce
- In the map phase, every computing node computes a map function of the assigned data
- In the shuffle phase, nodes communicate with each other to obtain some intermediate values
- In the reduce phase, each node computes the assigned output function



THANKS