MIT HAN LAB

Lecture 08
Neural Architecture Search

Part II

Song Han
songhan@mit.edu

Explore websites, people, and locations

amy finkelstein

Top resources for

_ prospective students
_ current students
_ faculty & staff
_ alumni
_ parents
_ Covid-19 and MIT
_ all resources

Join us in building a better world.

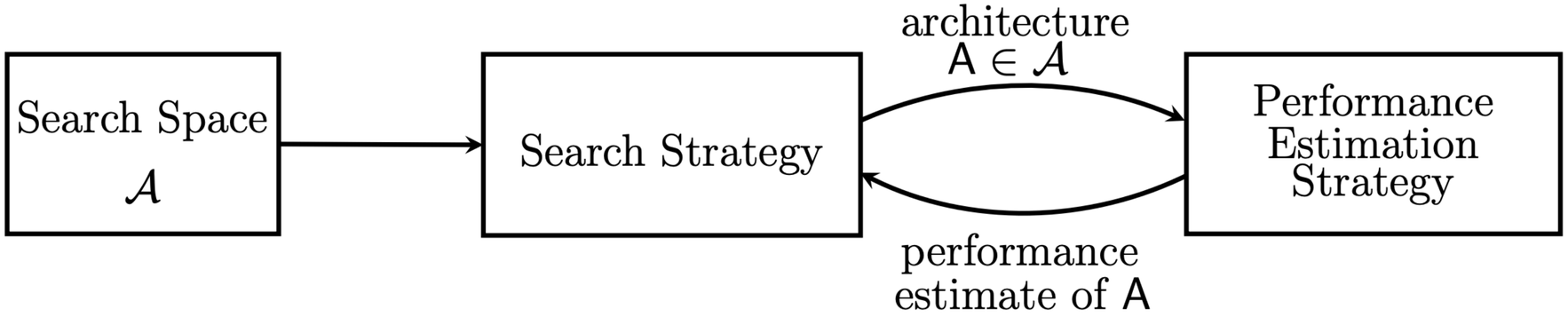
A new technique enables AI models to continually learn from new data on intelligent “edge devices” like smartphones and sensors, reducing energy costs and privacy risks. The advance “makes deep learning more accessible,” says Song Han.

Oct 4, 2022 Full story Share: Explore more spotlights

Recap

Framework of NAS

- Search space is a set of candidate neural network architectures.
- Search strategy defines how to explore the search space.
- Performance estimation strategy defines how to estimate/predict the performance of a given neural network architecture in the design space.

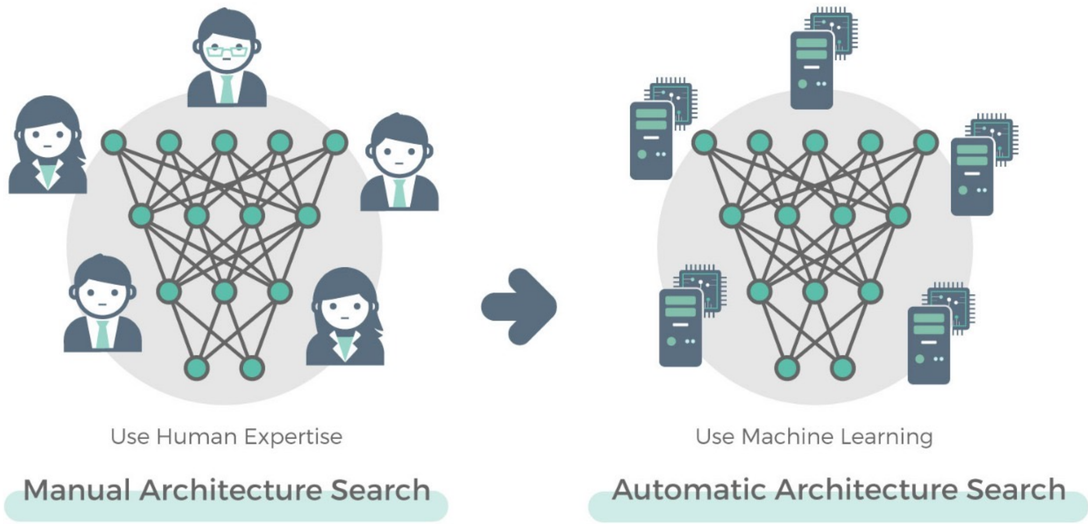


Neural Architecture Search: A Survey [Elskan *et al.*, JMLR 2019]

Lecture Plan

Today we will:

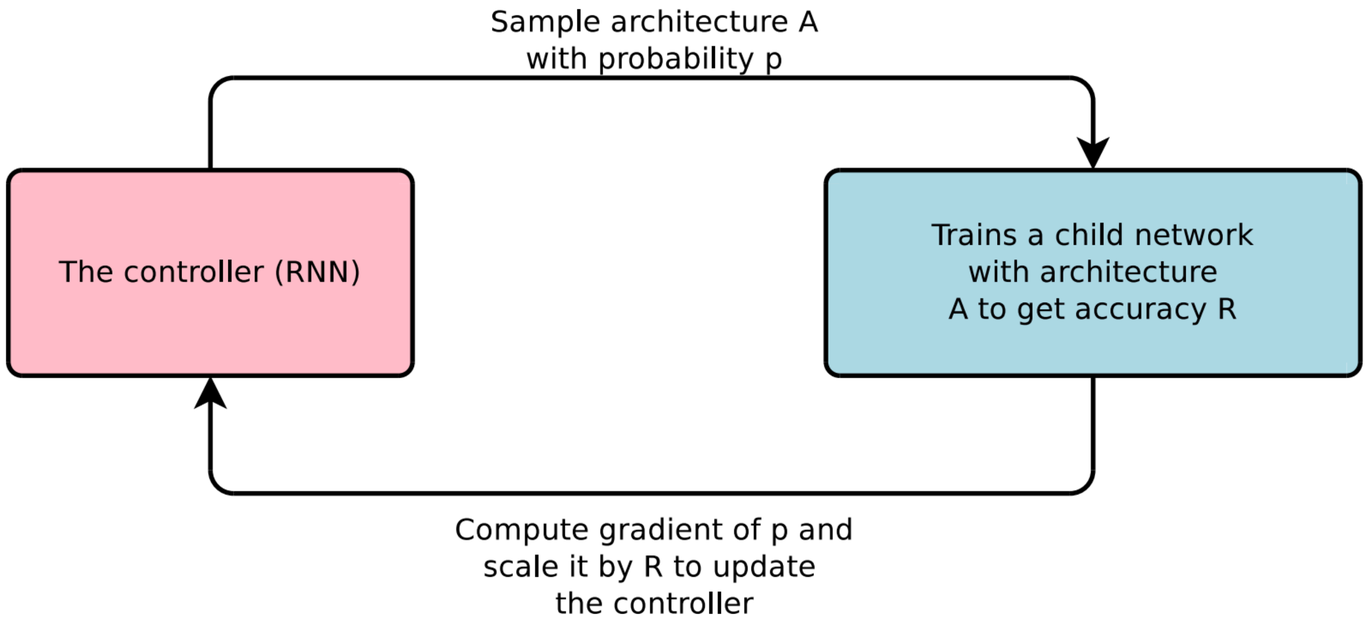
1. Discuss the performance estimation strategies of NAS;
2. Introduce hardware-aware neural architecture search to design efficient neural networks for target hardware;
3. Introduce the application of NAS in downstream tasks.



Performance Estimation in NAS

Train from Scratch

Train sampled model architectures from scratch to collect accuracy feedback



CIFAR10

Train 12,800 model architectures

22,400 GPU-hours

Large-Scale Datasets

ImageNet
COCO
...

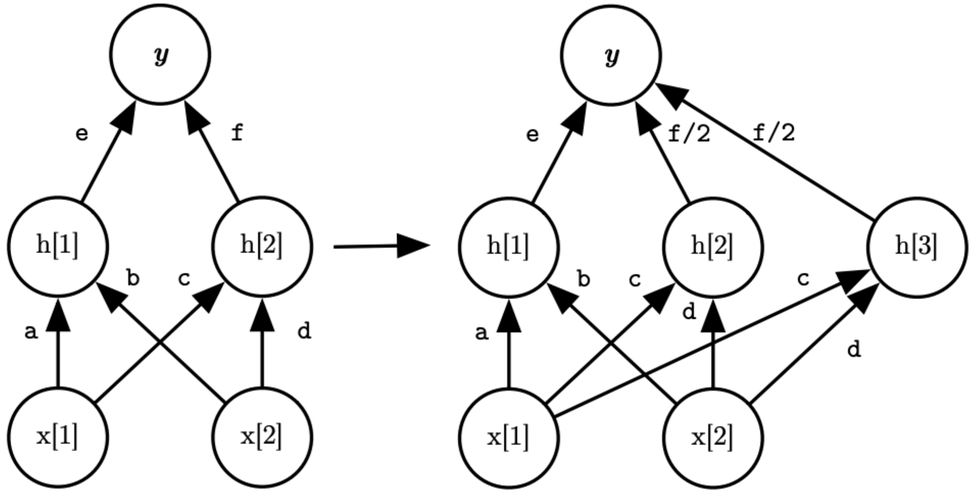
- Train the given model from scratch on the training set
- Evaluate the trained model on the validation set to get accuracy R
- Drawback: prohibitive training cost

Neural Architecture Search with Reinforcement Learning [Zoph *et al.*, ICLR 2017]

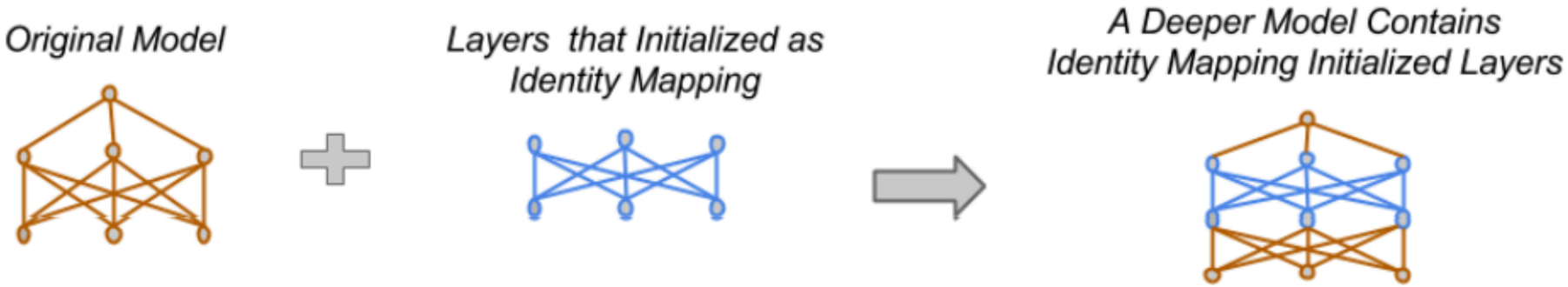
Weight Inheritance

Inherit weights from pre-trained models to reduce the training cost

Net2Wider



Net2Deeper

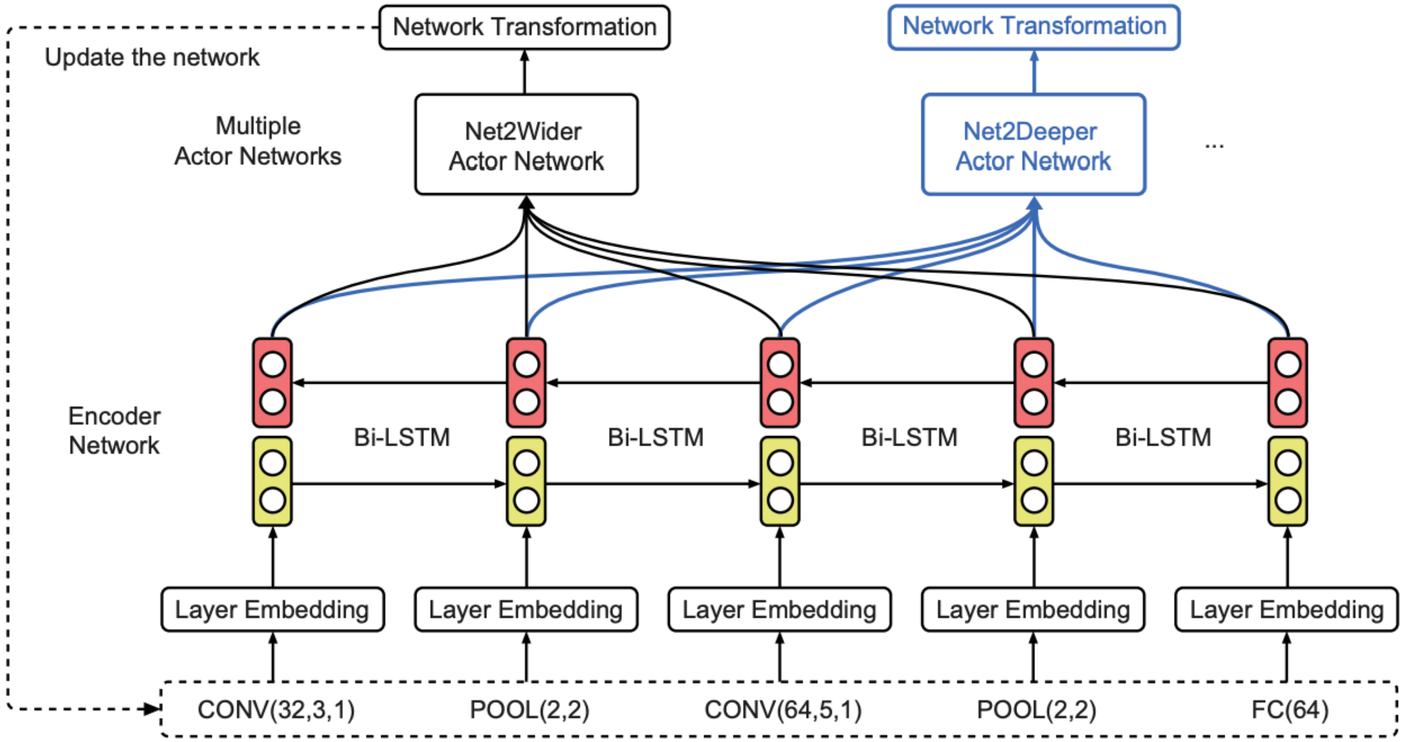


- Inherit weights from a parent model to reduce the training cost instead of training from scratch.

Net2Net: Accelerating Learning via Knowledge Transfer [Chen *et al.*, ICLR 2016]

Weight Inheritance

Take network transformation actions to modify the model architecture



- Inherit weights from a parent model to reduce the training cost instead of training from scratch.
- Generate network transformation actions to update the model architecture instead of directly generating the model architecture.

Efficient Architecture Search by Network Transformation [Cai *et al.*, AAAI 2018]

HyperNetwork: Weight Generator

Train a hypernetwork to generate weights for sampled model architectures

Algorithm 1 SMASH

input Space of all candidate architectures, $\mathbb{R}_c$
  Initialize HyperNet weights H
  loop
    Sample input minibatch x_i, random architecture c and architecture weights $W = H(c)$
    Get training error $E_t = f_c(W, x_i) = f_c(H(c), x_i)$, backprop and update H
  end loop
  loop
    Sample random c and evaluate error on validation set $E_v = f_c(H(c), x_v)$
  end loop
  Fix architecture and train normally with freely-varying weights W

- At each training step, a random model architecture is sampled from the search space.
- The hypernetwork generates weights based on the model architecture.
- Using gradient descent to update the weights of the hypernetwork.

SMASH: One-Shot Model Architecture Search through HyperNetworks [Brock *et al.*, ICLR 2018]

HyperNetwork: Weight Generator

Searched based on the weights generated by the hypernetwork

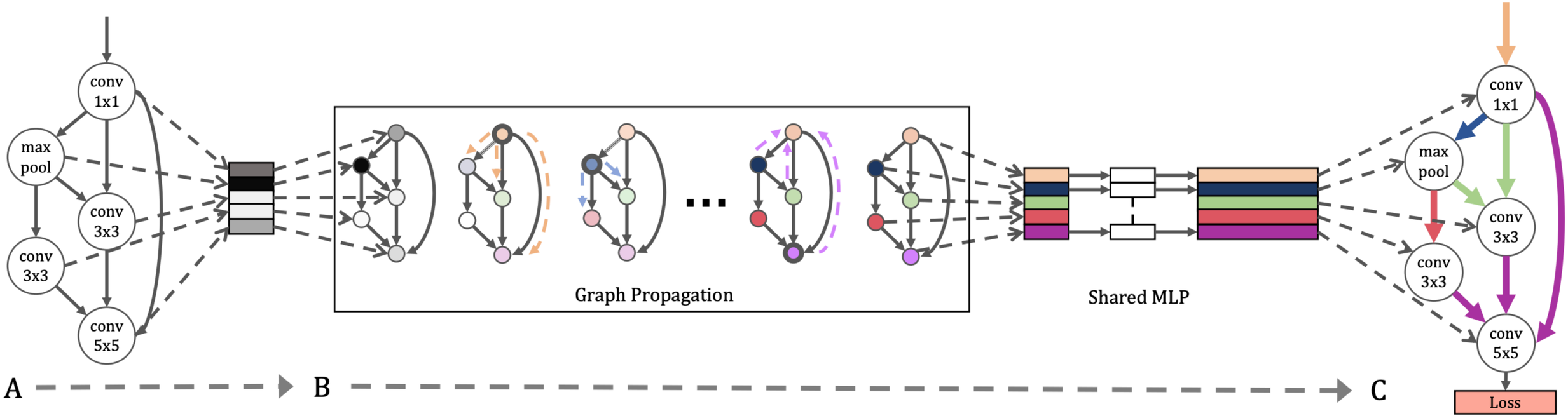
Algorithm 1 SMASH

input Space of all candidate architectures, $\mathbb{R}_c$
  Initialize HyperNet weights H
  loop
    Sample input minibatch x_i, random architecture c and architecture weights $W = H(c)$
    Get training error $E_t = f_c(W, x_i) = f_c(H(c), x_i)$, backprop and update H
  end loop
  loop
    Sample random c and evaluate error on validation set $E_v = f_c(H(c), x_v)$
  end loop
  Fix architecture and train normally with freely-varying weights W

SMASH: One-Shot Model Architecture Search through HyperNetworks [Brock *et al.*, ICLR 2018]

HyperNetwork: Weight Generator

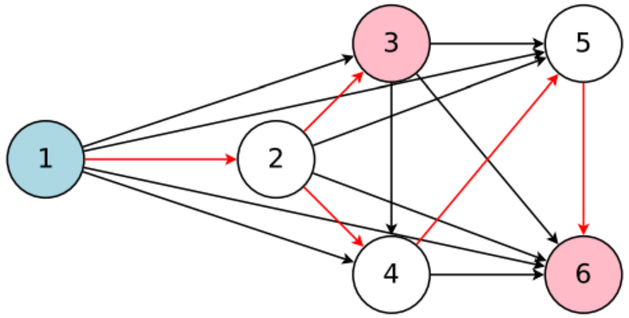
Weight generation



Graph HyperNetwork for Neural Architecture Search [Zhang *et al.*, ICLR 2019]

Weight Sharing

Share weights between different subnetworks



1. Train super-network

For training iterations:
    sample-subgraph();
    forward-backward();

2. Search based on super-network

For search iterations:
    sample-subgraph();
    evaluate();

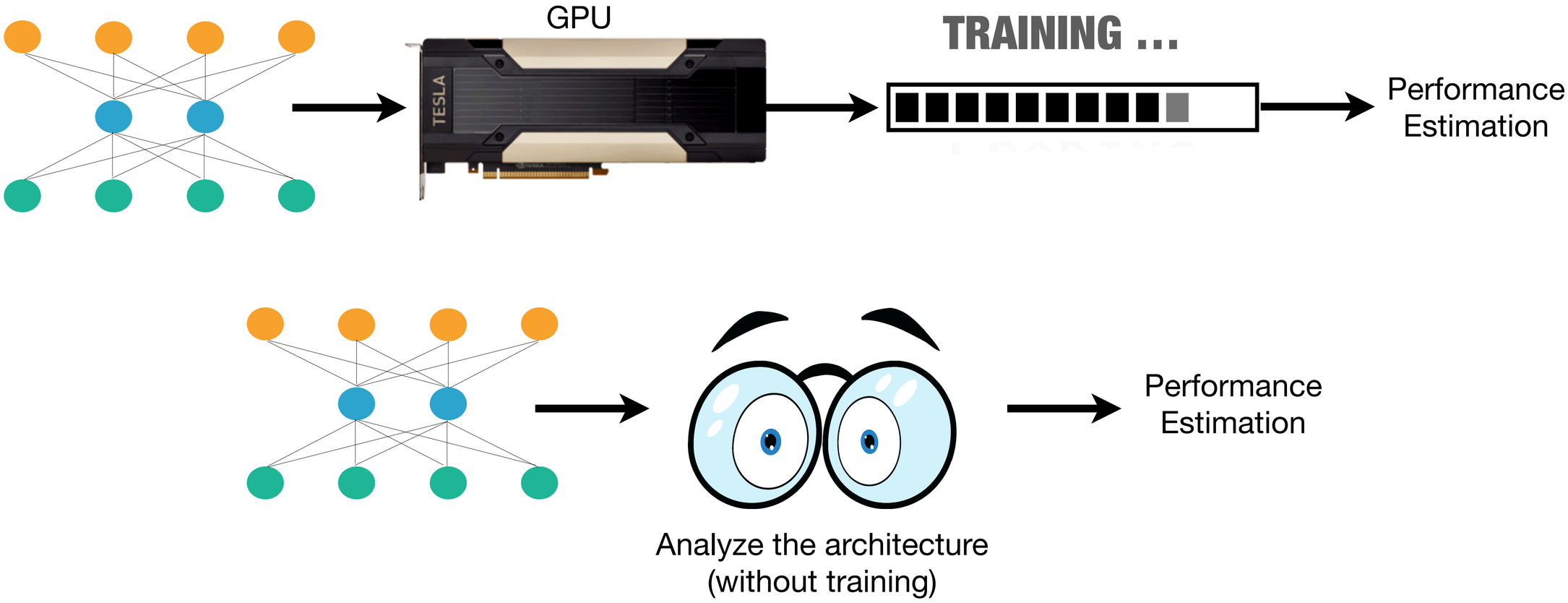
3. Train the select sub-network from scratch

For training iterations:
    forward-backward();

- Train a super-network that contains the entire graph.
- Generate architectures by extracting sub-graphs from the super-network.

Efficient Neural Architecture Search via Parameter Sharing [Pham *et al.*, ICML 2018]

Performance Estimation Heuristics



Performance Estimation Heuristics

ZenNAS

Implementation:

1. Initialize the random input $x \sim \mathcal{N}(0,1)$,

2. Add a small perturbation to x and obtain $x' = x + \epsilon$,

3. Initialize all the weights of the neural network f to a normal distribution $\mathcal{N}(0,1)$,

4. Calculate the log difference: $z_1 = \log(f(x') - f(x))$: **a good model should be sensitive to input perturbations**,

5. Add a compensation term for batch normalization layers:

Consider BN for layer i: $x_i = (x_i - \mu_i)/\sigma_i$. We calculate the square mean $\bar{\sigma}_i$ as $\sqrt{\frac{\sum_j \sigma_{i,j}^2}{c_{\text{out}}}}$.

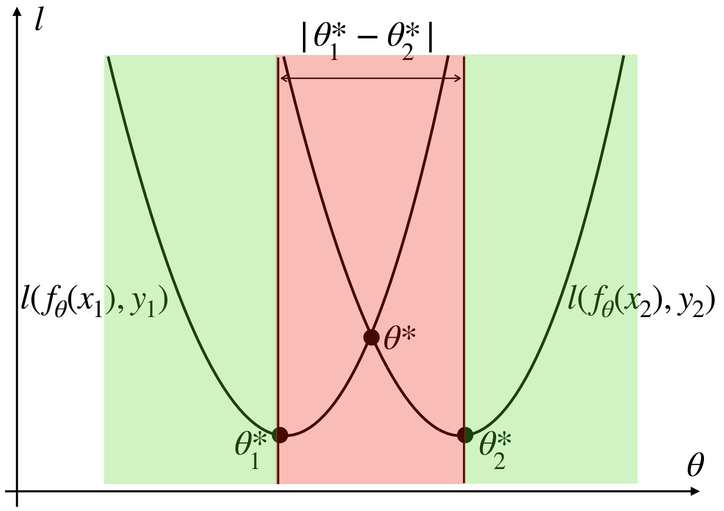
Finally, we have $z_2 = \sum_i \log \bar{\sigma}_i$ and zen score $z = z_1 + z_2$.

