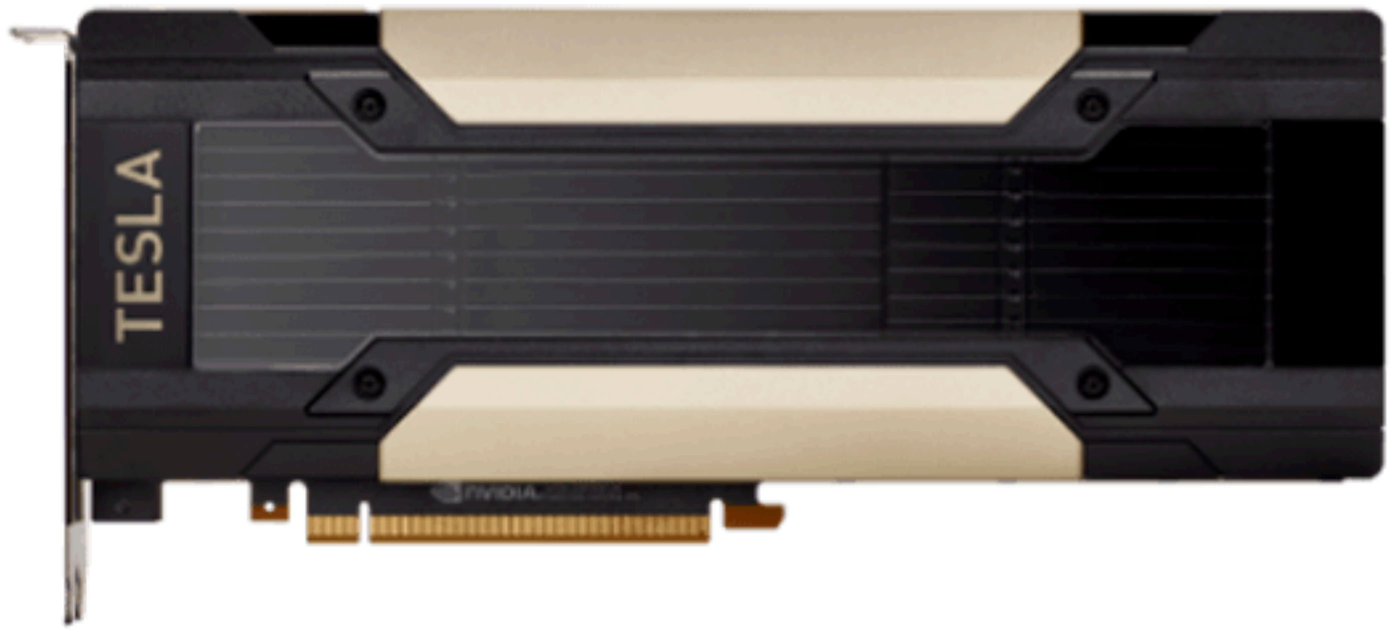
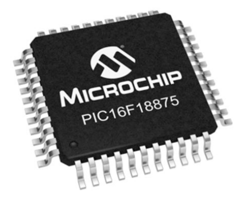
Efficient Neural Architecture Search and Tiny Transfer Learning

Han Cai

Challenge: Efficient Inference on Diverse Hardware Platforms

Cloud AI		Mobile AI		Tiny AI (AIoT)
	less resource →		less resource →	

Cloud AI
- Memory: 32GB
- Computation: 10^{12} FLOPS

Mobile AI
- Memory: 4GB
- Computation: 10^{9} FLOPS

Tiny AI (AIoT)
- **Memory: 100 KB**
- **Computation: <10^{6} FLOPS**

- Different hardware platforms have different resource constraints. We need to **customize** our models **for each platform** to achieve the best accuracy-efficiency trade-off, **especially on** resource-constrained **edge devices**.

Challenge: Efficient Inference on Diverse Hardware Platforms

2019

For training iterations:
forward-backward();

The design cost is calculated under the assumption of using MobileNet-v2.