Zen-NAS: A Zero-Shot NAS for High-Performance Image Recognition [Lin *et al.*, ICCV 2021]

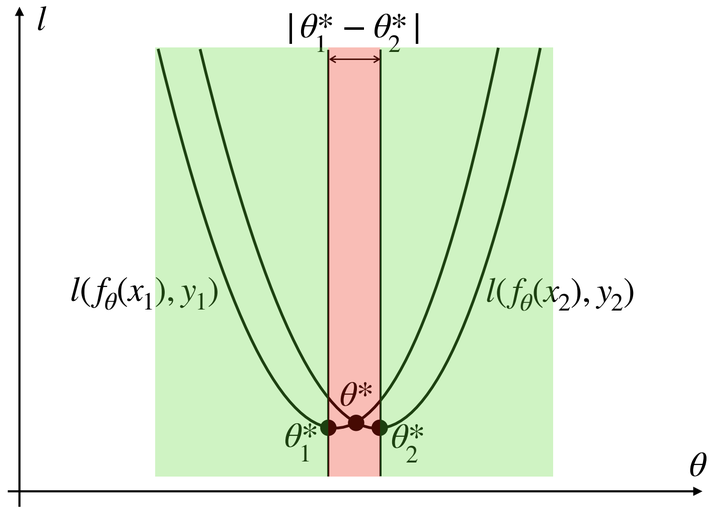
Performance Estimation Heuristics

GradSign

- **Intuition**: A good model has **denser** sample-wise local minima, i.e. the distance between local minima points θ_1^* and θ_2^* should be smaller. In this case, there is a higher probability in (b) that the gradients of different samples have the same sign (green region).



(a) Optimization landscape with **sparser** sample-wise local optima corresponding to **worse** $J(\theta^*)$.



(b) Optimization landscape with **denser** sample-wise local optima corresponding to **better** $J(\theta^*)$.

GradSign: Model Performance Inference with Theoretical Insights [Zhang and Jia, ICLR 2022]

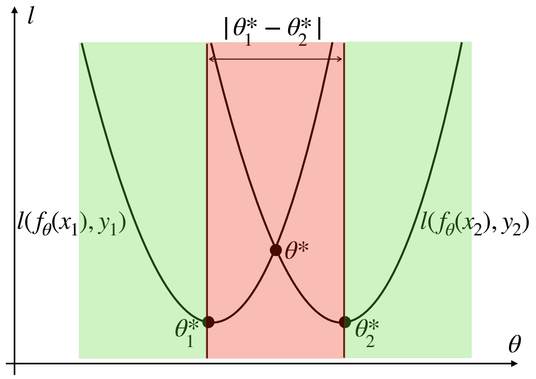
Performance Estimation Heuristics

GradSign

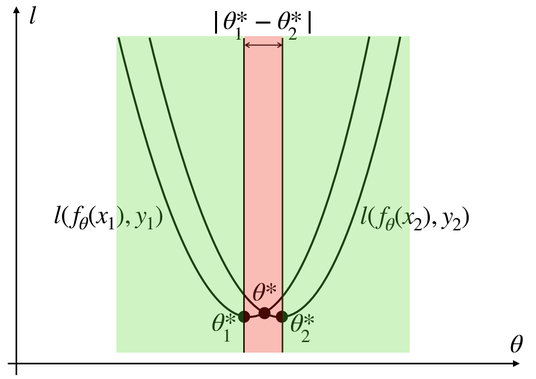
- **Implementation:**

Algorithm 1: GradSign

Result: GradSign score τ_f for a function class f_θ
Given $\mathcal{S} = \{(x_i, y_i)\}_{i\in[n]}$, randomly select initialization point θ_0;
Initialize $g[n, m]$;
for $i = 1, 2, \cdots, n$ **do** i, n corresponds to samples
  for $k = 1, 2, \cdots, m$ **do** k, m corresponds to layers
    $g[i, k] = \text{sign}([\nabla_\theta l(f_\theta(x_i), y_i)|_{\theta_0}]_k)$
  end
end
$\tau_f = \sum_k |\sum_i g[i, k]|$; $|\sum_{i=1}^{n} g[i, k]| \leq n$, "=" is achieved only when the gradients at all samples have the same sign.
return τ_f



(a) Optimization landscape with **sparser** sample-wise local optima corresponding to **worse** $J(\theta^*)$.



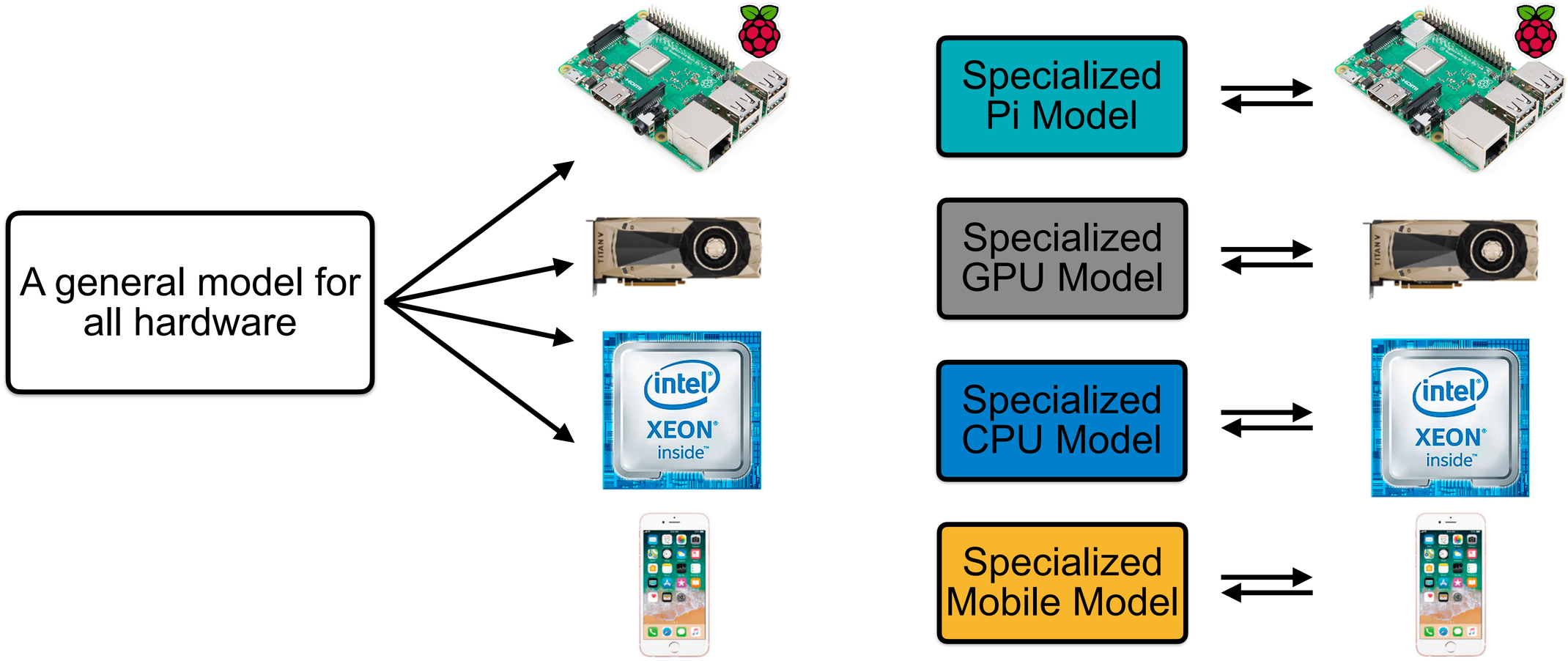
(b) Optimization landscape with **denser** sample-wise local optima corresponding to **better** $J(\theta^*)$.

GradSign: Model Performance Inference with Theoretical Insights [Zhang and Jia, ICLR 2022]

Hardware-Aware NAS

Efficiently search for efficient models

From General Model to Specialized Models

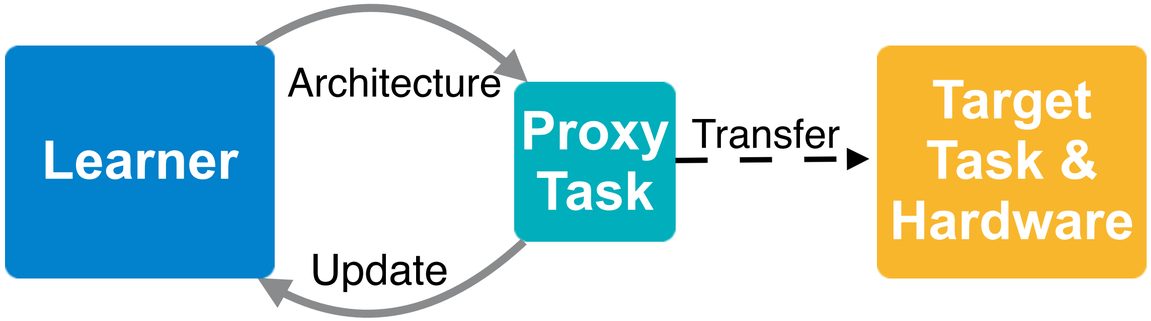


General purpose: inefficient

Specialized: efficient

Efficiently Search on Target Task & Hardware

Proxy task: sub-optimal



Previous neural architecture search (NAS) is **very expensive**.

- NASNet: 48,000 GPU hours, even on Cifar ≈ 5 years on single GPU
- DARTS: 100GB GPU memory if directly search on ImageNet

Therefore, previous work have to utilize ***proxy tasks***:

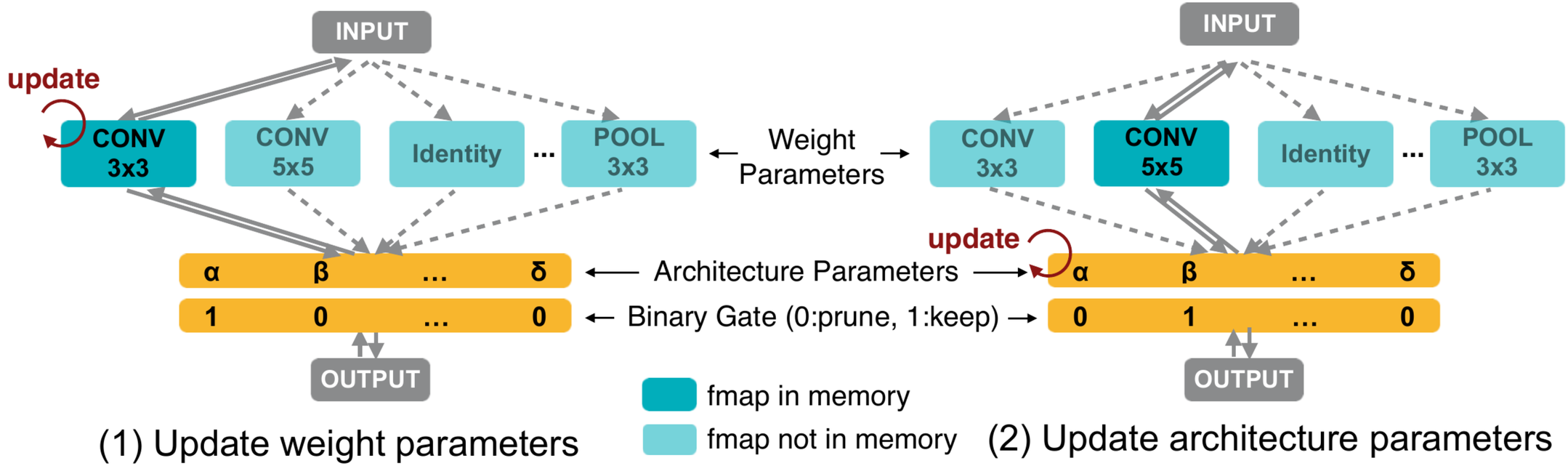
- CIFAR-10
- Small architecture space (e.g. low depth)
- Fewer epochs training
- Flops and parameter counts

Fidelity ↓

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Efficiently Search on Target Task & Hardware

ProxylessNAS



Build the over-parameterized network **with all candidate paths**

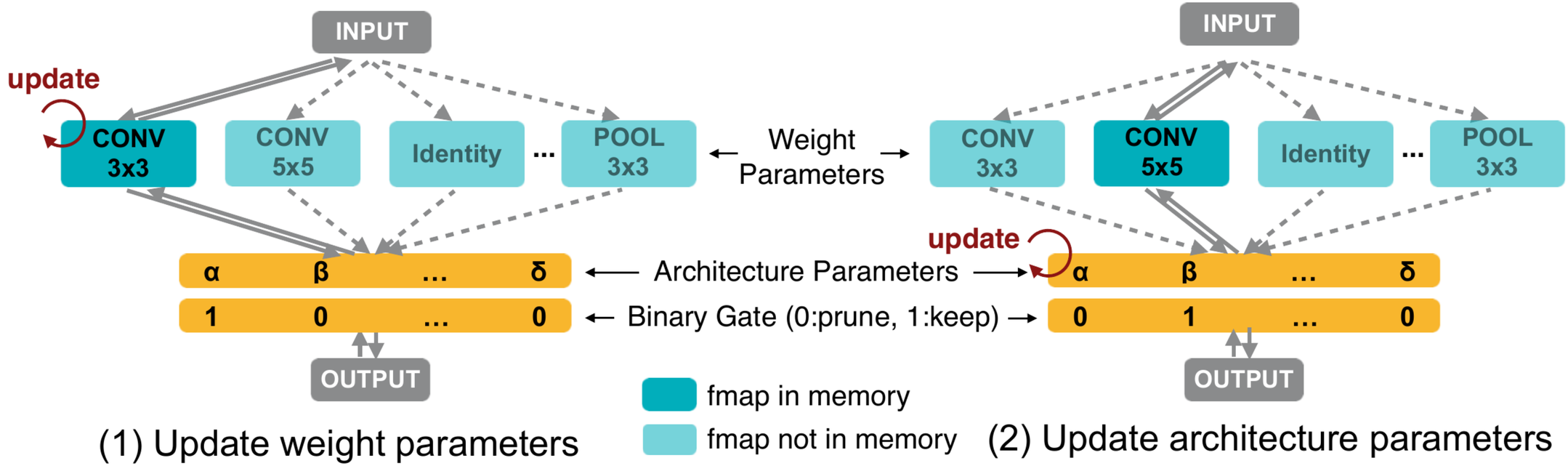
Simplify NAS to be a **single training process** of a over-parameterized network.

Pruning redundant paths based on architecture parameters

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Efficiently Search on Target Task & Hardware

ProxylessNAS



Binarize the architecture parameters and allow only **one path of activation to be active**.

Thereby, the memory footprint reduces from **O(N) to O(1)**.

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Efficiently Search on Target Task & Hardware

ProxylessNAS

Previous neural architecture search (NAS) is **very expensive**.

- NASNet: 48,000 GPU hours, even on Cifar ≈ 5 years on single GPU
- DARTS: 100GB GPU memory if directly search on ImageNet

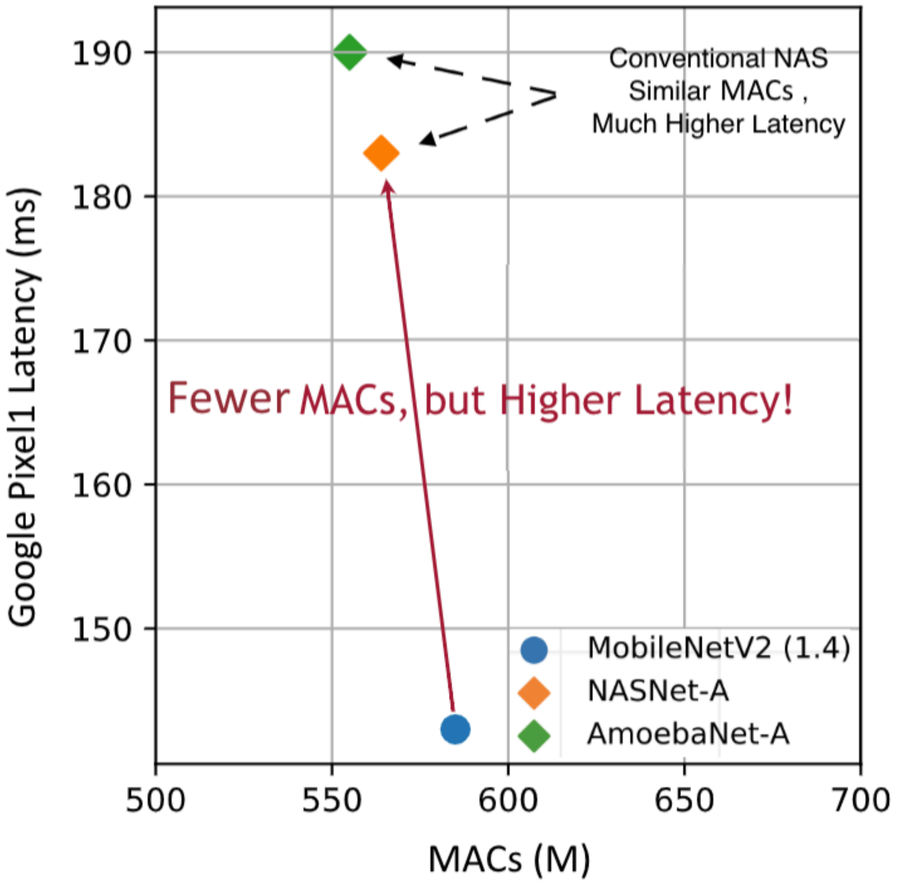
Therefore, previous work have to utilize ***proxy tasks***:

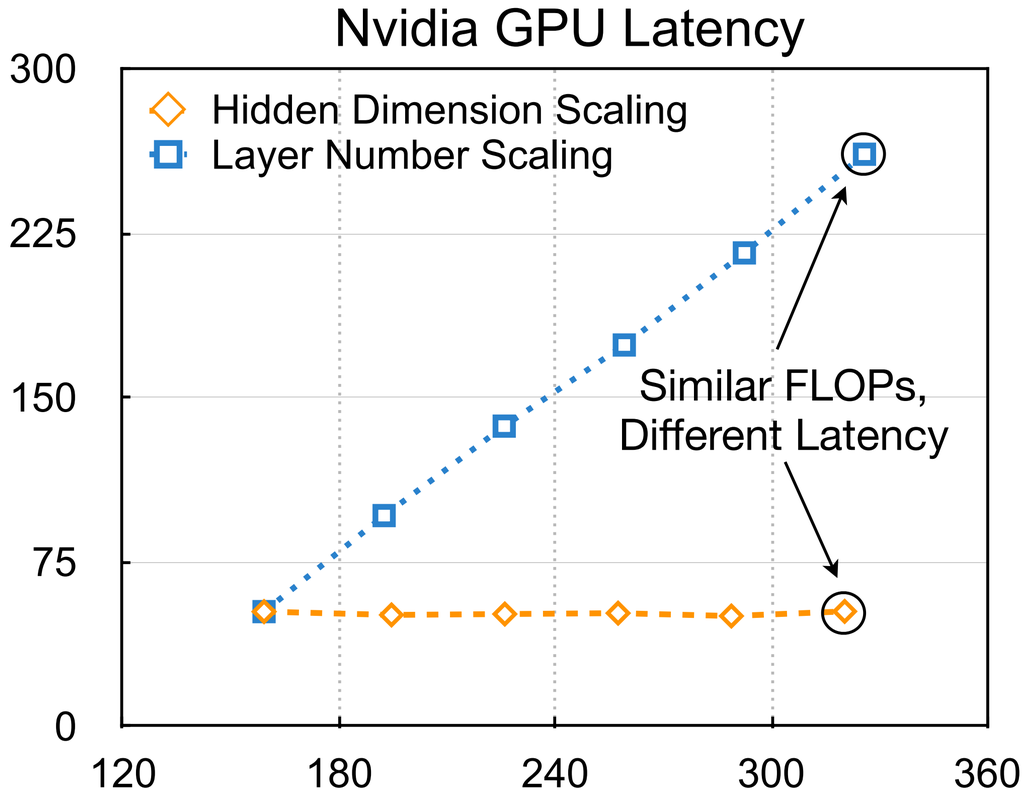
• CIFAR-10	→ • ImageNet
• Small architecture space (e.g. low depth)	→ • Large architecture space
• Fewer epochs training	→ • Full training
• Flops and parameter counts	→ • Profiled Latency

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Search for Efficient Model

MACs != Real Hardware Efficiency

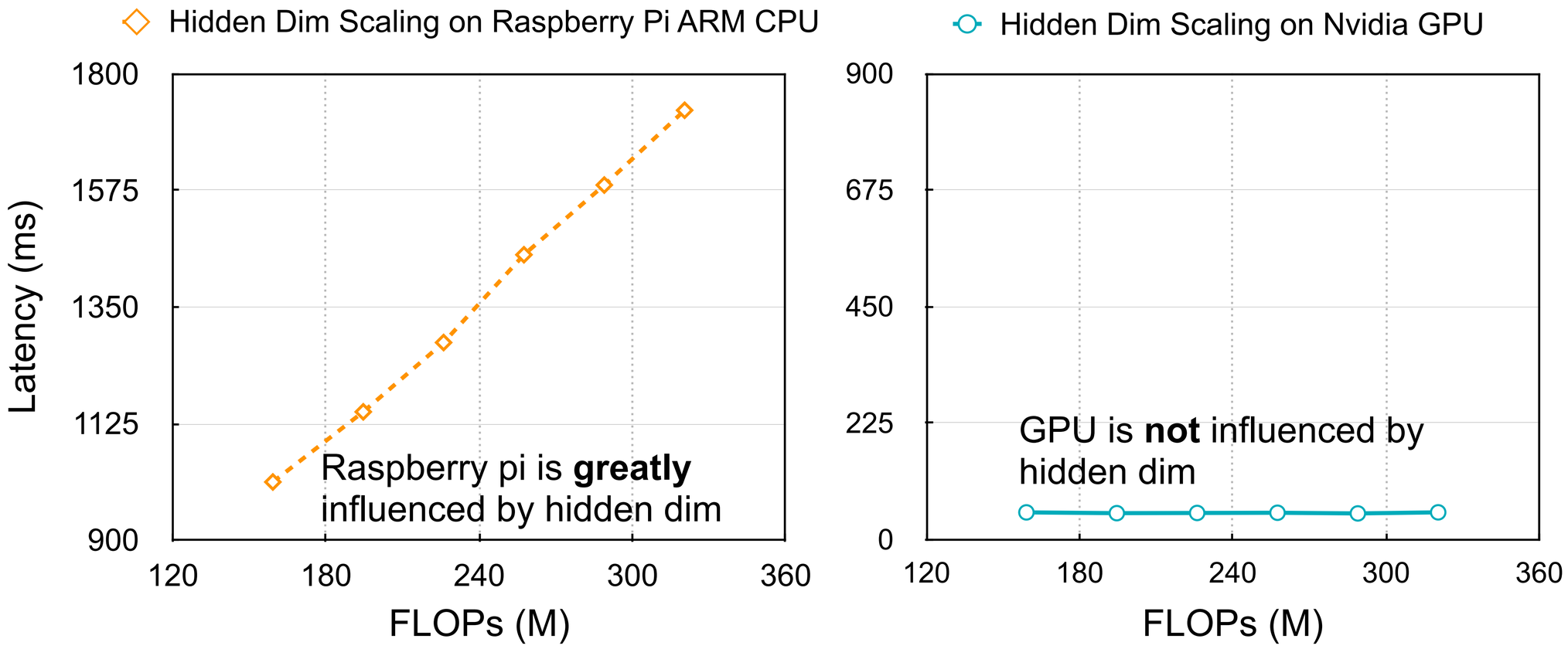




ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]

Search for Efficient Model

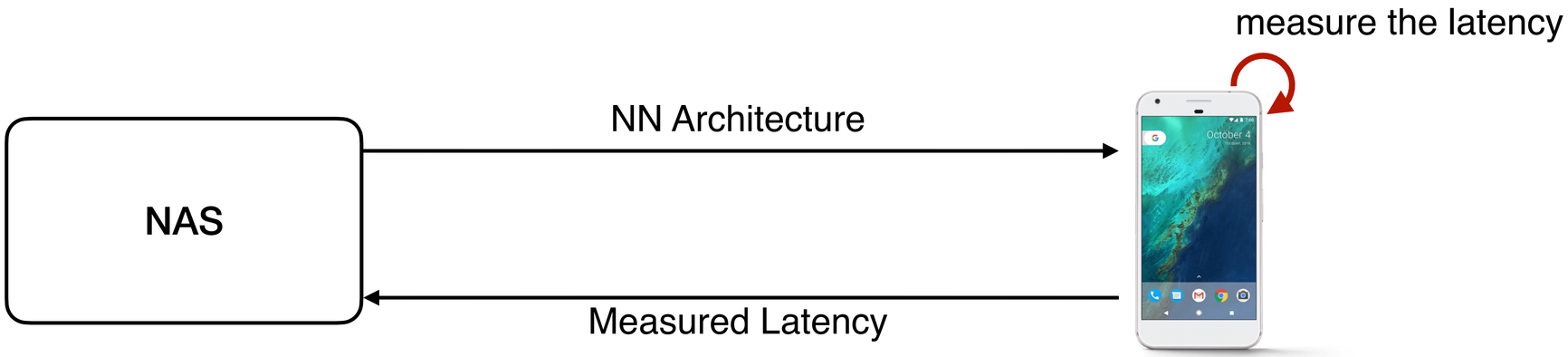
MACs != Real Hardware Efficiency



HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]

Incorporate Hardware Feedback into NAS

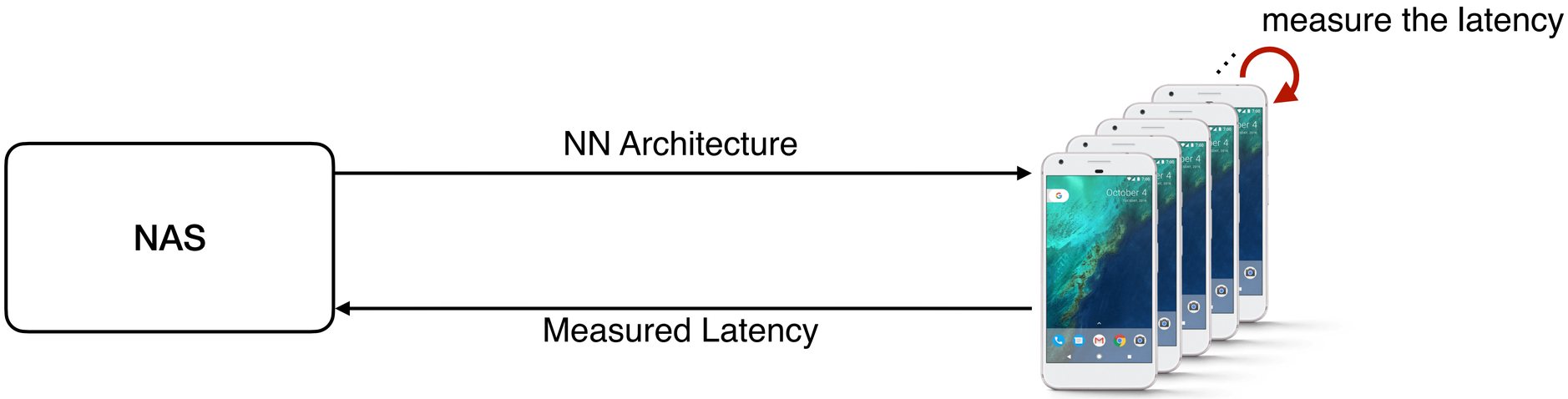
Measure the latency on device: slow



Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang *et al.*, CVPR 2019]

Incorporate Hardware Feedback into NAS

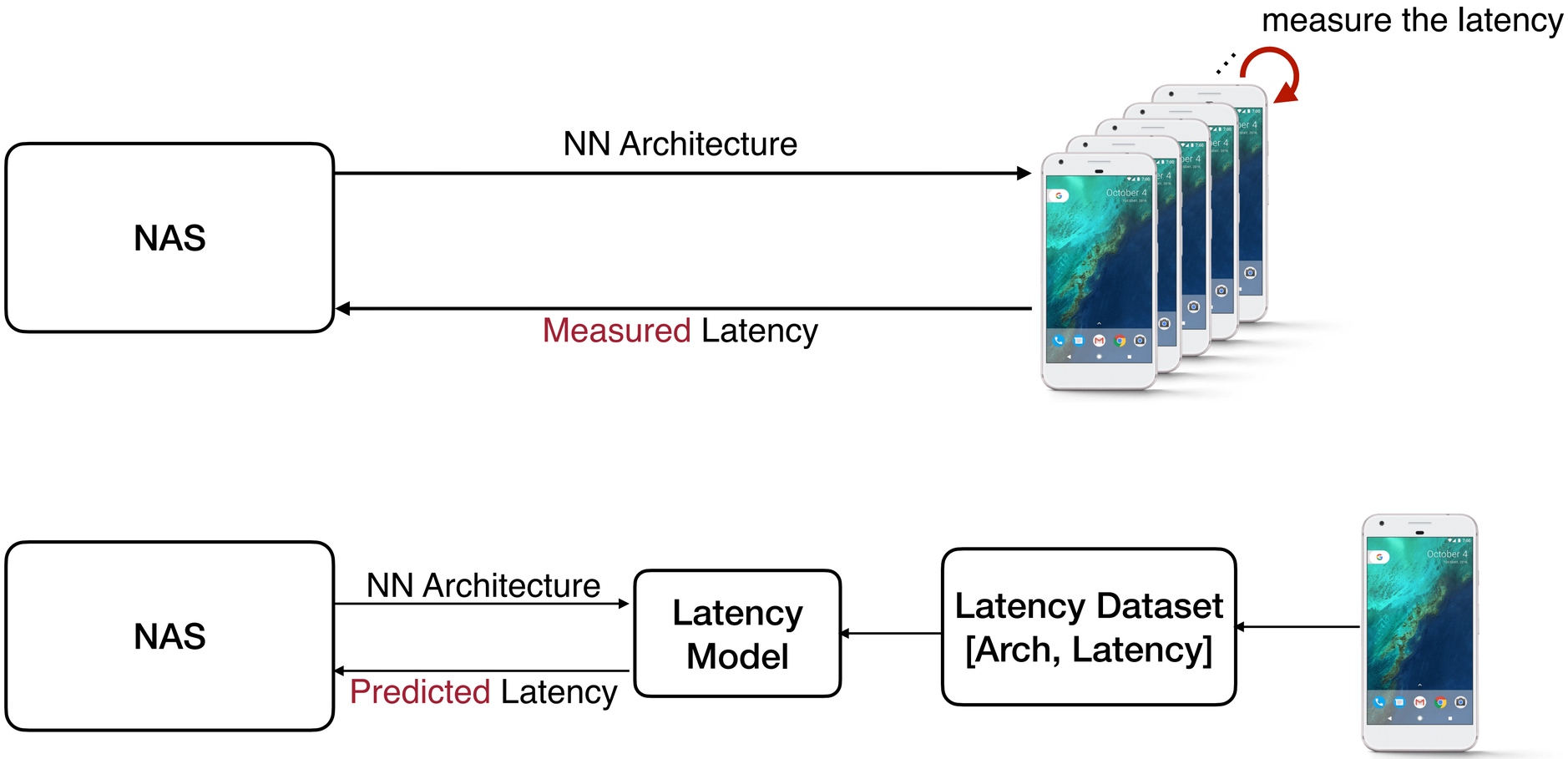
Measure the latency on device: expensive



Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang *et al.*, CVPR 2019]

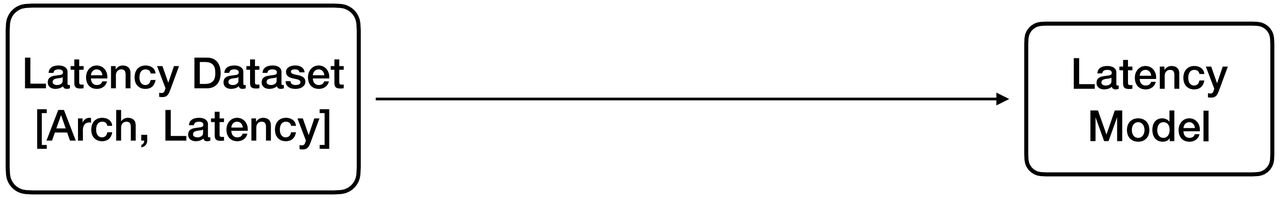
Incorporate Hardware Feedback into NAS

Latency prediction: fast and low cost



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

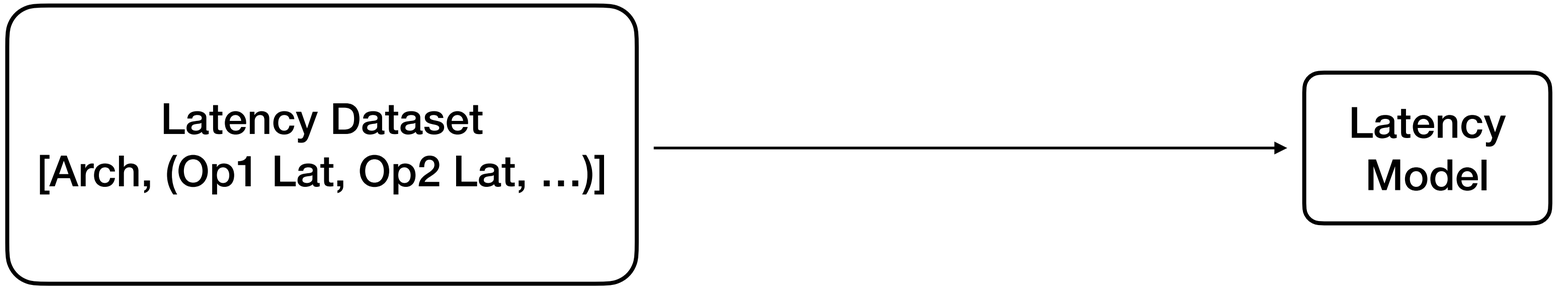
Latency Prediction



- Layer-wise latency profiling: latency lookup table

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

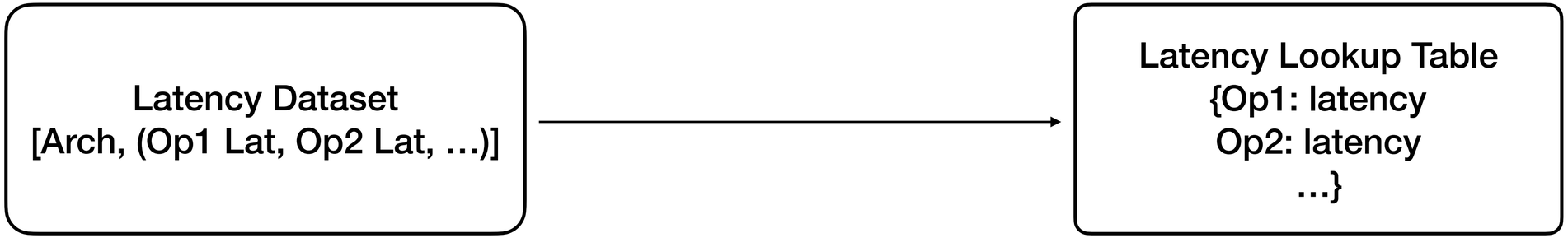
Latency Prediction



- Layer-wise latency profiling: latency lookup table

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Latency Prediction



- Layer-wise latency profiling: latency lookup table

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Latency Prediction

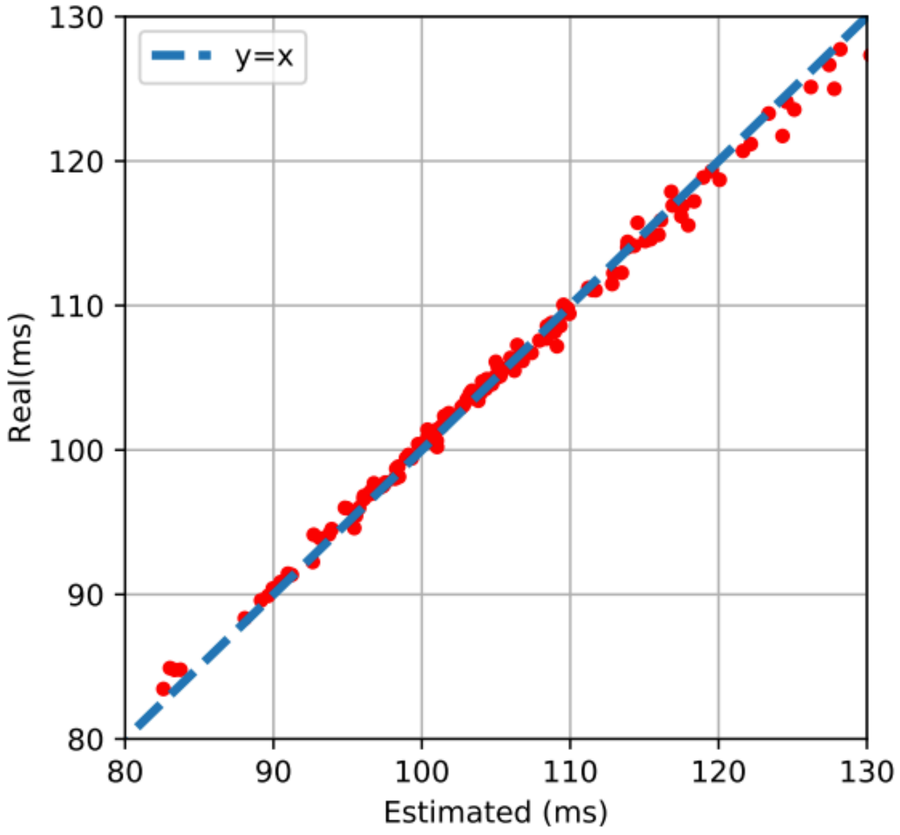
Latency Lookup Table

```
expanded_conv-input:112x112x24-output:112x112x24-expand:24-kernel:3-stride:1-idskip:1-se:0-hs:0:  <-- Key
  count: 500
  mean: 1.7728020000000002  <-- Predicted latency
  std: 0.026630561315901715
expanded_conv-input:112x112x24-output:56x56x32-expand:144-kernel:3-stride:2-idskip:0-se:0-hs:0:
  count: 54
  mean: 4.812240740740741
  std: 0.08956353738593026
expanded_conv-input:112x112x24-output:56x56x32-expand:144-kernel:5-stride:2-idskip:0-se:0-hs:0:
  count: 63
  mean: 6.585063492063491
  std: 0.21637063094840722
```

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

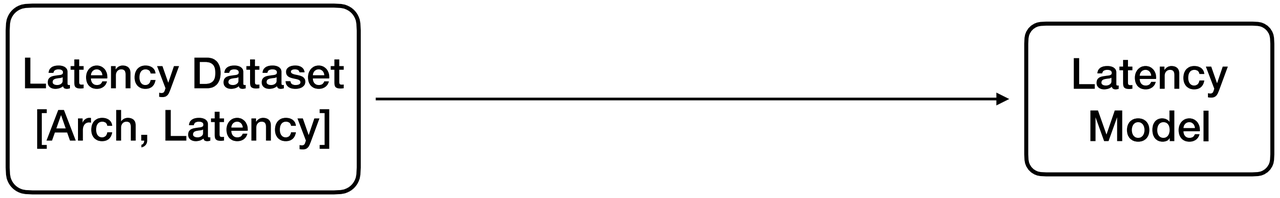
Latency Prediction

Latency Lookup Table

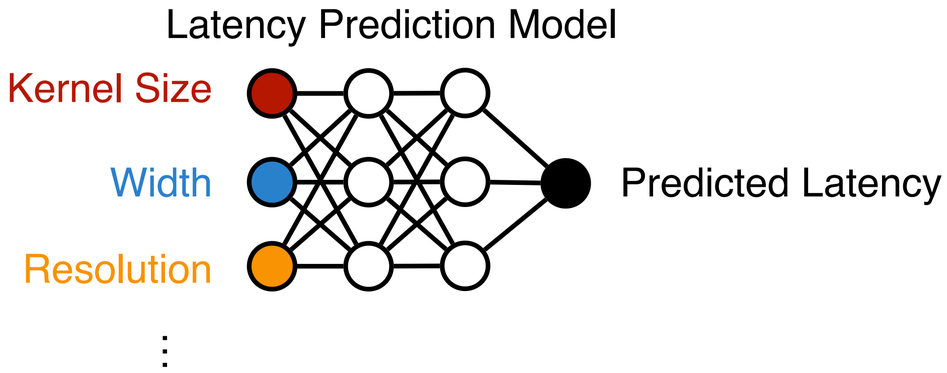


ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Latency Prediction

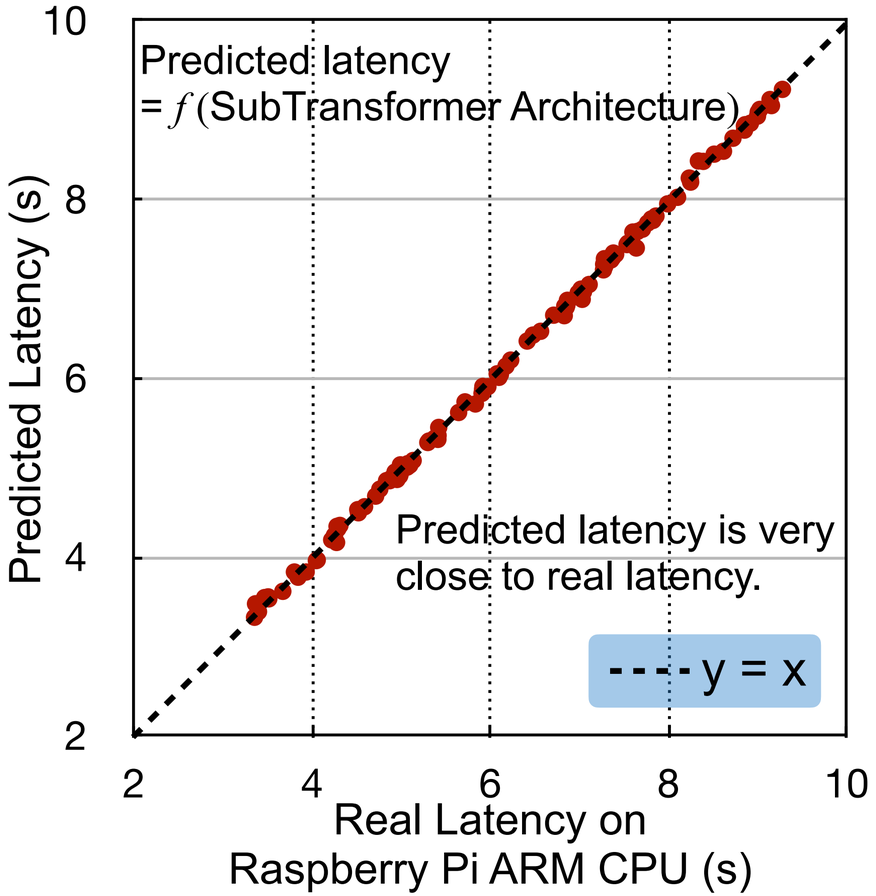


- Network-wise latency profiling: latency prediction model



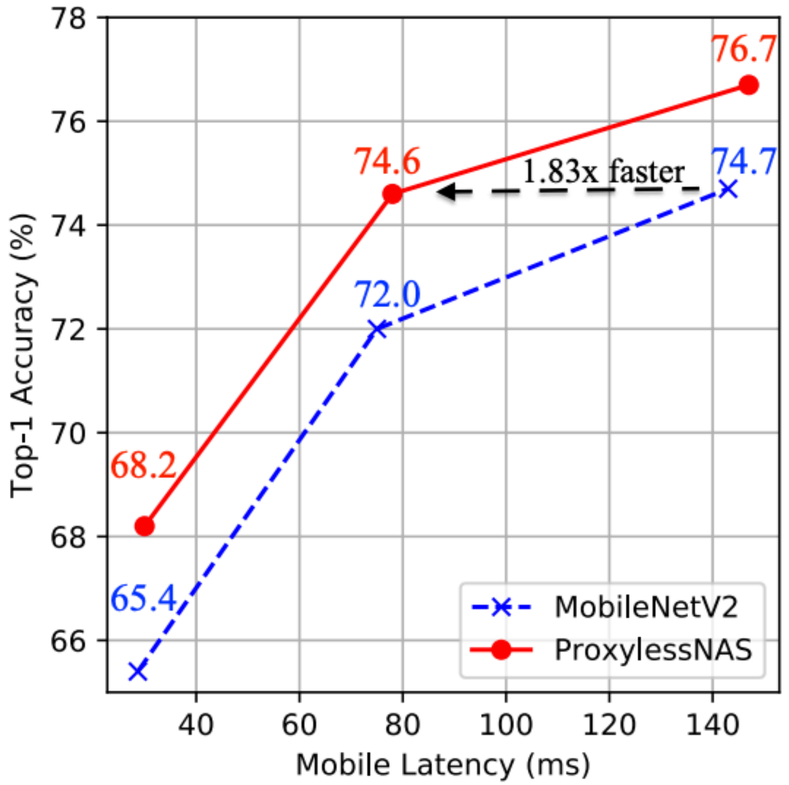
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]

Latency Prediction



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]

Specialized Models for Different Hardware



Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	38.3ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	**75.1**	**92.5**	**5.1ms**

- The specialized model is 1.83x faster than the non-specialized counterpart on mobile.
- The improvement is larger on GPU.

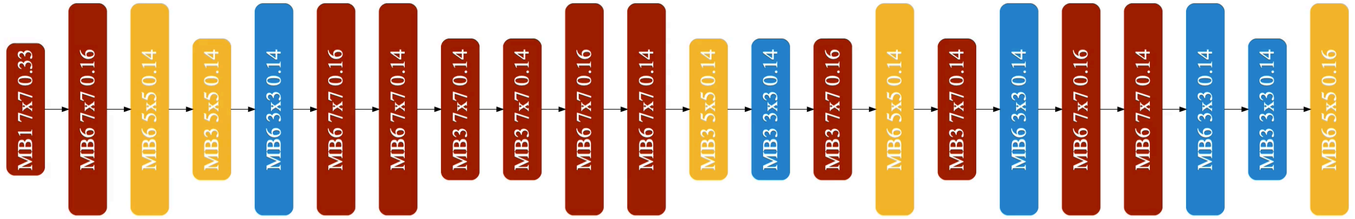
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Specialized Models for Different Hardware

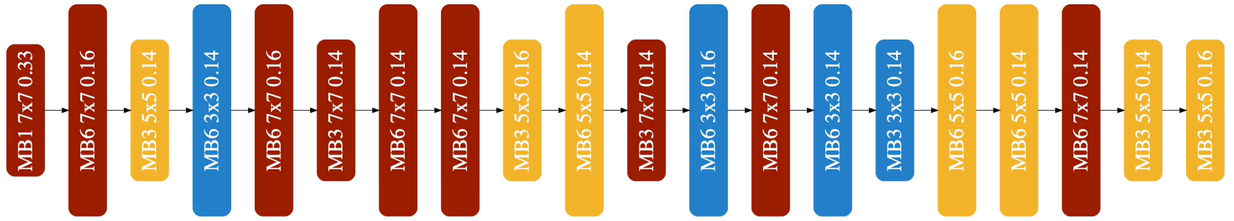
Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Specialized Models for Different Hardware

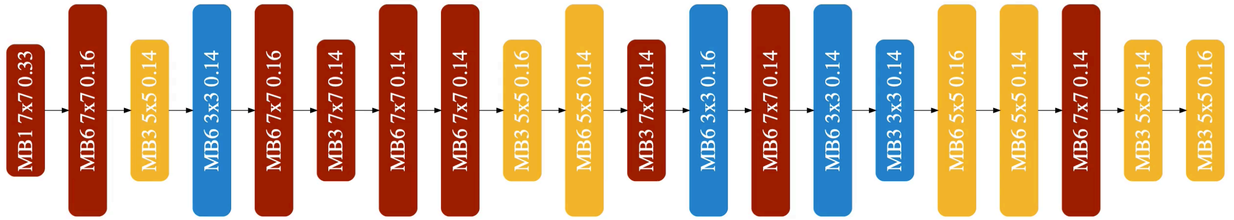


(1) The history of finding efficient Mobile model



(2) The history of finding efficient CPU model

Epoch-00

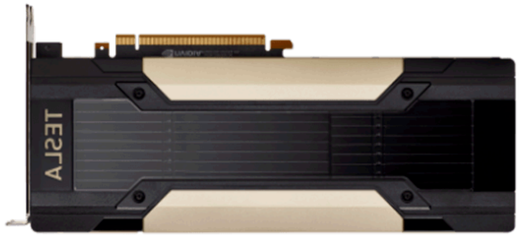


(3) The history of finding efficient GPU model

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]

Diverse Platforms and Efficiency Constraints

Diverse Hardware Platforms



...