Challenge: Efficient Inference on Diverse Hardware Platforms

2019

For search episodes: // meta controller
  For training iterations:
    forward-backward(); **Expensive**
  If good_model: **break;**
For post-search training iterations:
  forward-backward(); **Expensive**

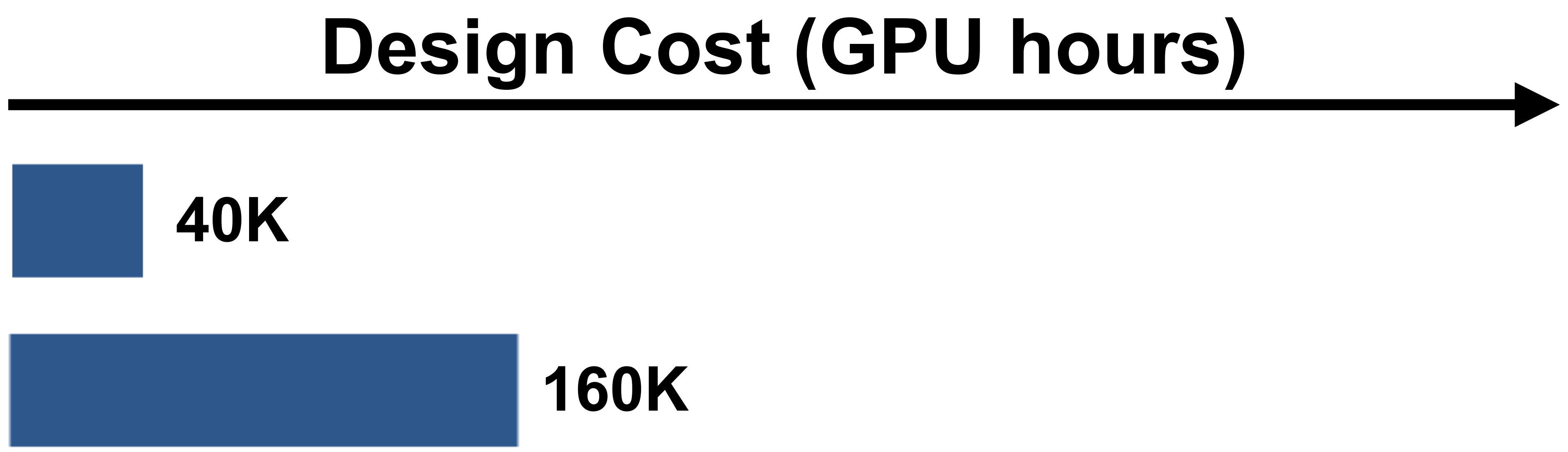
The design cost is calculated under the assumption of using MnasNet.

[1] Tan, Mingxing, et al. "Mnasnet: Platform-aware neural architecture search for mobile." *CVPR*. 2019.

Challenge: Efficient Inference on Diverse Hardware Platforms

Diverse Hardware Platforms

For devices:
  For search episodes: // meta controller
    For training iterations:
      forward-backward(); **Expensive!**
    If good_model: **break;**
  For post-search training iterations:
    forward-backward(); **Expensive!**

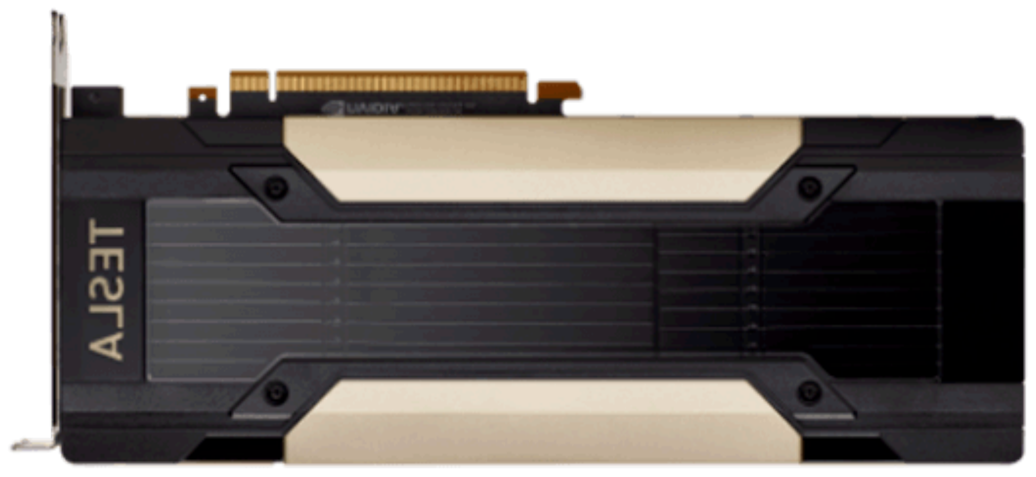


The design cost is calculated under the assumption of using MnasNet.