Cloud AI (10^{12} FLOPS)

Mobile AI (10^{9} FLOPS)

Tiny AI (10^{6} FLOPS)

For many devices:

  For search episodes:

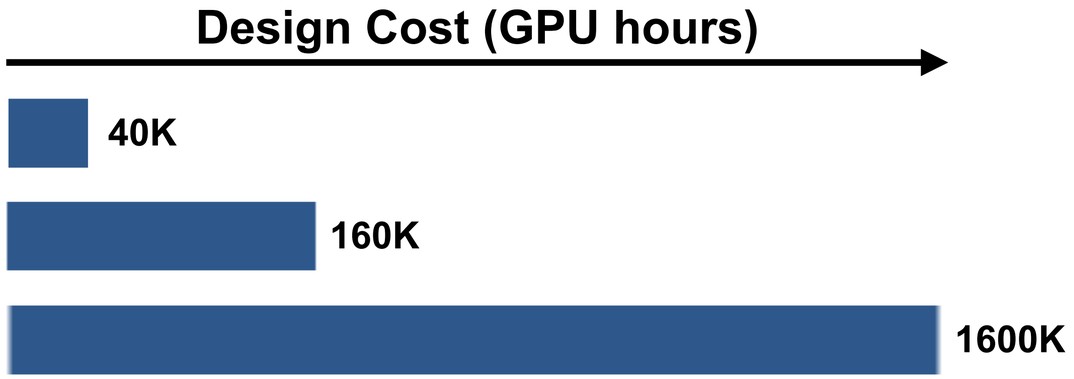
    For training iterations:

      forward-backward(); **Expensive!!**

    If good_model: **break;**

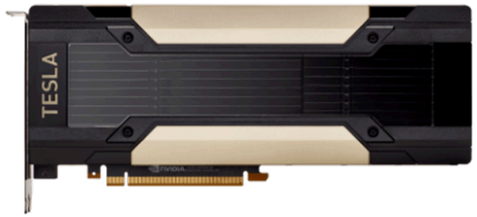
For post-search training iterations:

  forward-backward(); **Expensive!!**

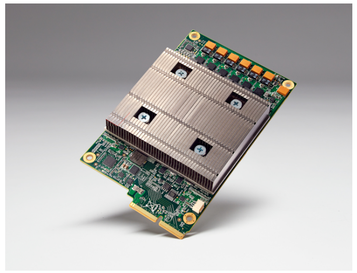


Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang *et al.*, CVPR 2019]

Improve the productivity of designing models



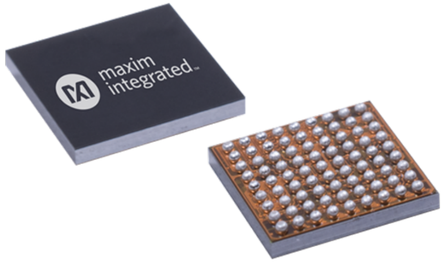
GPU



TPU

DSP

...



Accelerator

Existing models are heavily optimized for GPUs

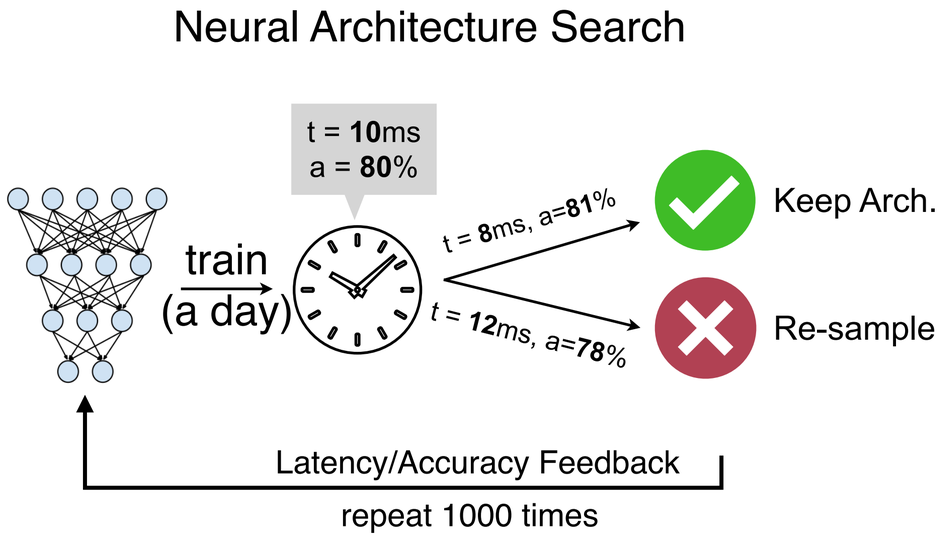
But doesn't fit well on other platforms

Hardware-aware NAS with Once-for-All Network

Rather than training each network from scratch, can we train multiple models at the same time?

1. Train a network, get accuracy and latency **Expensive**
2. Repeat, select best

1. Train a once-for-all network, sparsely activated
2. Select sub-network, get accuracy and latency
3. Repeat 2, select best **Cheap**



Automatic Design:
Synthesize NN to fit latency/accuracy/memory constraints

NAS : Neural net = EDA tool : Circuit

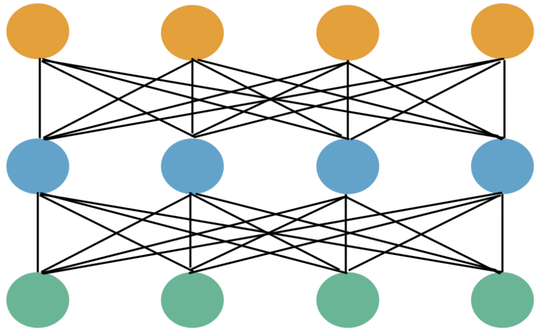
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

Train once, get many

Reduce the design cost

Fit diverse hardware constraints



OFA network contains many child networks that are sparsely activated

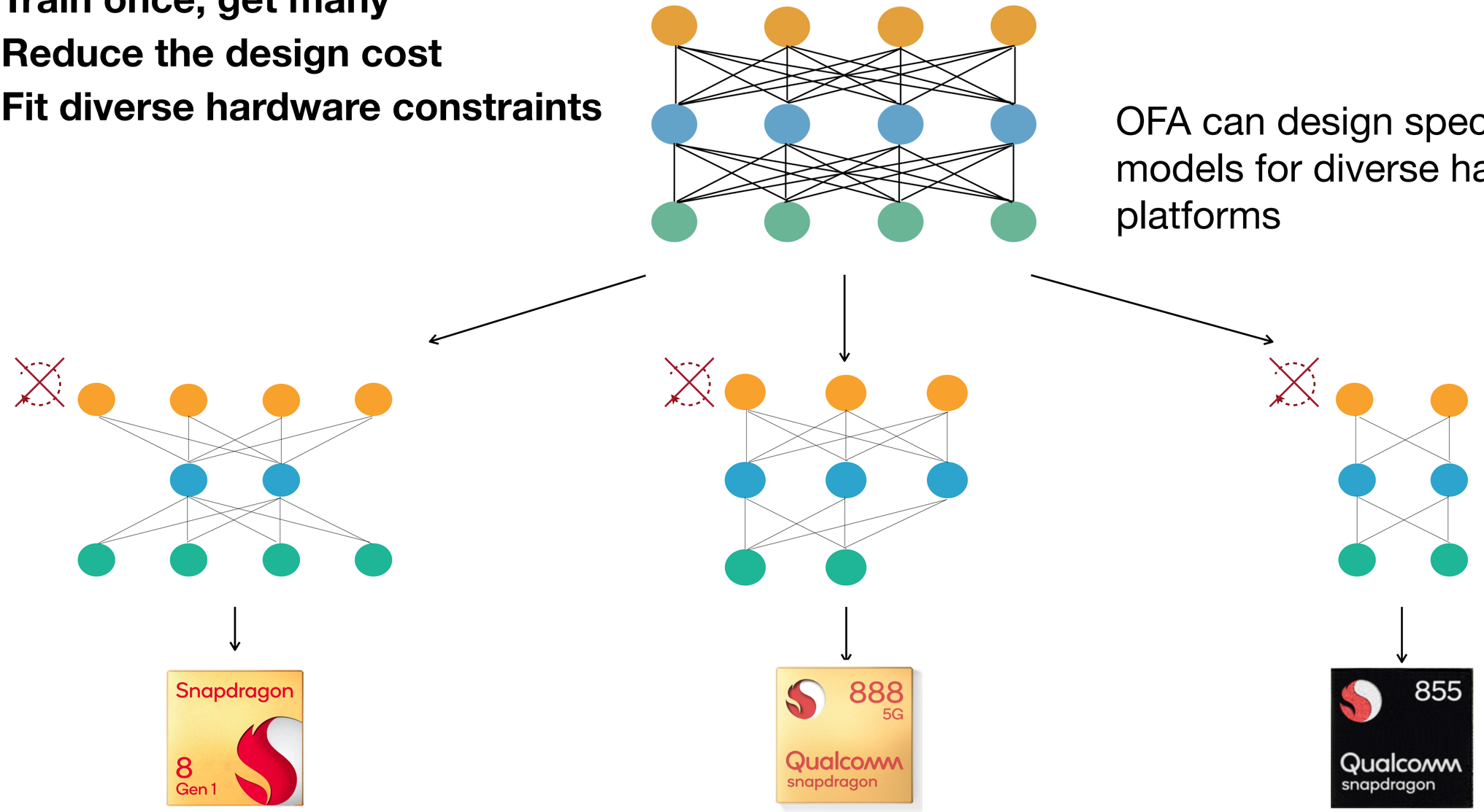
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

Train once, get many

Reduce the design cost

Fit diverse hardware constraints



OFA can design specialized NN models for diverse hardware platforms

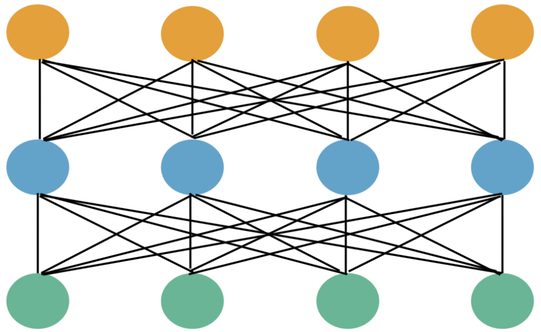
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

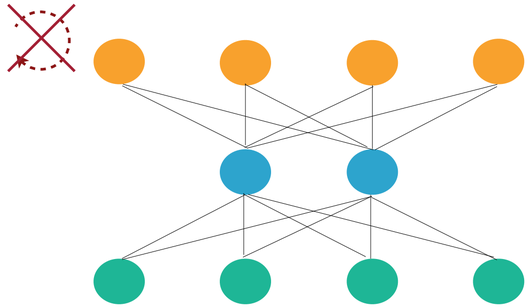
Train once, get many

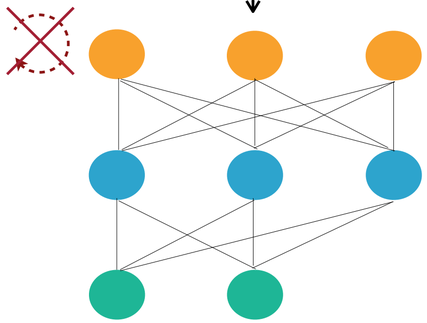
Reduce the design cost

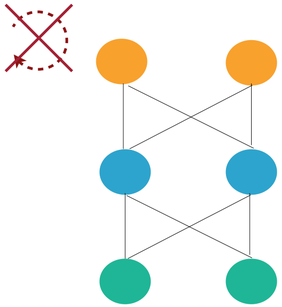
Fit diverse hardware constraints



OFA can design specialized NN models for diverse hardware platforms







Cortex M7
STM32H743
(512kB/2MB)

Cortex M7
STM32F746
(320kB/1MB)

Cortex M4
STM32F412
(256kB/1MB)

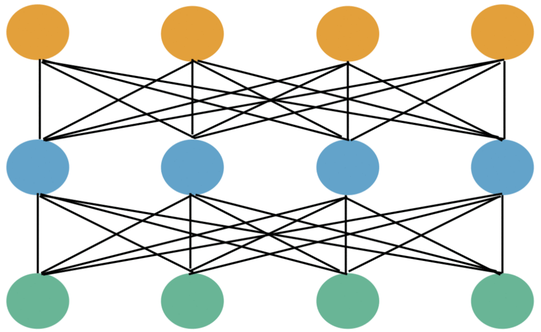
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

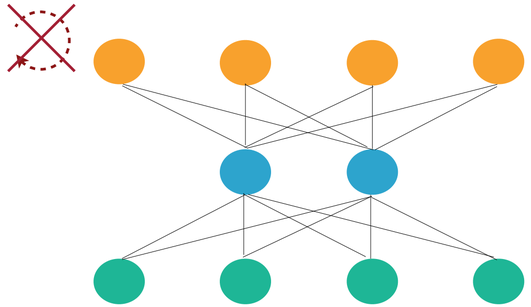
Train once, get many

Reduce the design cost

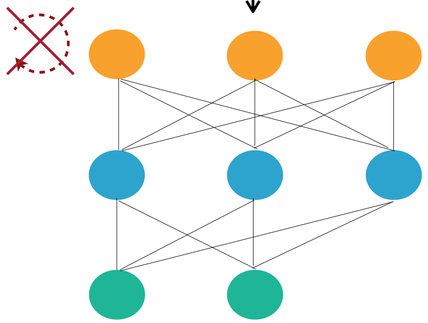
Fit diverse hardware constraints



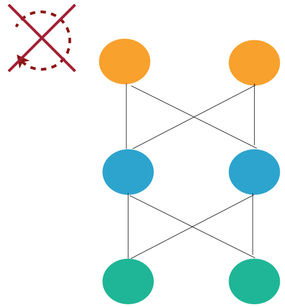
OFA can design specialized NN models for diverse hardware platforms



Samsung S22



Samsung S21



Samsung S20

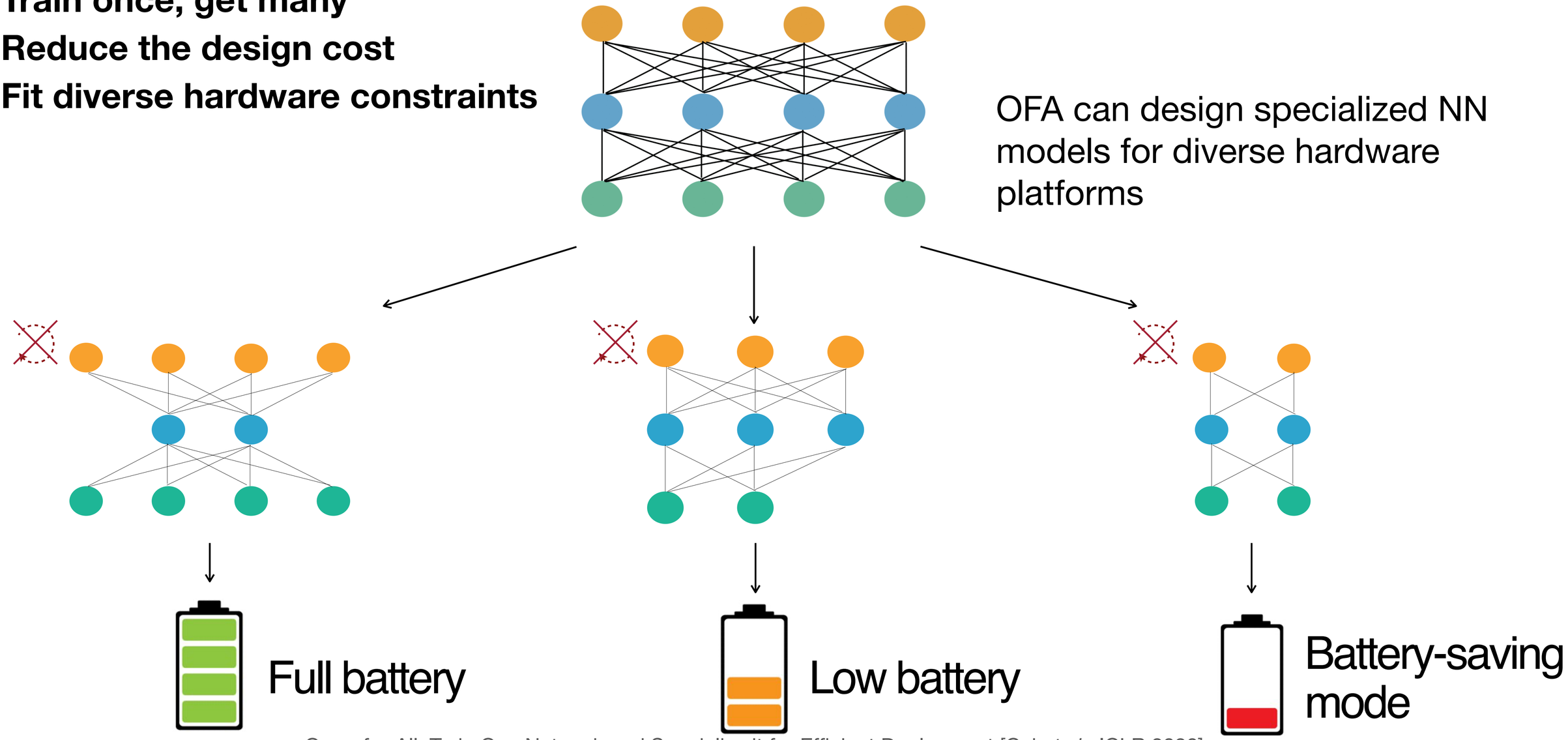
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

Train once, get many

Reduce the design cost

Fit diverse hardware constraints



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

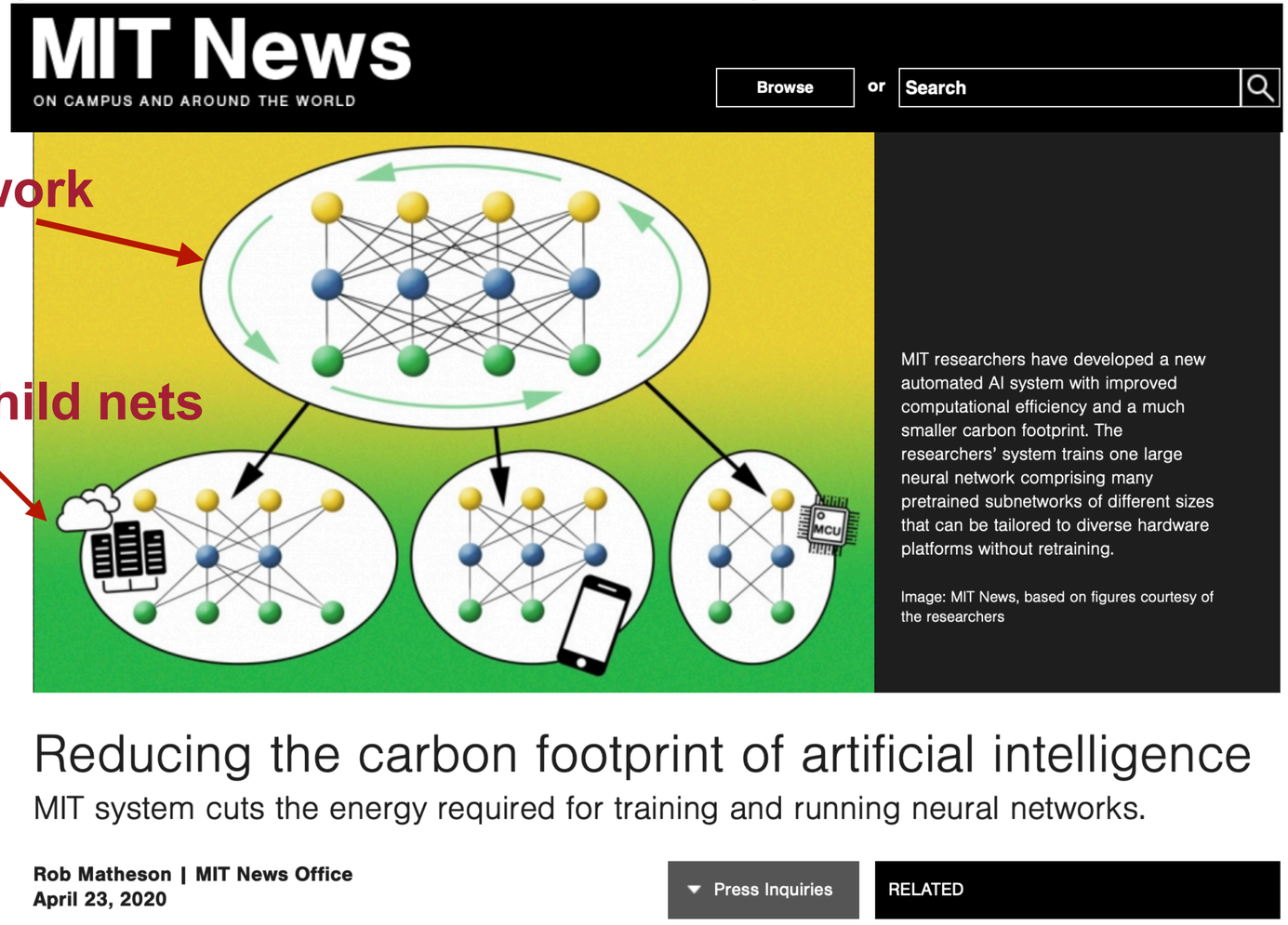
Once for All Network: Train 10^{19} networks at the same time

Design new efficient models rather than compressing existing models

Train once, get many, reduce the design cost

Once-for-All Network

Get many (10^{19}) child nets for free



OFA network contains many child networks that are sparsely activated

Child networks share the weights with the once-for-All network, trained jointly, amortize the training cost.

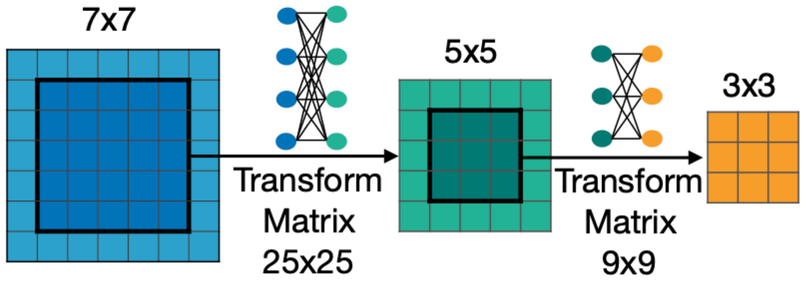
OFA can design specialized NN models for diverse hardware platforms

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

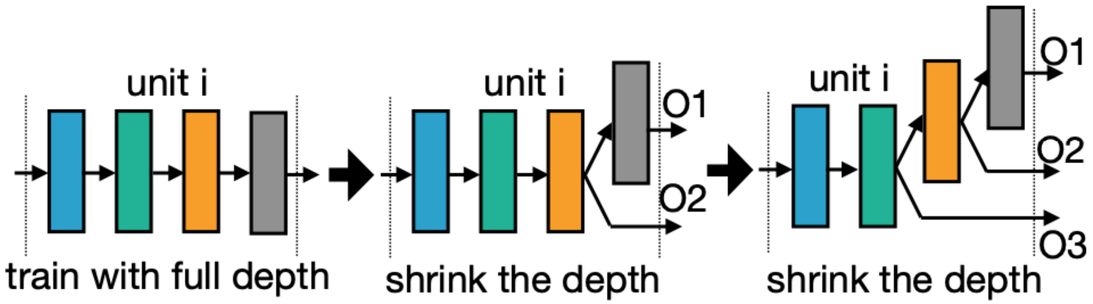
Once-for-All Network

progressively prune the kernel size, depth, resolution

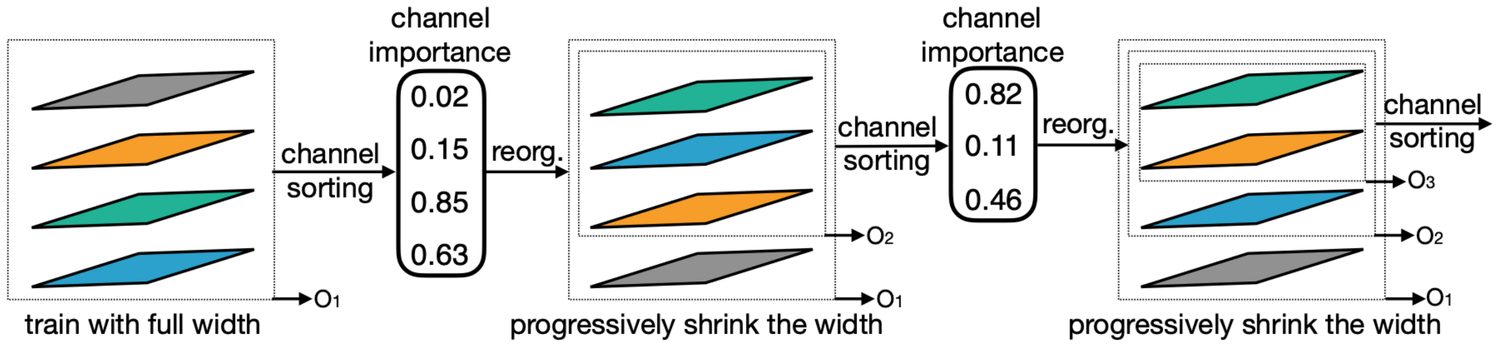
progressive shrinking:
kernel size



progressive shrinking:
layers



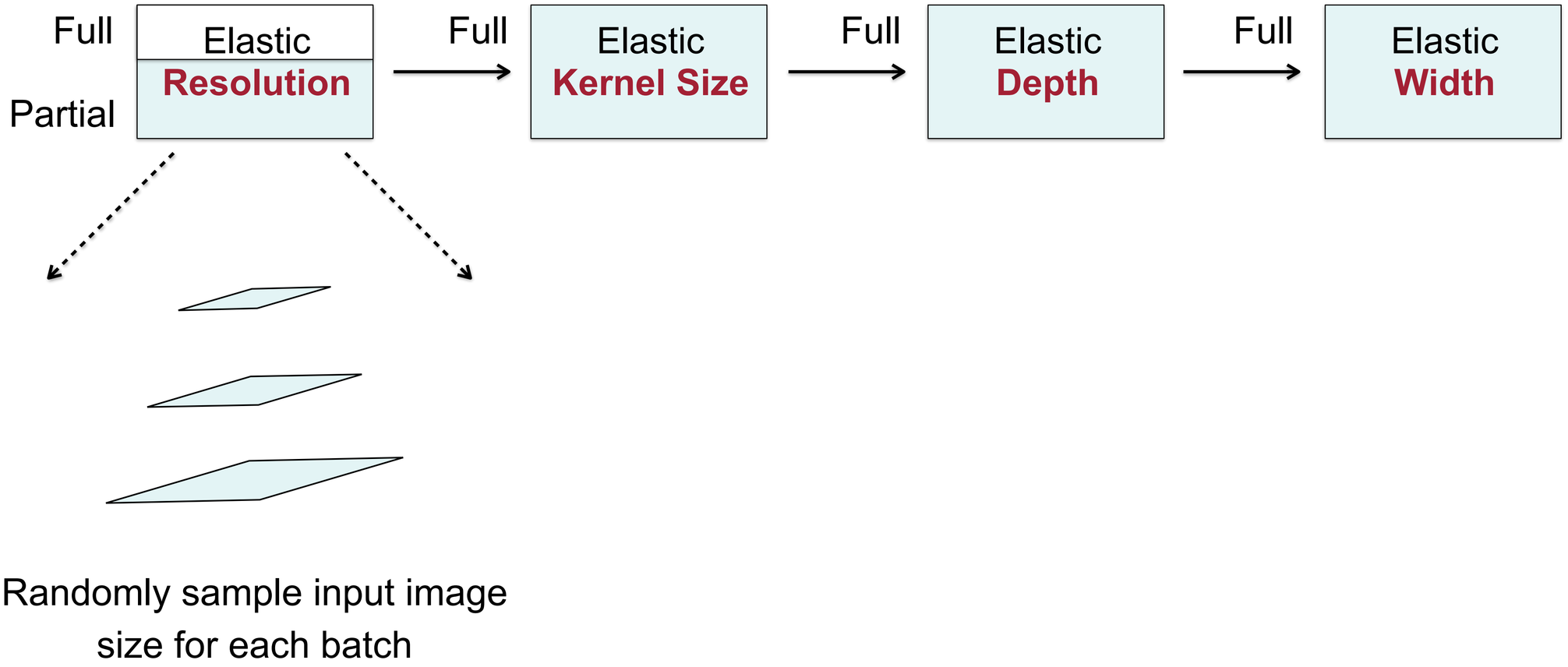
progressive shrinking:
channels



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

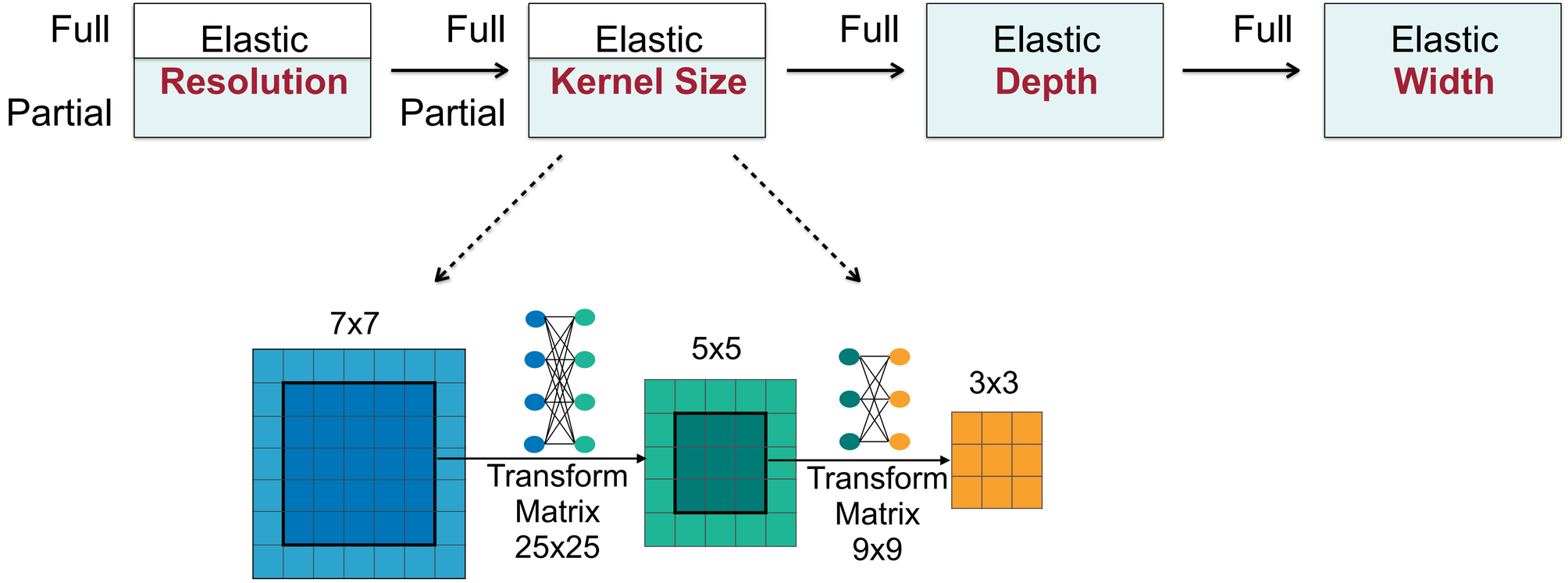
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

progressively prune the kernel size, depth, resolution



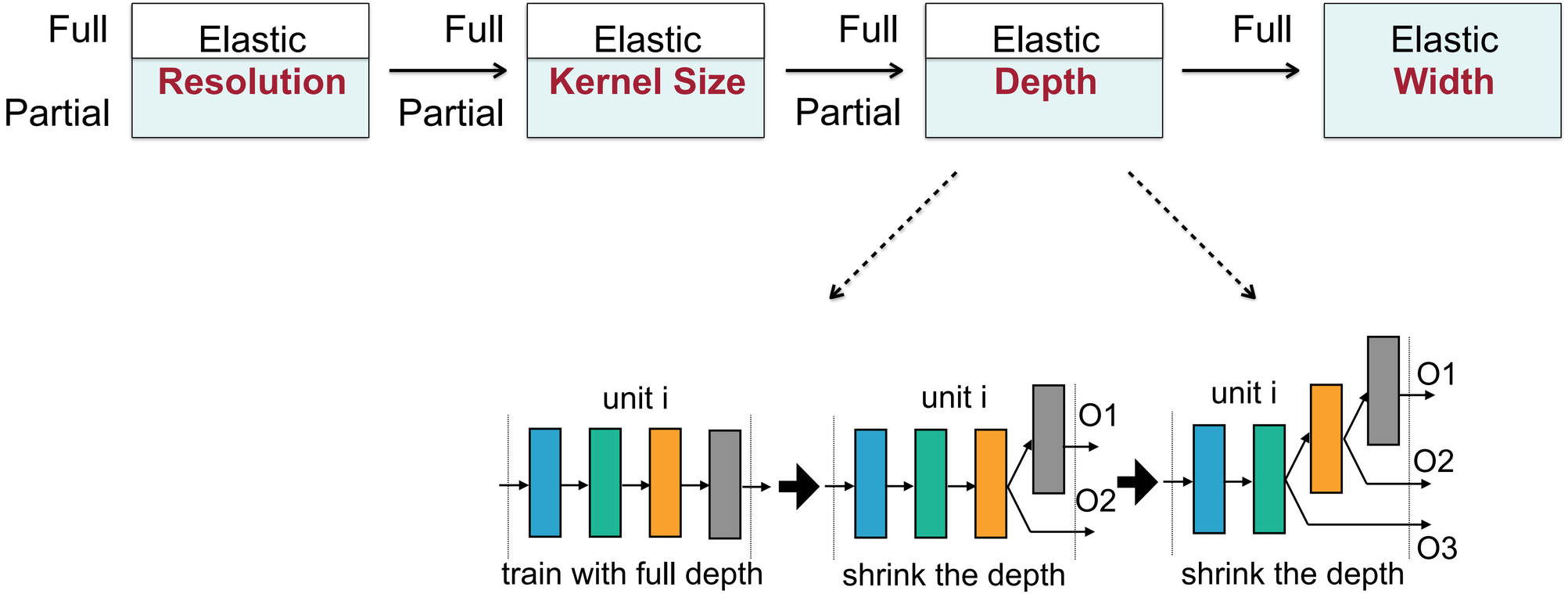
Start with full kernel size

Smaller kernel takes centered weights via a transformation matrix

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

progressively prune the kernel size, depth, resolution

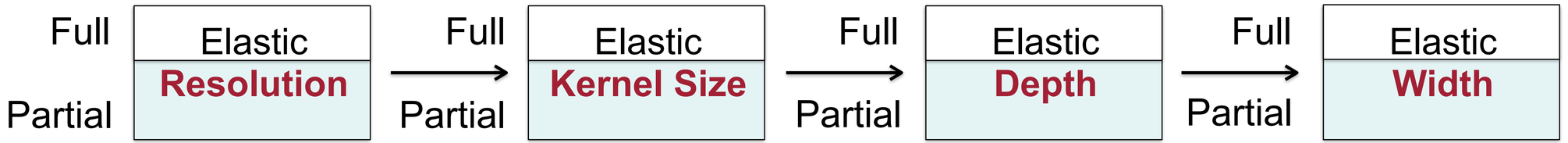


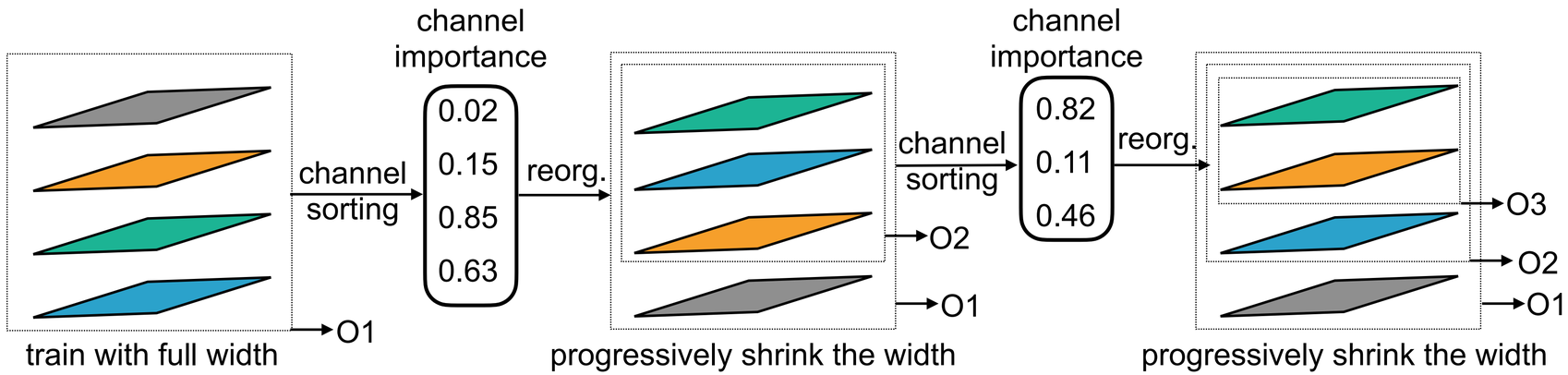
Gradually allow later layers in each unit
to be skipped to reduce the depth

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

Once-for-All Network

progressively prune the kernel size, depth, resolution





Gradually shrink the width

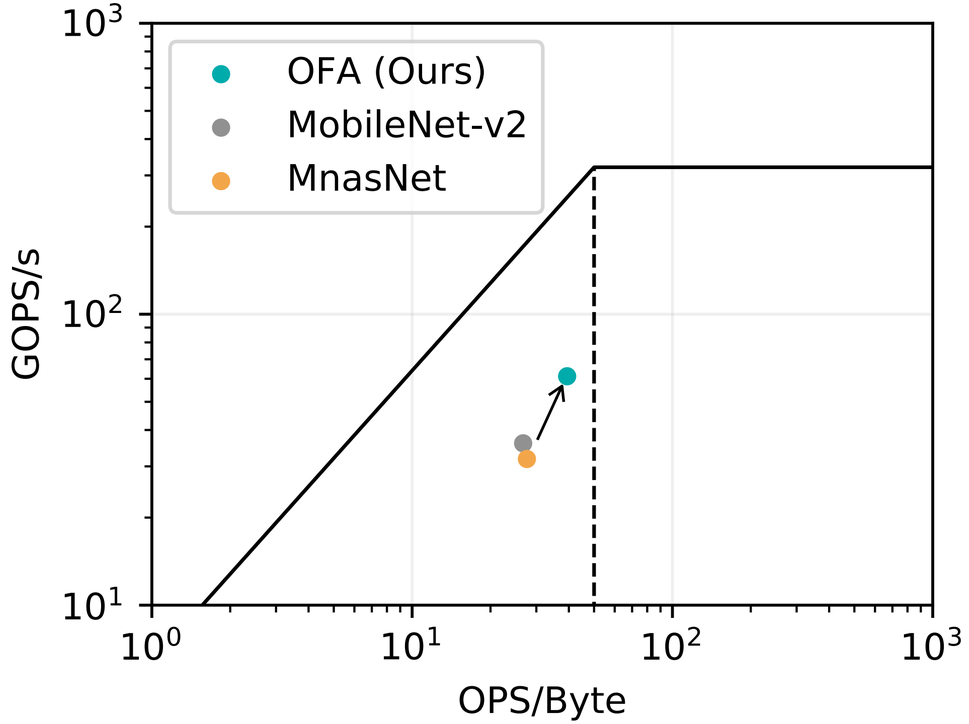
Keep the most important channels when shrinking via channel sorting

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

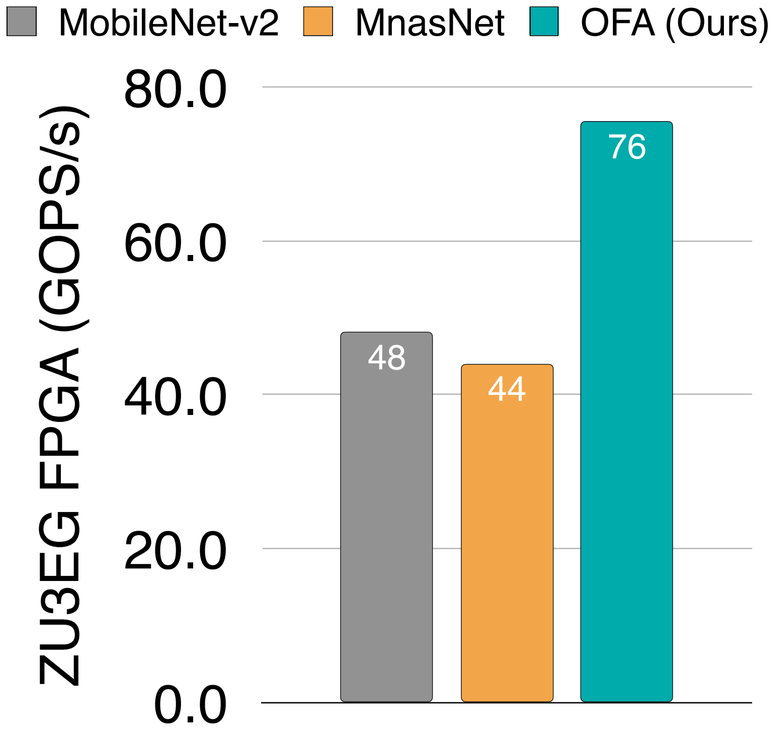
Once-for-All Network: Roofline Analysis

Computation is cheap; memory is expensive.

- OFA designed model has higher arithmetic intensity (Ops/Byte)
  - => less memory bounded
  - => higher utilization and performance without changing the RTL

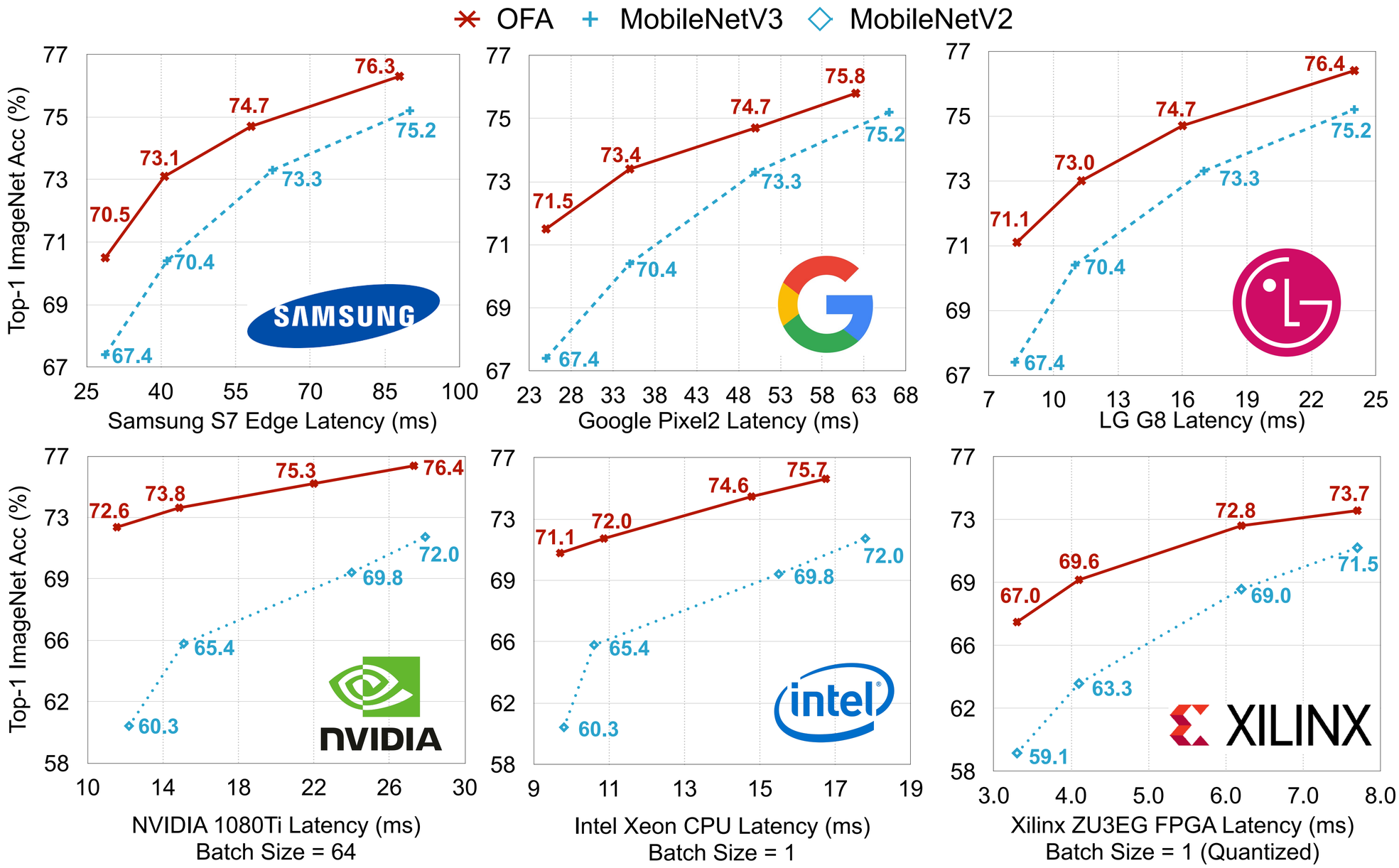


(b) on Xilinx ZU3EG FPGA



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

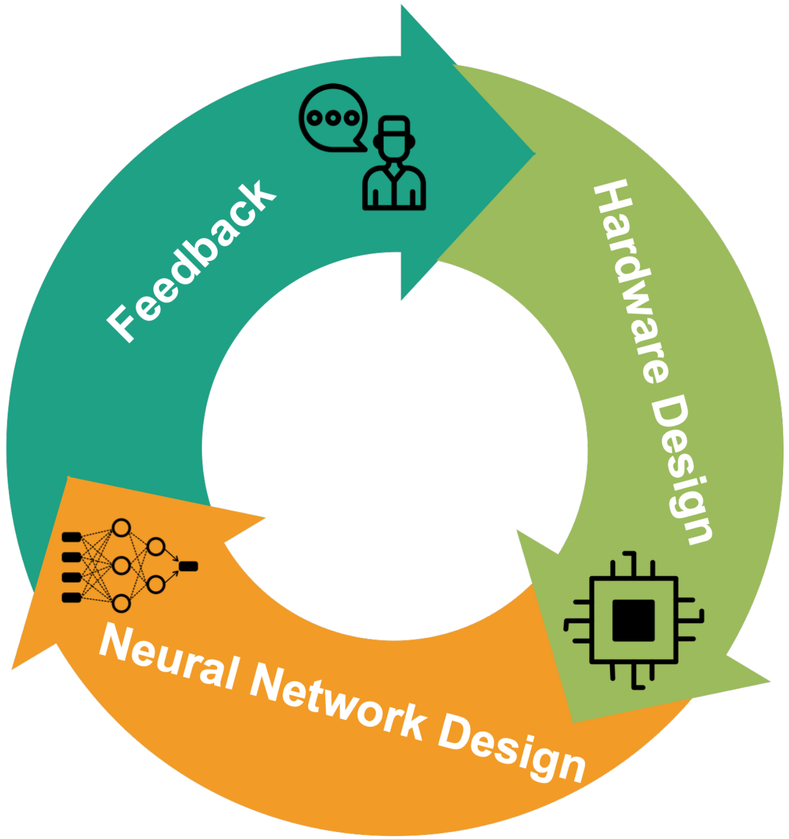
OFA on Diverse HW Platforms

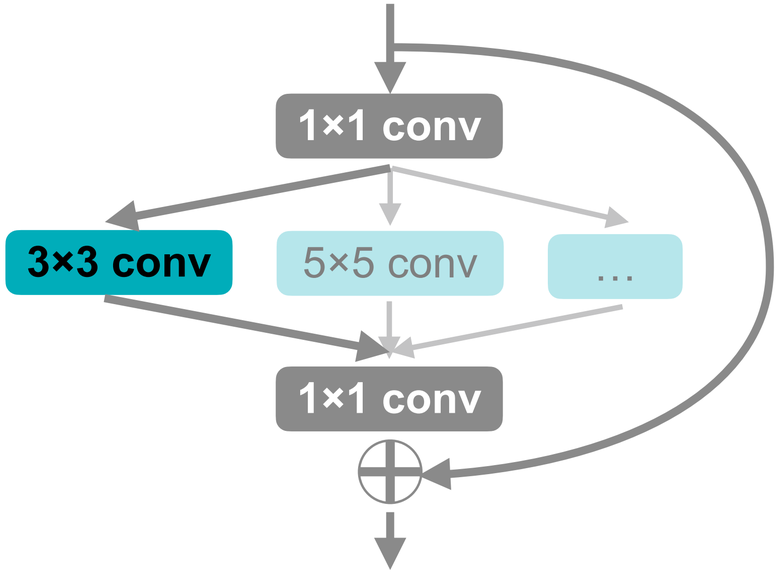


Neural Hardware Architecture Search

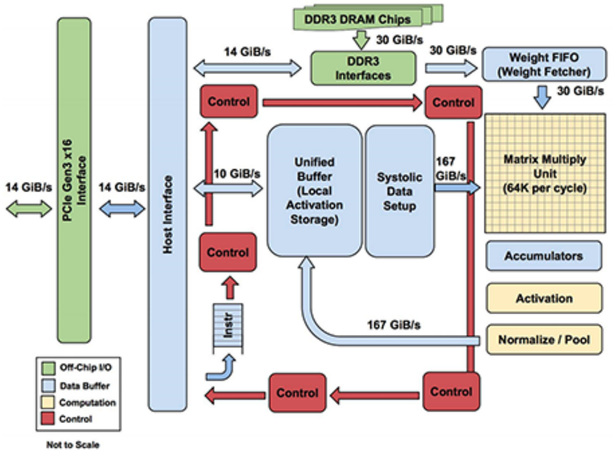
Accelerating Deep Learning Computing

Both **neural architecture** and **accelerator architecture** design are important to enable specialization and acceleration.