[1] Tan, Mingxing, et al. "Mnasnet: Platform-aware neural architecture search for mobile." *CVPR*. 2019.

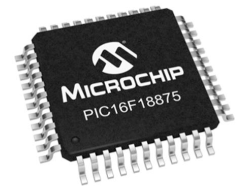
Challenge: Efficient Inference on Diverse Hardware Platforms

Diverse Hardware Platforms



Cloud AI (10^{12} FLOPS)

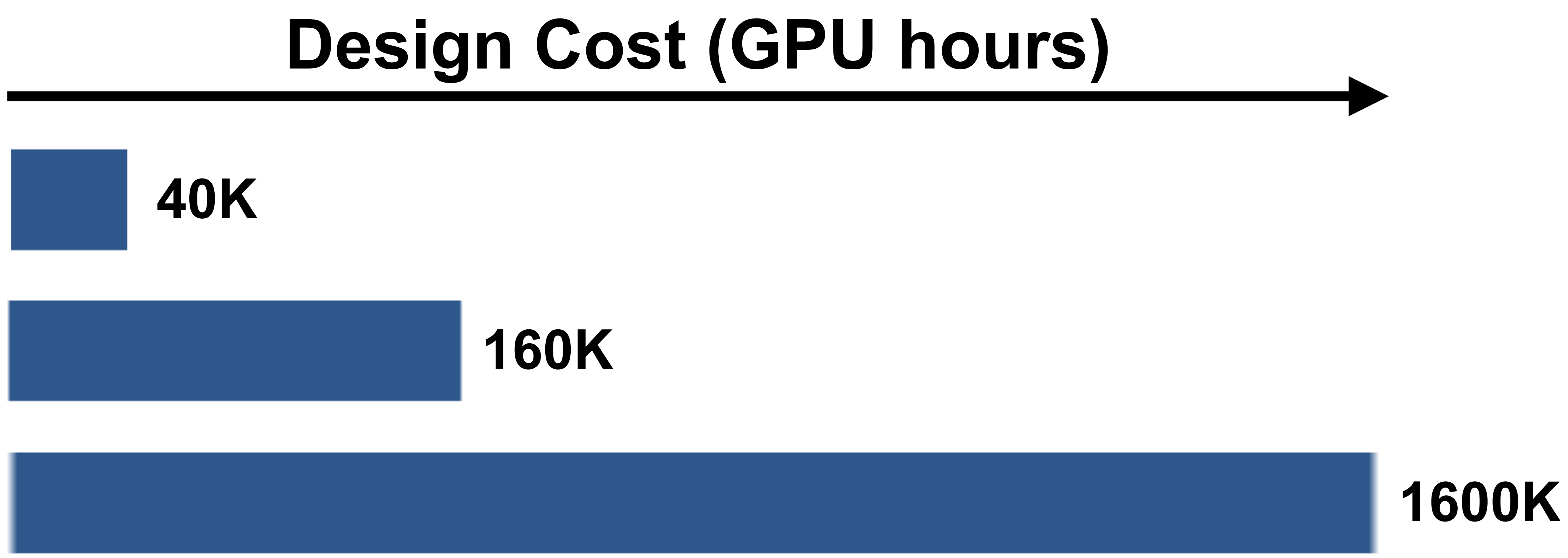
Mobile AI (10^{9} FLOPS)



Tiny AI (10^{6} FLOPS)

...

For many devices:
    For search episodes: // meta controller
        For training iterations:
            forward-backward(); **Expensive!!**
        If good_model: **break;**
    For post-search training iterations:
        forward-backward(); **Expensive!!**