Neural Architecture



Accelerator Architecture

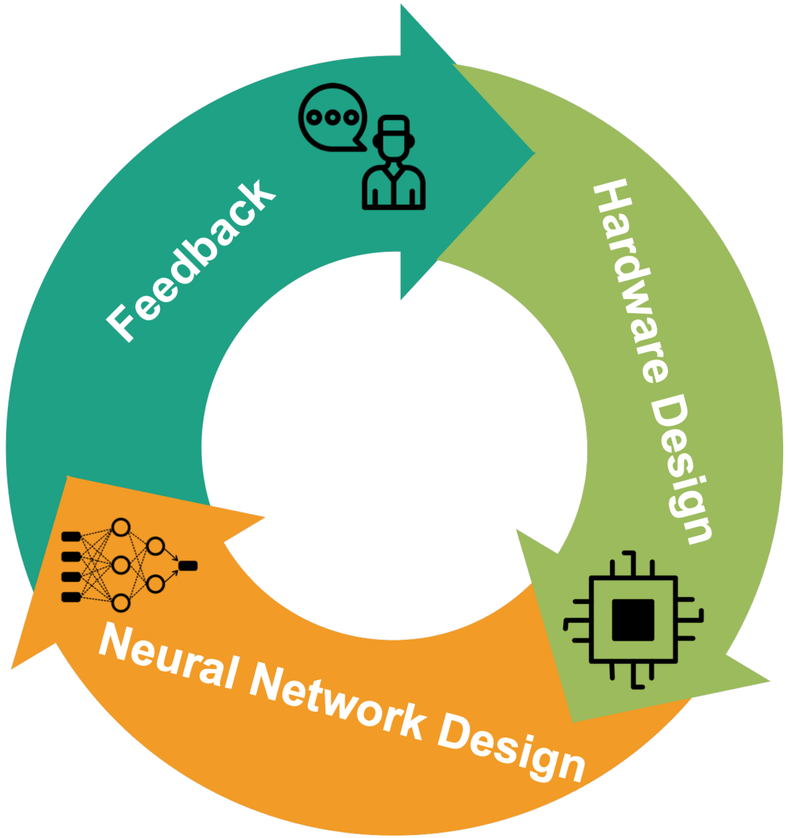
NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]
In-Datacenter Performance Analysis of a Tensor Processing Unit [Jouppi *et al.*, ISCA 2017]

Data Driven Approach is Desirable

- Given the huge design space, data-driven approach is desirable, where new architecture design evolves as new designs and rewards are collected
- Hardware-aware Neural Architecture Search (NAS) and auto compiler optimization (e.g., autoTVM)
  - Only focus on off-the-shelf hardware
  - Neglect the freedom in the hardware design space

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Design Spaces

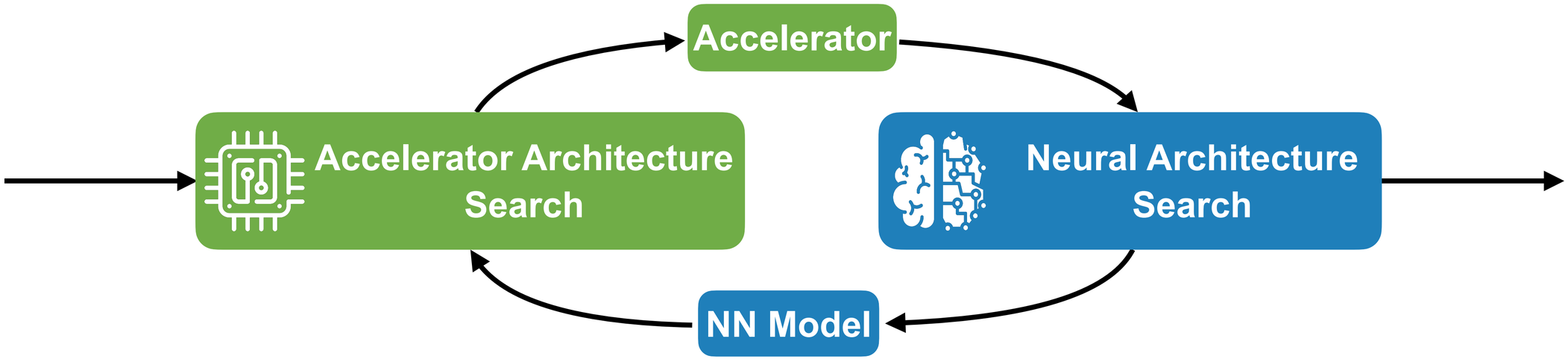


	Key Dimensions
Accelerator	Local Buffer Size, Global Buffer Size, #PEs
	Compute Array Size, PE Connectivity
Compiler	Loop Orders, Loop Tiling Size, Dataflow
Neural Network	#Layers, #Channels, Kernel Size, Bypass
	(Input / Weight) Quantization Precision

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Jointly Search Accelerator and Neural Network

- Searching accelerator and neural architecture *in one optimization loop* offers highly matched solutions



NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Architecture Design Space

	Key Dimensions	
Accelerator	**Local Buffer Size, Global Buffer Size, #PEs**	**Architectural Sizing**
	Compute Array Size, PE Connectivity	**Connectivity Parameters**
Compiler	Loop Orders, Loop Tiling Size, Dataflow	
Neural Network	#Layers, #Channels, Kernel Size, Bypass	
	(Input / Weight) Quantization Precision	

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Convolution Loop Nests

- Convolution loop nests can be divided into two parts: *temporal mapping* and *spatial parallelism*

Tensor Dimension	Notation
Batch	N
Output Channel	K
Input Channel	C
Input (Output) Row	Y (Y')
Input (Output) Column	X (X')
Kernel Row	R
Kernel Column	S

```
For _R in range(R / R):
 For _S in range(S / S):
  For _C in range(C / T_C):
   For _Y' in range(Y' / T_Y'):
    For _X' in range(X' / T_X'):
     For r in range(R):
      For s in range(S):
       For _k in range(K / 16):
        For _y' in range(T_Y'):
         For _x' in range(T_X'):
          For _c in range(T_C / 16):
           Parallel-For _m in range(16):
           Parallel-For _n in range(16):
            c = _C * T_C + _c * 16;
            k = _k * 16;
            y' = _Y' * T_Y' + _y';
            x' = _X' * T_X' + _x';
            y = y' + r - R;
            x = x' + s - S;
            psum[b,k,y',x'] += acts[b,x,y,x]
                               * wgts[k,c,y,x];
```

Mapping

HW

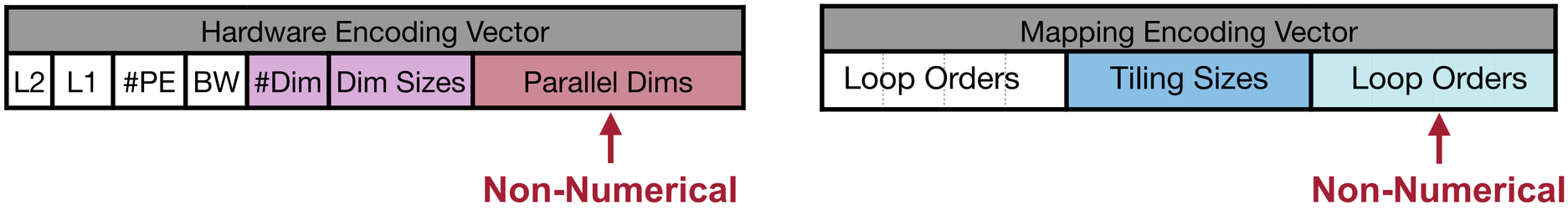
Loop Tiling

Loop Order

Hardware Parallelism

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Encoding Non-numerical Parameters

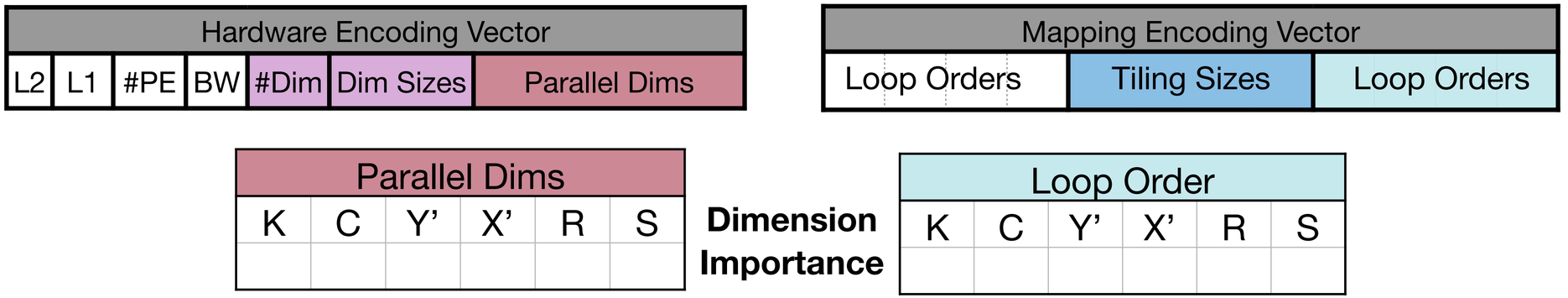


- Index-based Encoding
  - Increment/Decrement of index value does not convey any physical information

Non-Numerical Parameter Loop Orders	Numerical Encoding Value Index
CRXKYS	0
CXYRSK	1
…	…

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

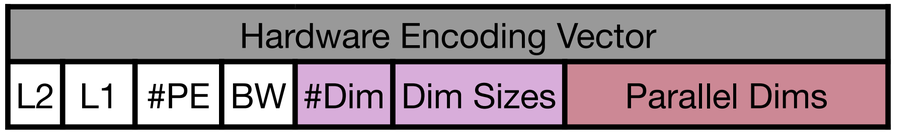
Encoding Non-numerical Parameters

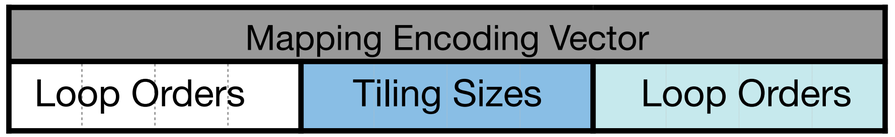


- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Encoding Non-numerical Parameters





Parallel Dims					
K	C	Y'	X'	R	S
4	6	2	2	3	1

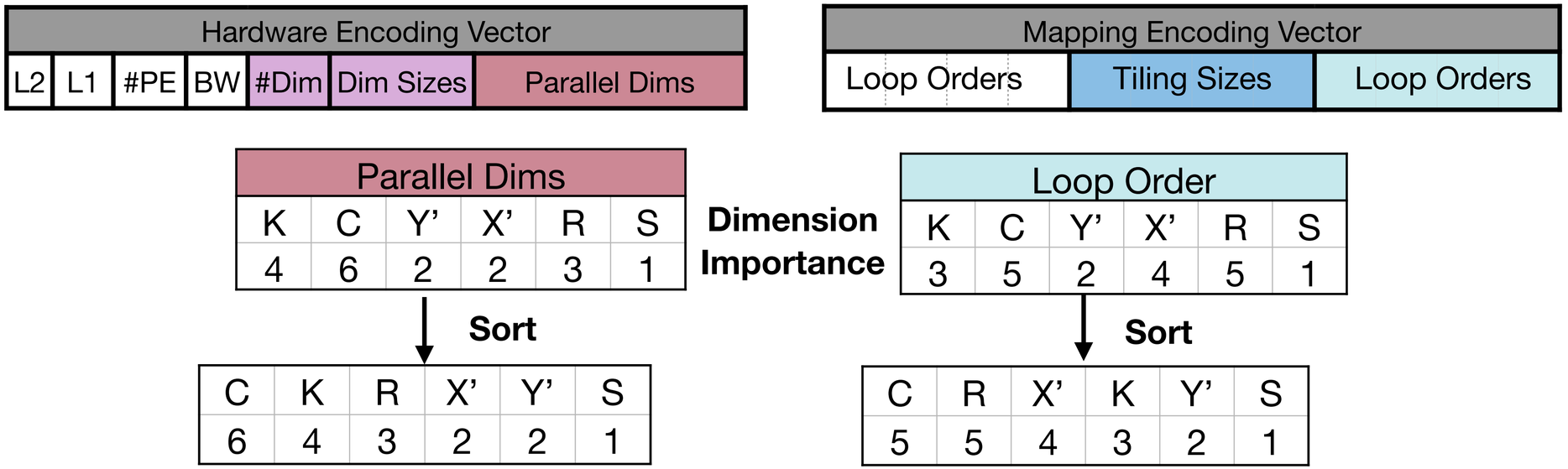
Dimension Importance

Loop Order					
K	C	Y'	X'	R	S
3	5	2	4	5	1

- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
    - by sampling based on multivariate normal distribution

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

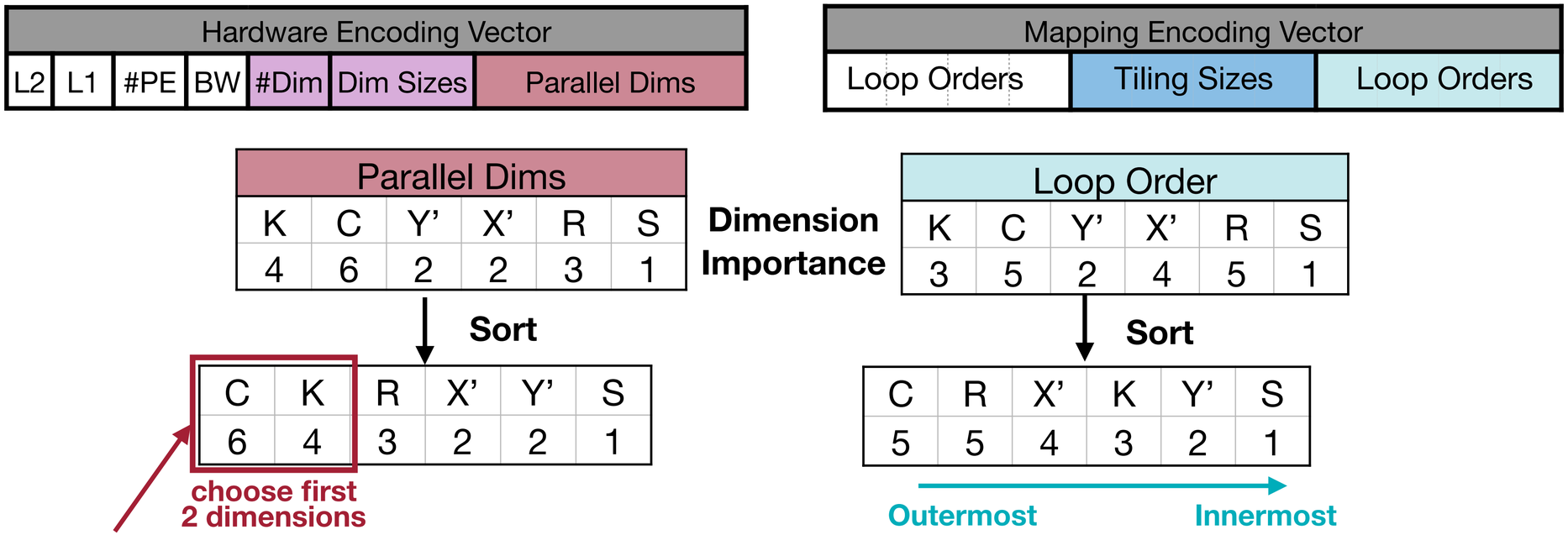
Encoding Non-numerical Parameters



- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
  - Step 3: Sort the dimensions by the importance value in decreasing order

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

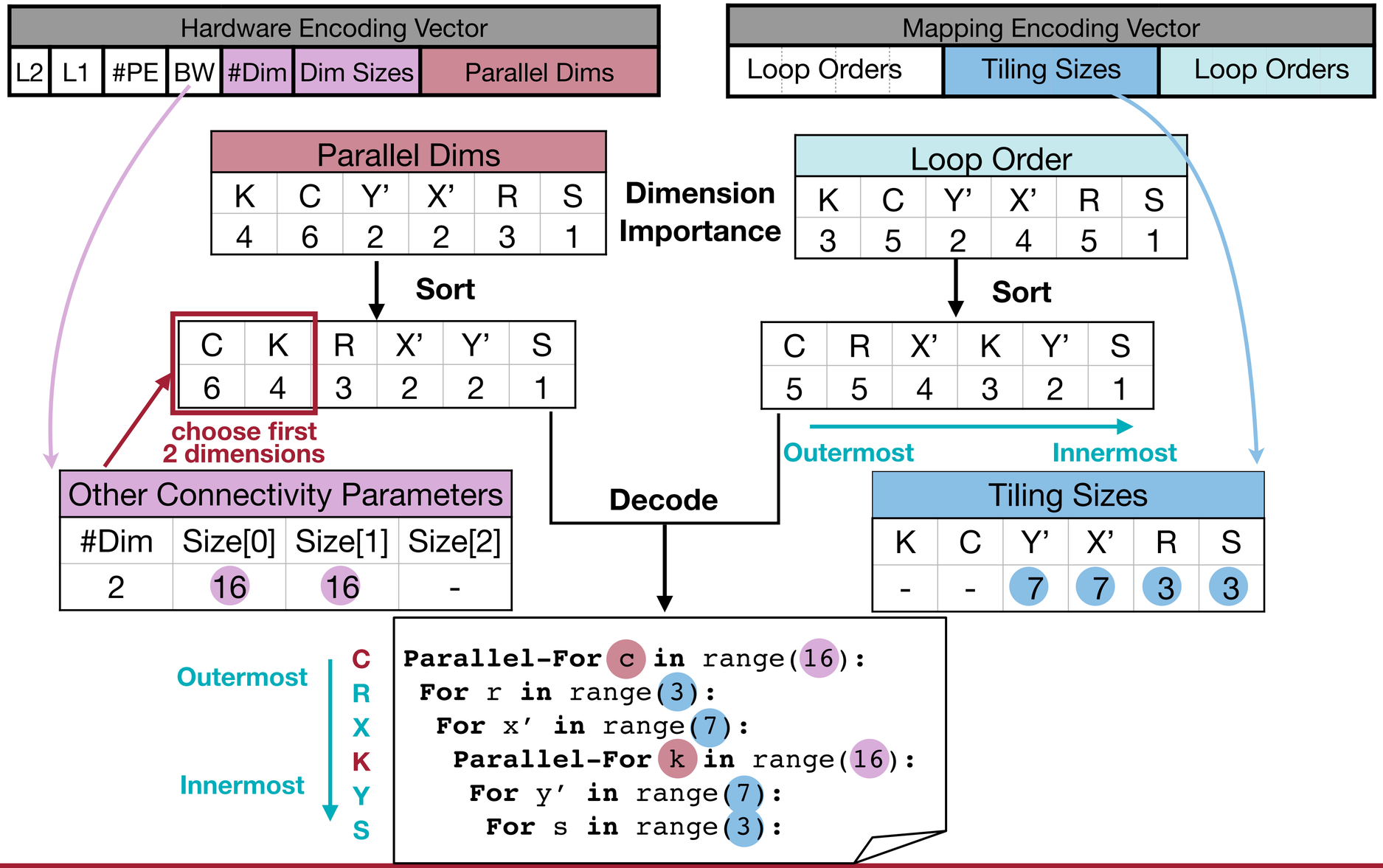
Encoding Non-numerical Parameters



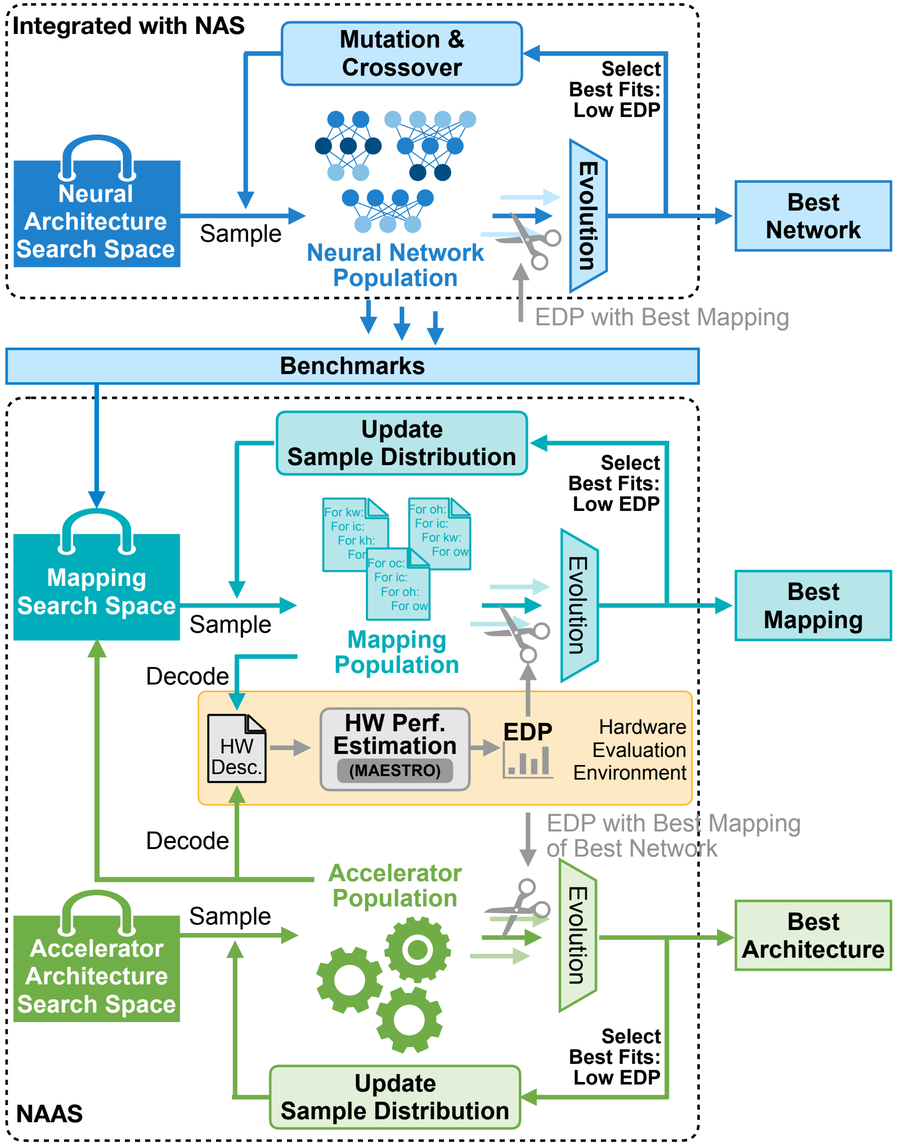
- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
  - Step 3: Sort the dimensions by the importance value in decreasing order

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

Encoding Non-numerical Parameters



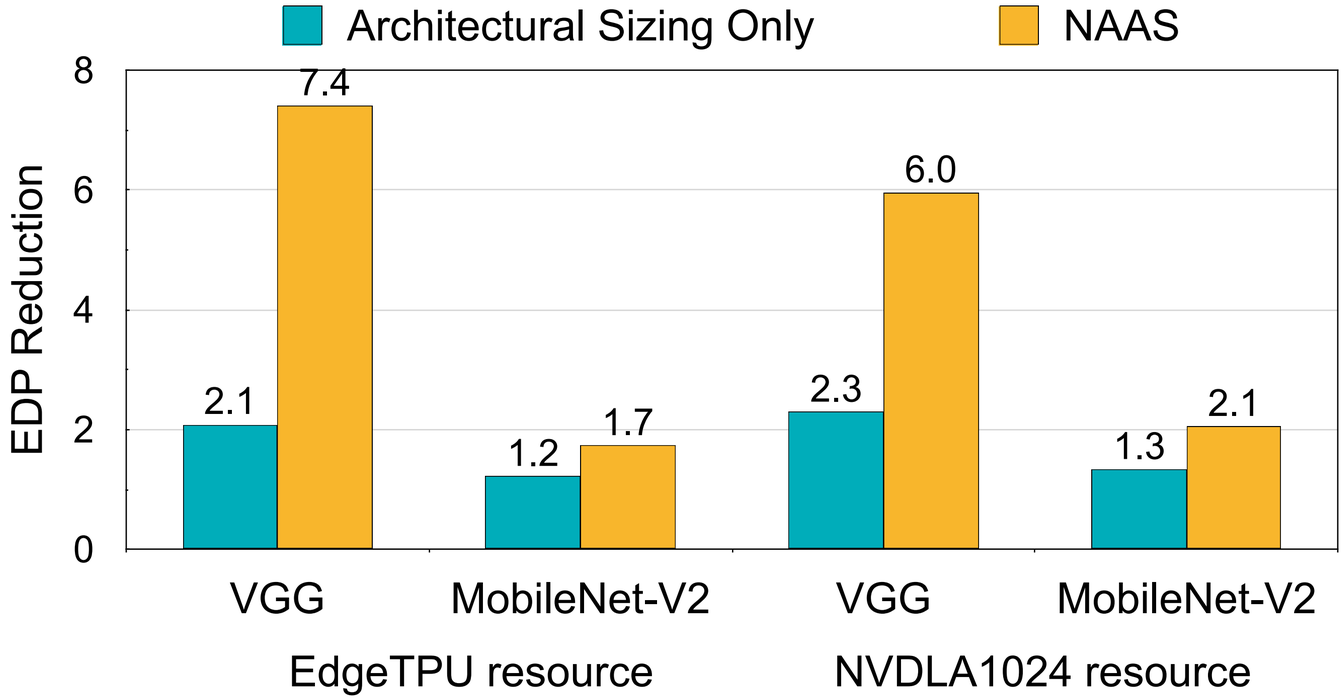
Jointly Optimize NN, Mapping, Accelerator




```
For epoch_naas in range(max_naas_epochs):
  accelerators = NAAS_generate_hardware()
  For hw in accelerators:
    For epoch_ofa in range(max_ofa_epochs):
      networks = OFA_generate_networks(accuracy)
      For nn in networks:
        map = NAAS_optimize_mappings(hw, nn)
        edp = NAAS_get_edp(hw, nn, map)
        OFA_update_optimizer(nn, edp)
        best_nn, best_map, best_edp = OFA_update_best(nn, map, edp)
    NAAS_update_optimizer(hw, best_nn, best_map, best_edp)
```

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

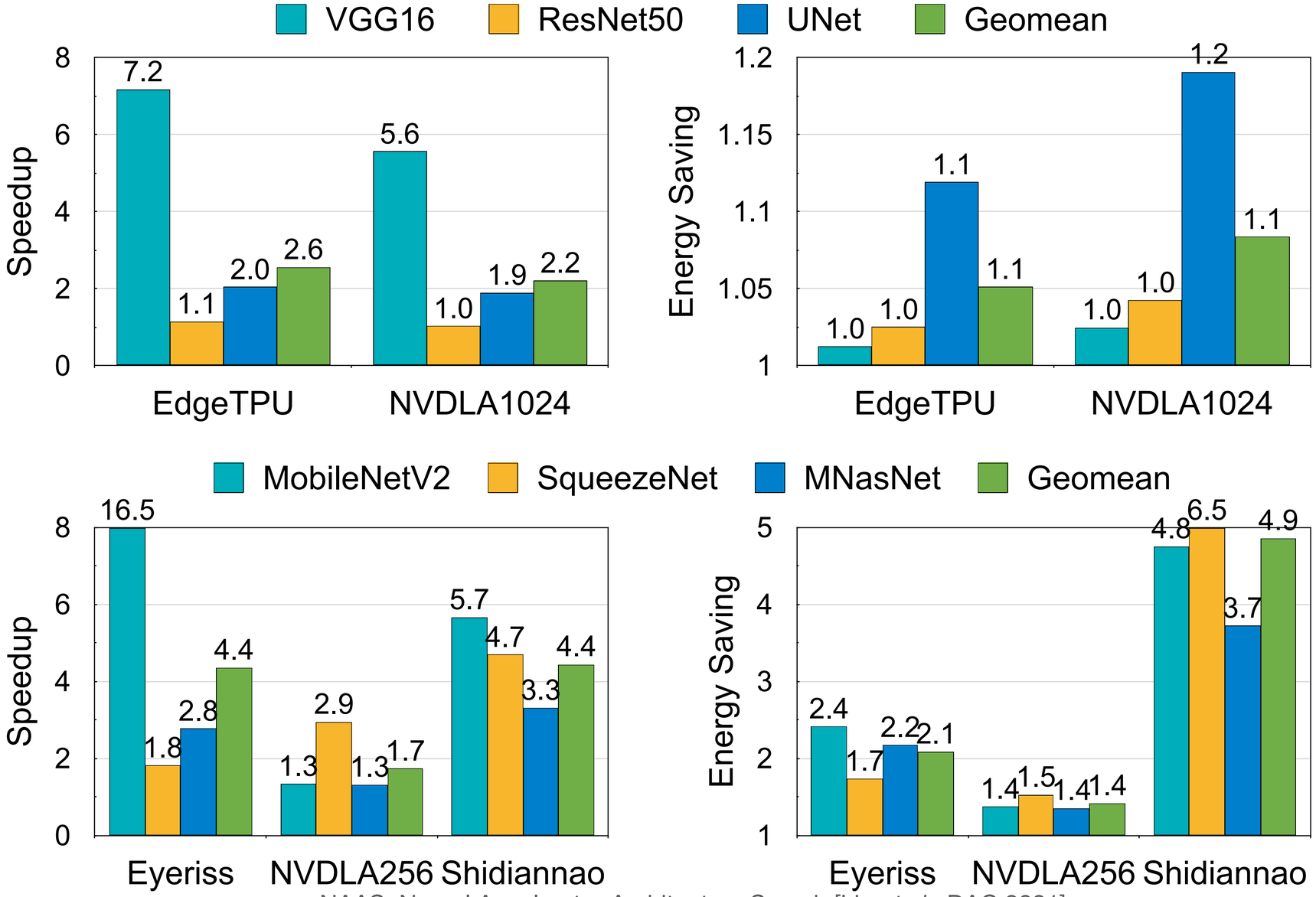
Search Beyond Architecture Sizing



- Compared to searching the architectural sizing only, searching the connectivity parameters and mapping strategies as well achieves considerable EDP reduction.

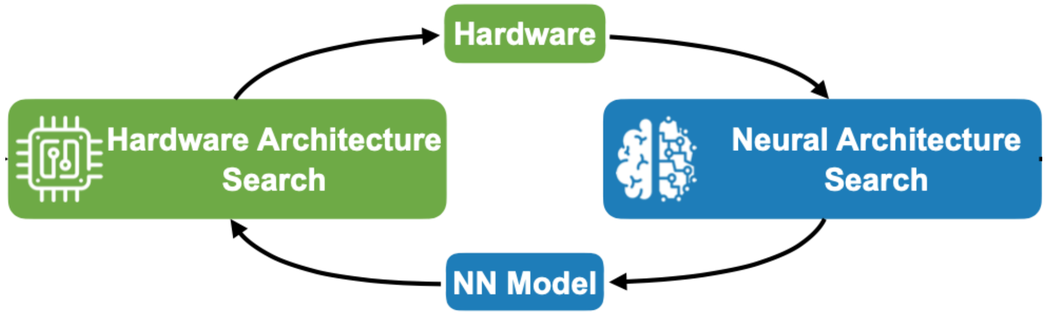
NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

NAAS offers better solution than baseline

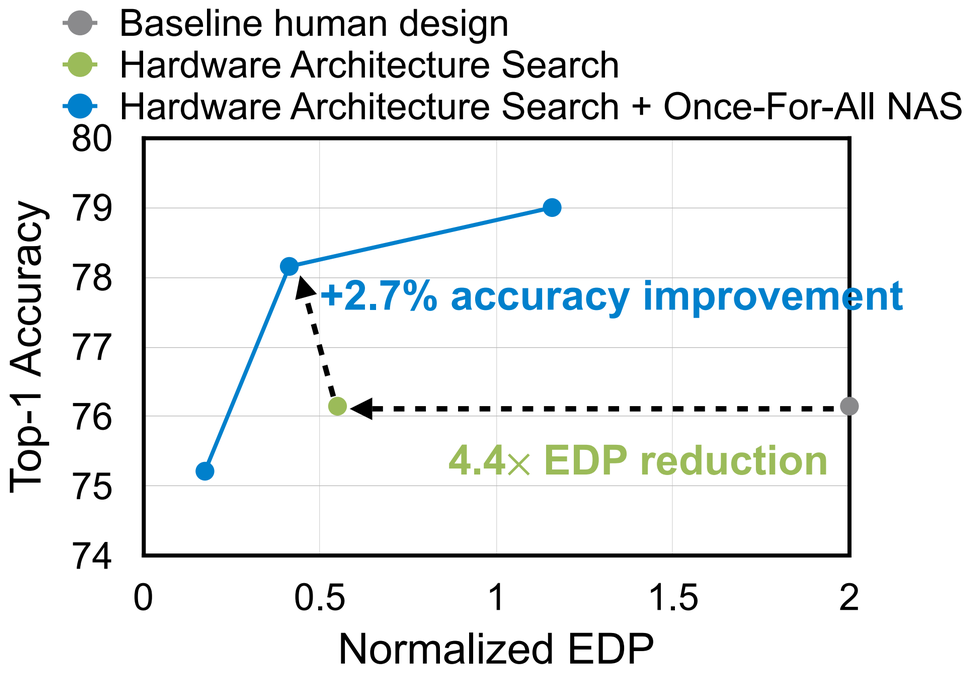


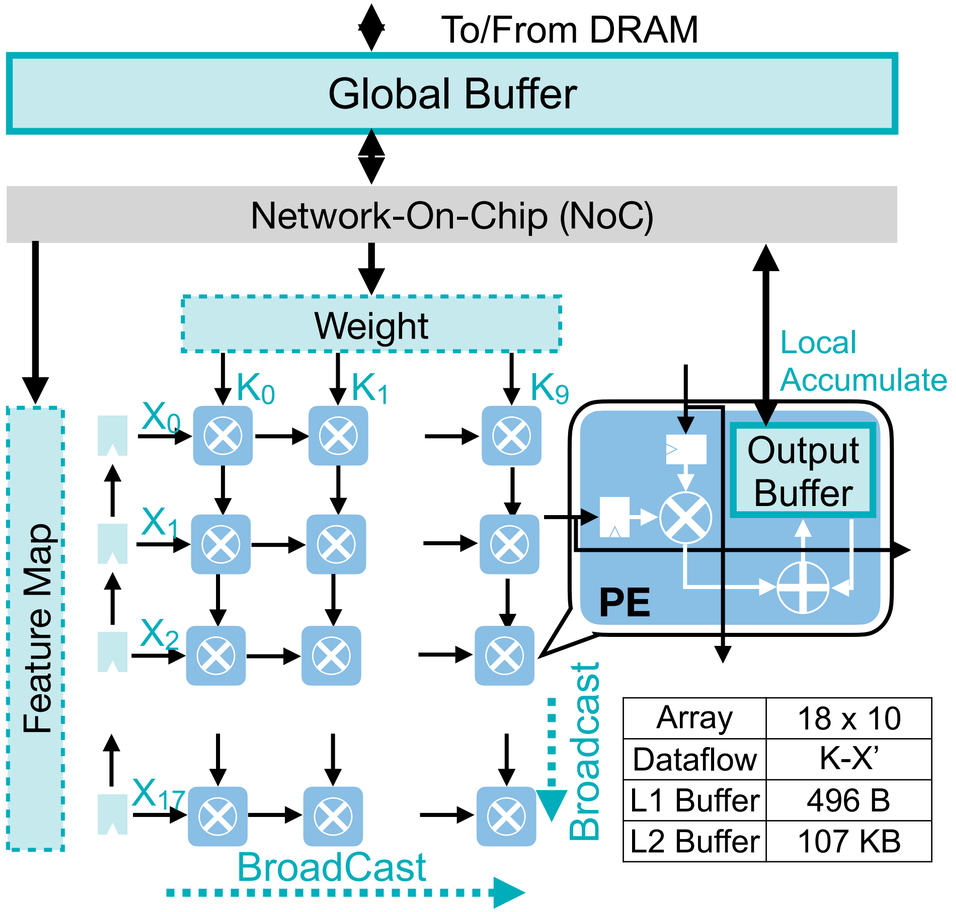
NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

NAAS outperforms human design



Neural-Hardware Architecture Search in one loop



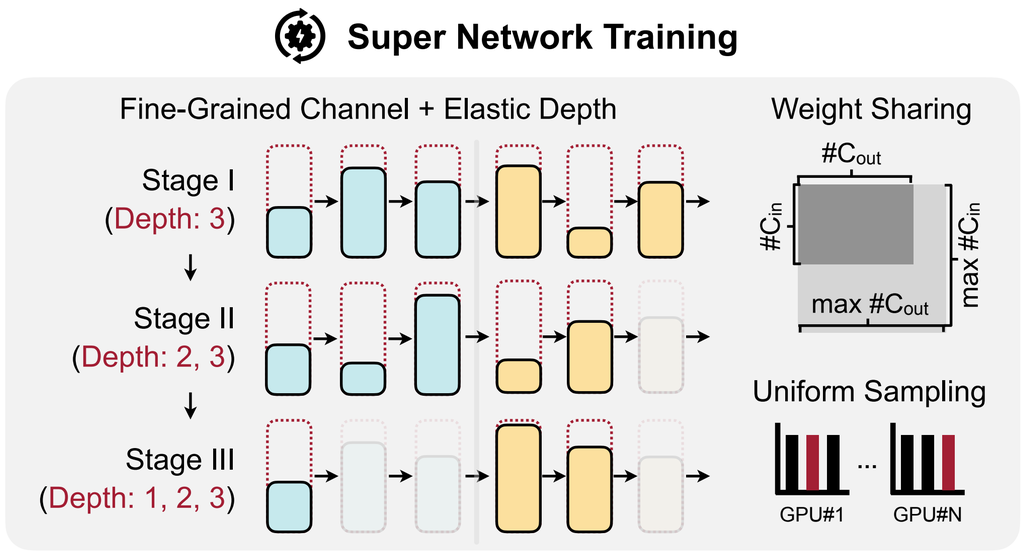


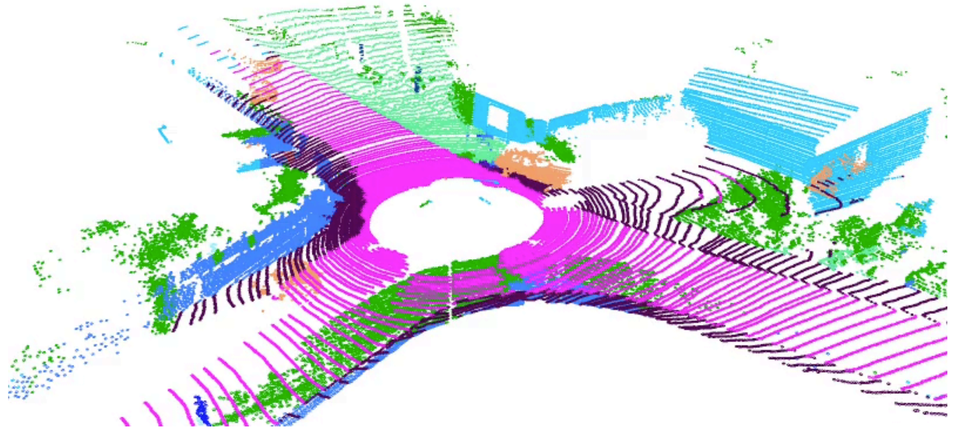
NAAS designed data flow parallelizes the output height and output channel dimension, which is very different from the human design

NAAS: Neural Accelerator Architecture Search [Lin *et al.*, DAC 2021]

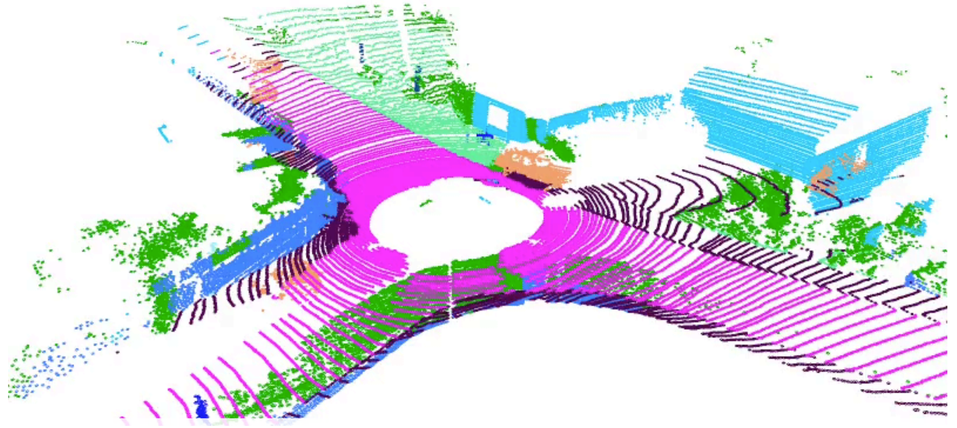
Applications of NAS

NAS for Point Cloud Understanding





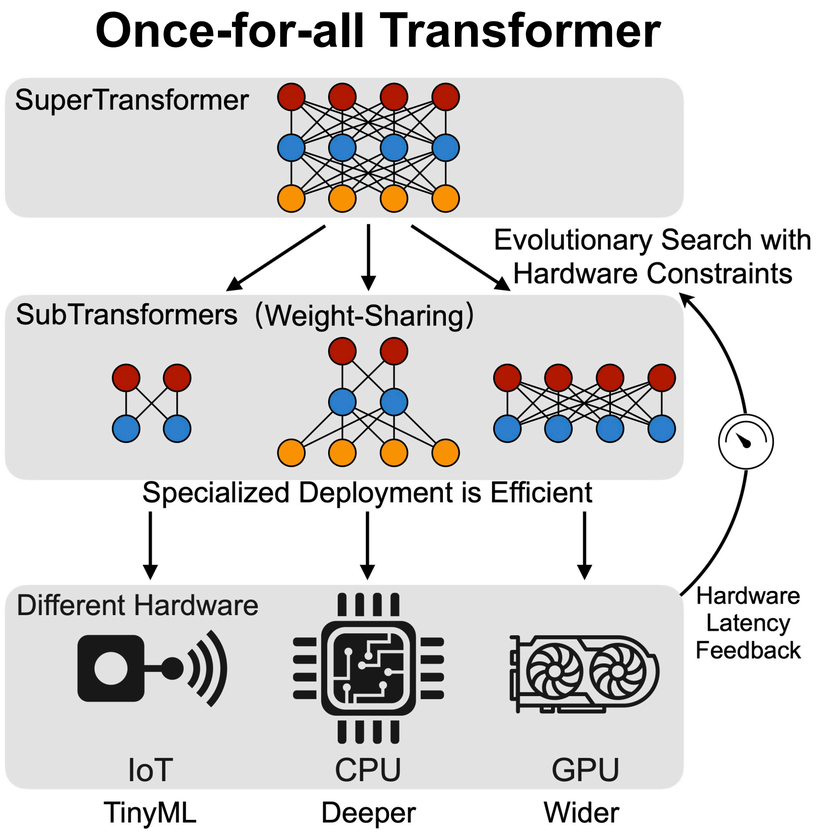
MinkowskiNet: **3.4** FPS

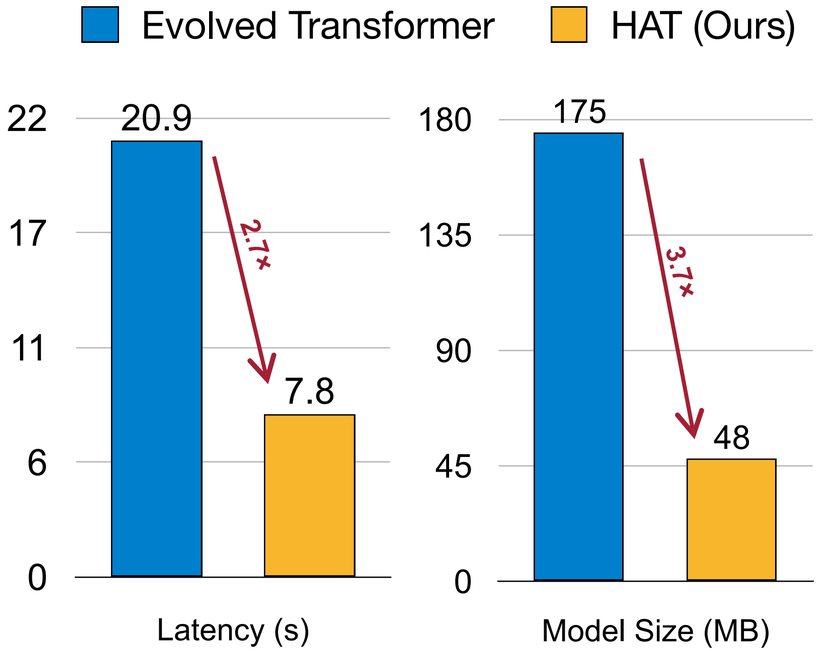


SPVNAS (Ours): **9.1** FPS

Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang *et al.*, ECCV 2020]

NAS for Transformers and NLP



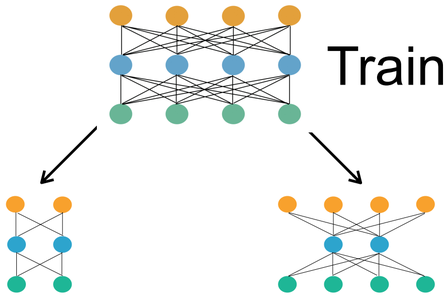


- For WMT'14 En-Fr running on Raspberry Pi, HAT is 2.7x faster, 3.7x smaller model size, 3.2x fewer FLOPs, and 10,148x less search cost, even achieves 0.1 higher BLEU score.

HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]

NAS for Anycost GAN

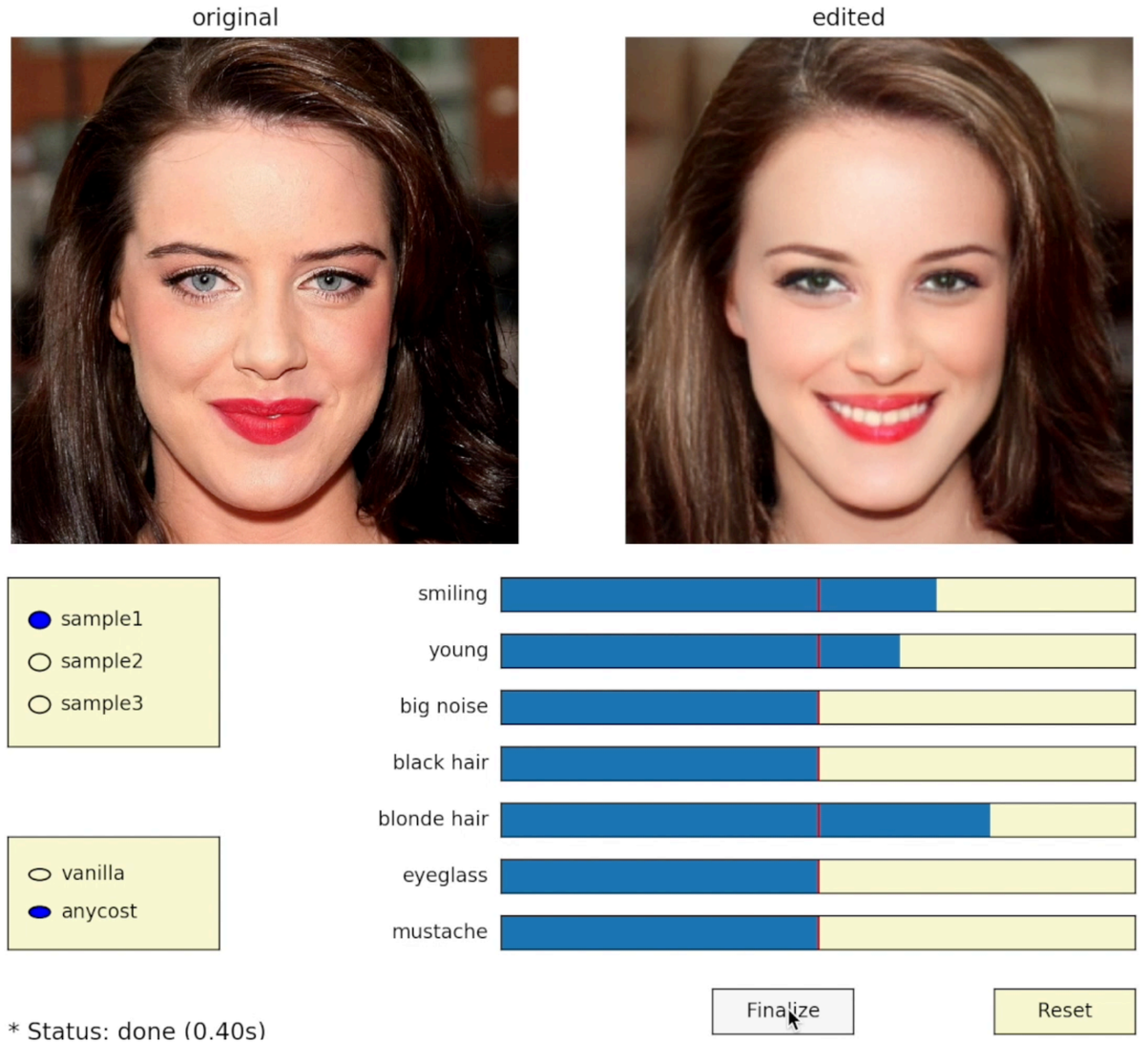
- Generative Adversarial Network (GAN) is computationally heavy and slow
- Difficult for interactive photo editing on mobile device (iPad)
- Anycost GAN with once-for-all network:



Train once

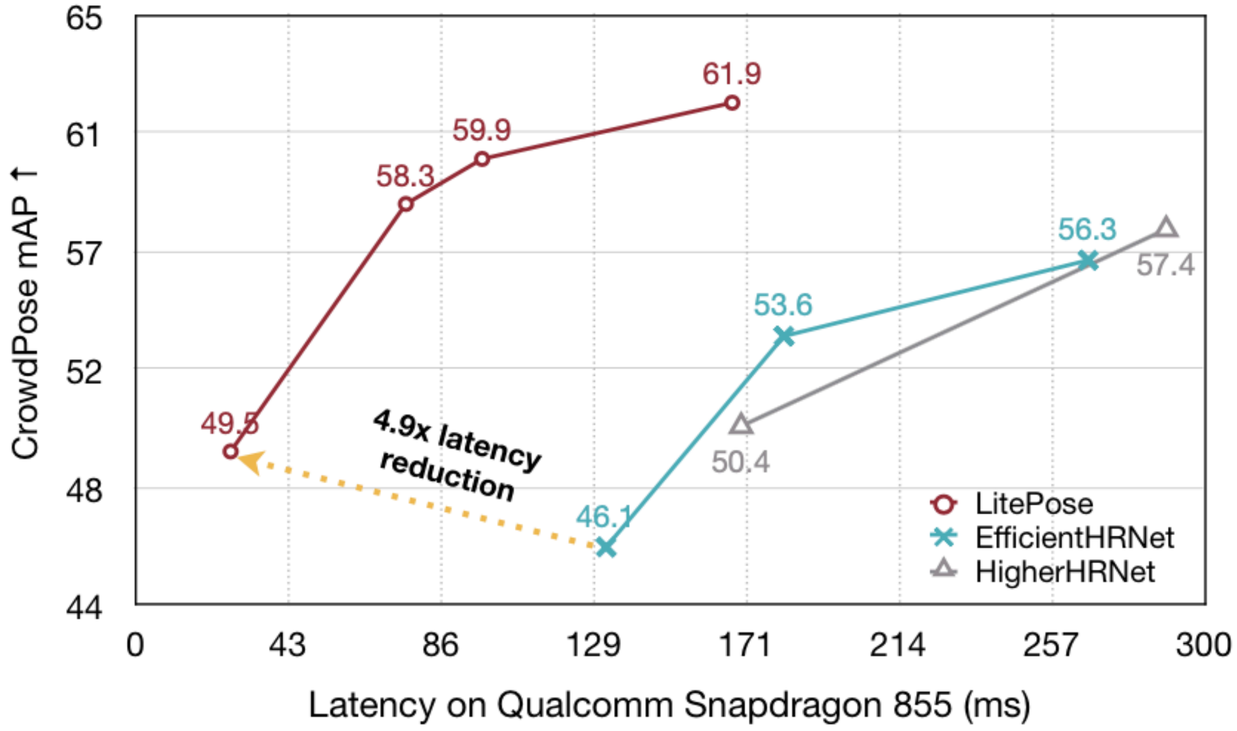
Small sub-net:
low cost,
fast prototyping

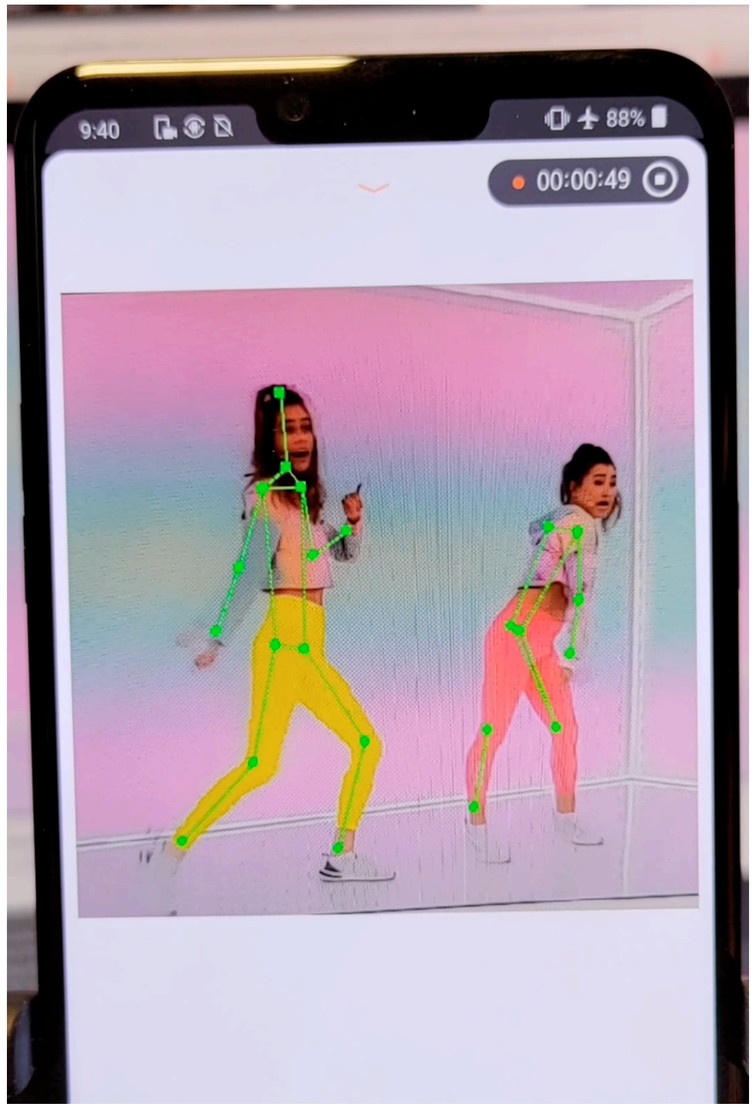
Large sub-net:
high-quality
finalization



Anycost GANs for Interactive Image Synthesis and Editing [Lin *et al.*, CVPR 2021]

NAS for Pose Estimation

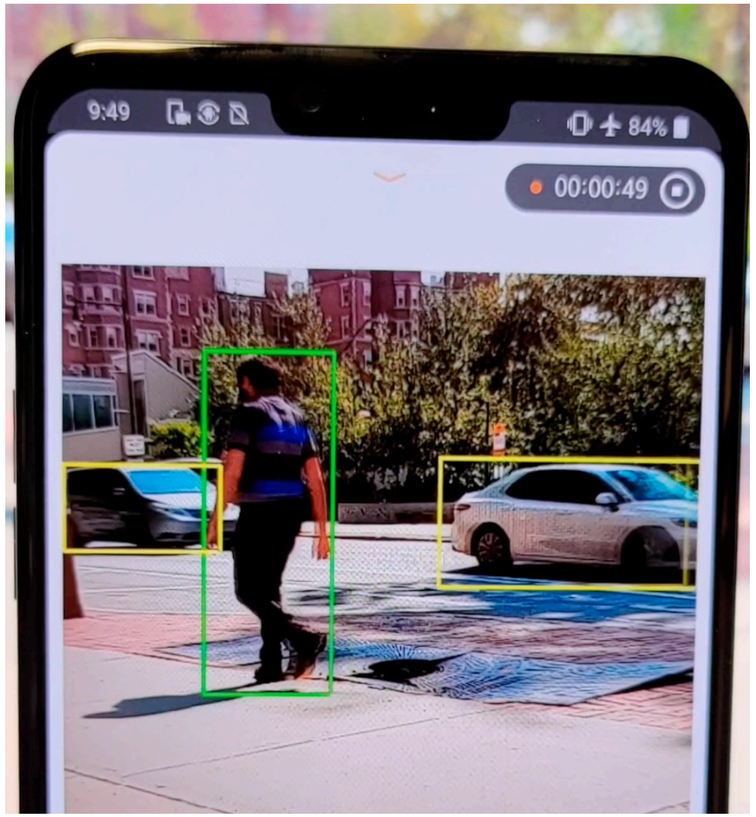




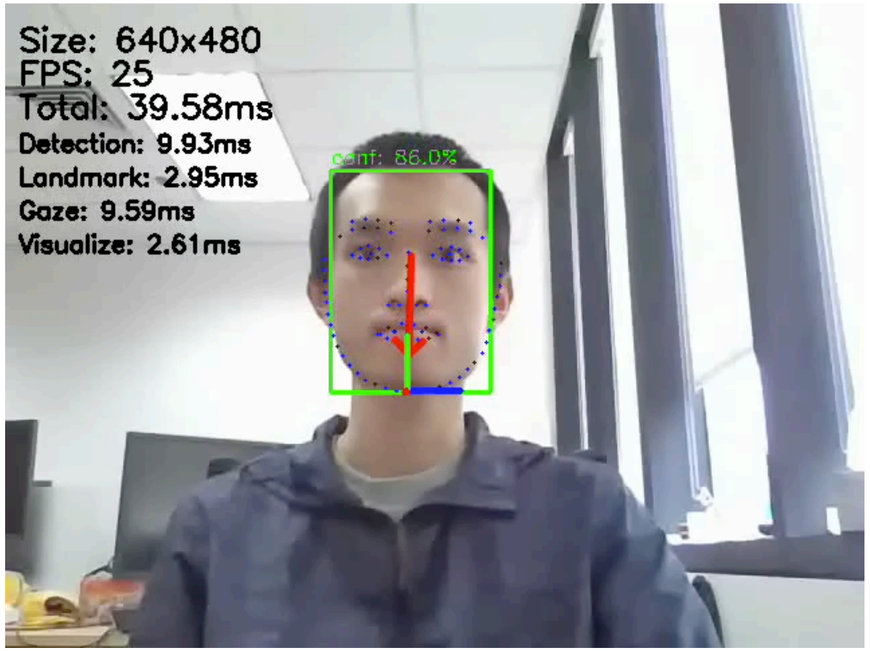
On-device pose estimation by:
Designing new design space + HW-aware NAS

Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation [Wang *et al.*, CVPR 2022]

NAS Designs Light-weight Model, Bring AI to Mobile Devices



on-device car/person detection

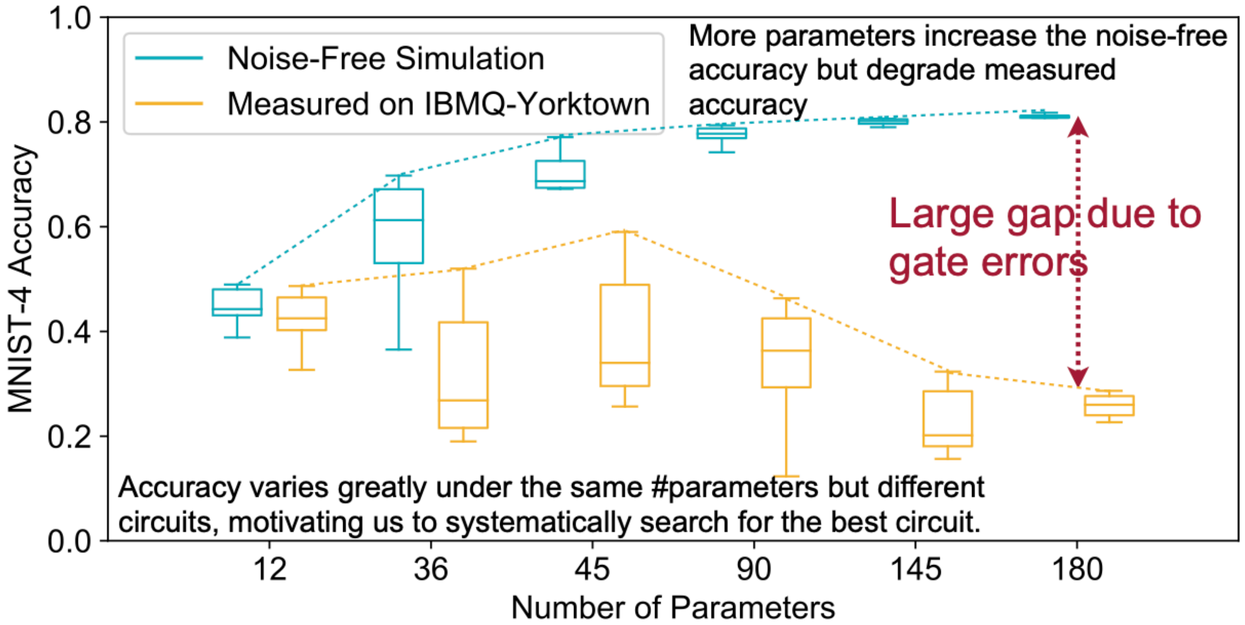


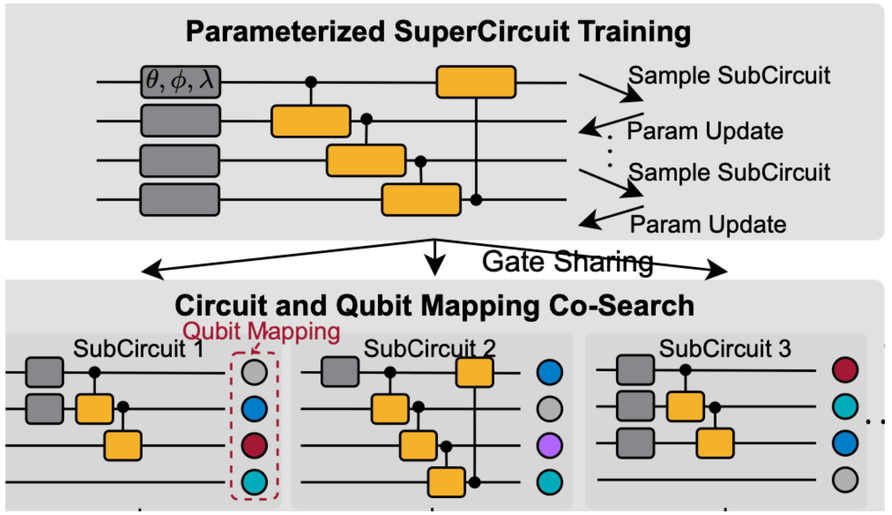
on-device gaze estimation
(Rasp Pi)

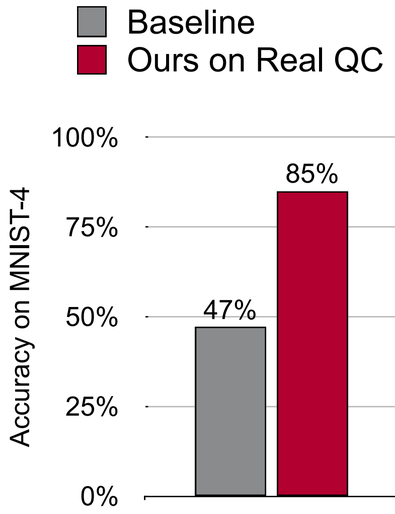
on-device segmentation

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]

NAS for Quantum AI





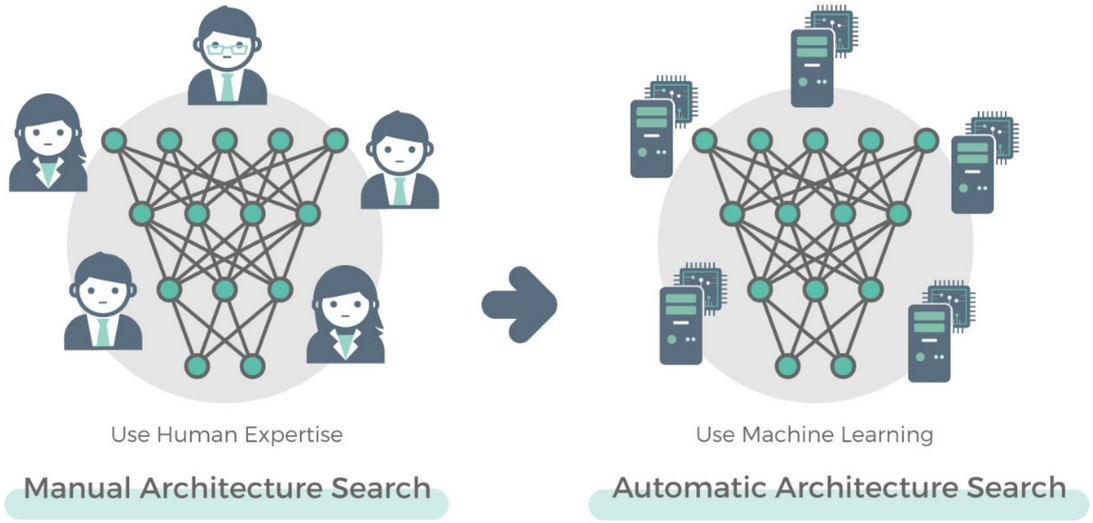


- Quantum noise is the bottleneck for quantum neural nets, accuracy: 87% -> 47%
- **QuantumNAS: N**oise-**A**ware **S**earch for robust circuit architecture by training a "super circuit", then search a "sub-circuit" that is robust to noise; **prune** small-magnitude quantum gates
- **TorchQuantum:** open-source library used by UT Austin, Cornell, Notre Dame, USC, U Chicago, Tsinghua U

QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits [Wang et al., HPCA 2022]
QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization [Wang et al., DAC 2022]
QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning [Wang et al., DAC 2022]
qmlsys.mit.edu

Summary of Today's Lecture

- In this lecture, we introduce:
  - Performance estimation in NAS
  - Hardware-Aware NAS
  - Applications of NAS
- In the next lecture, Dr. Lucas Liebenwein from OmniML will give a guest talk.



References

1. Neural Architecture Search with Reinforcement Learning [Zoph *et al.*, ICLR 2017]
2. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]
3. Net2Net: Accelerating Learning via Knowledge Transfer [Chen et al., ICLR 2016]
4. Efficient Architecture Search by Network Transformation [Cai et al., AAAI 2018]
5. Simple And Efficient Architecture Search for Convolutional Neural Networks [Elsken et al., arXiv 2017]
6. Path-Level Network Transformation for Efficient Architecture Search [Cai et al., ICML 2018]
7. SMASH: One-Shot Model Architecture Search through HyperNetworks [Brock et al., ICLR 2018]
8. Efficient Neural Architecture Search via Parameter Sharing [Pham et al., ICML 2018]
9. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]
10. Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tan et al., CVPR 2019]
11. Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]
12. SNIP: Single-Shot Network Pruning based on Connection Sensitivity [Lee et al., ICLR 2019]

References

- 13. Picking Winning Tickets before Training by Preserving Gradient Flow [Wang et al., ICLR 2020]
- 14. Pruning Neural Networks without Any Data by Iteratively Conserving Synaptic Flow [Tanaka et al., NeurIPS 2020]
- 15. GradSign: Model Performance Inference with Theoretical Insights [Zhang and Jia, ICLR 2022]
- 16. Zen-NAS: A Zero-Shot NAS for High-Performance Image Recognition [Lin et al., ICCV 2021]
- 17. MAE-DET: Maximum Entropy Principle for Efficient Object Detection [Sun et al., ICML 2022]
- 18. NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]
- 19. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang et al., ECCV 2020]
- 20. Anycost GANs for Interactive Image Synthesis and Editing [Lin et al., CVPR 2021]
- 21. Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation [Wang et al., CVPR 2022]
- 22. QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits [Wang et al., HPCA 2022]

References

- 23. QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization [Wang et al., DAC 2022]
- 24. QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning [Wang et al., DAC 2022]