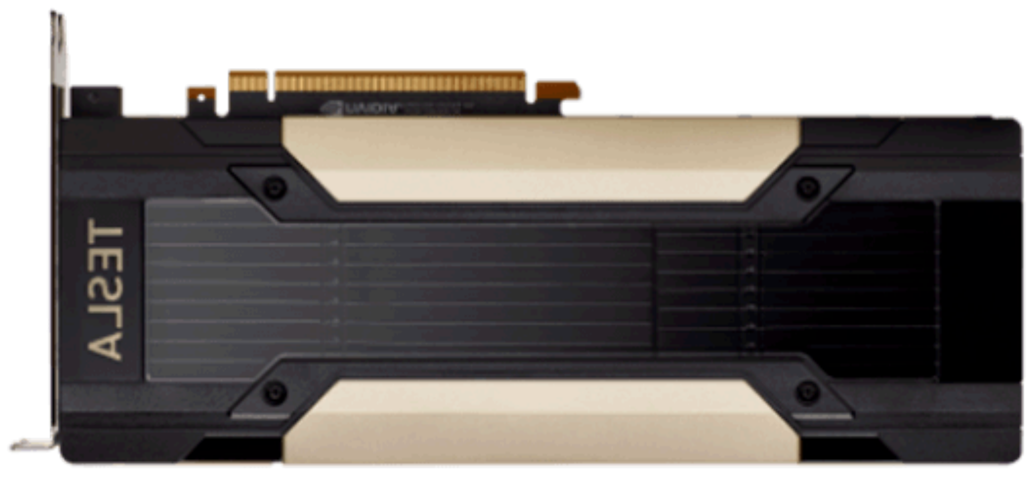


The design cost is calculated under the assumption of using MnasNet.

[1] Tan, Mingxing, et al. "Mnasnet: Platform-aware neural architecture search for mobile." *CVPR*. 2019.

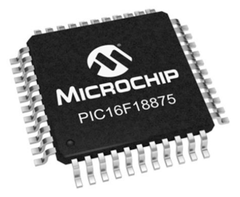
Challenge: Efficient Inference on Diverse Hardware Platforms

Diverse Hardware Platforms



Cloud AI (10^{12} FLOPS)

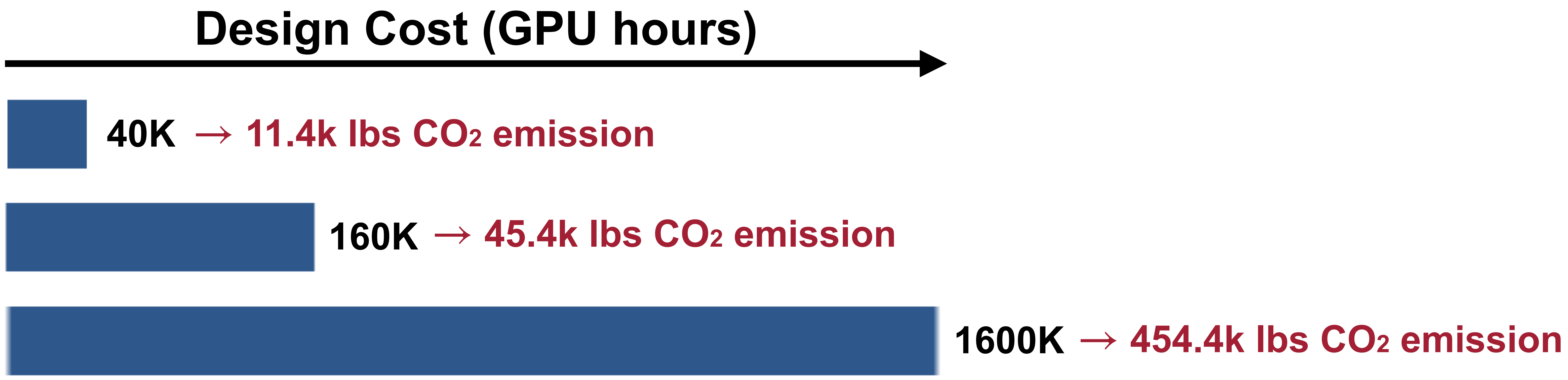
Mobile AI (10^{9} FLOPS)



Tiny AI (10^{6} FLOPS)

...

<u>**For** many devices:</u>
  <u>**For** search episodes:</u> // meta controller
    For training iterations:
      forward-backward(); **Expensive!!**
    If good_model: **break;**
  For post-search training iterations:
    forward-backward(); **Expensive!!**



1 GPU hour translates to 0.284 lbs CO_2 emission according to
Strubell, Emma, et al. "Energy and policy considerations for deep learning in NLP." *ACL.* 2019.

Problem:

TinyML comes at the cost of BigML

(inference) (training/search)

We need Green AI:

Solve the Environmental Problem of NAS

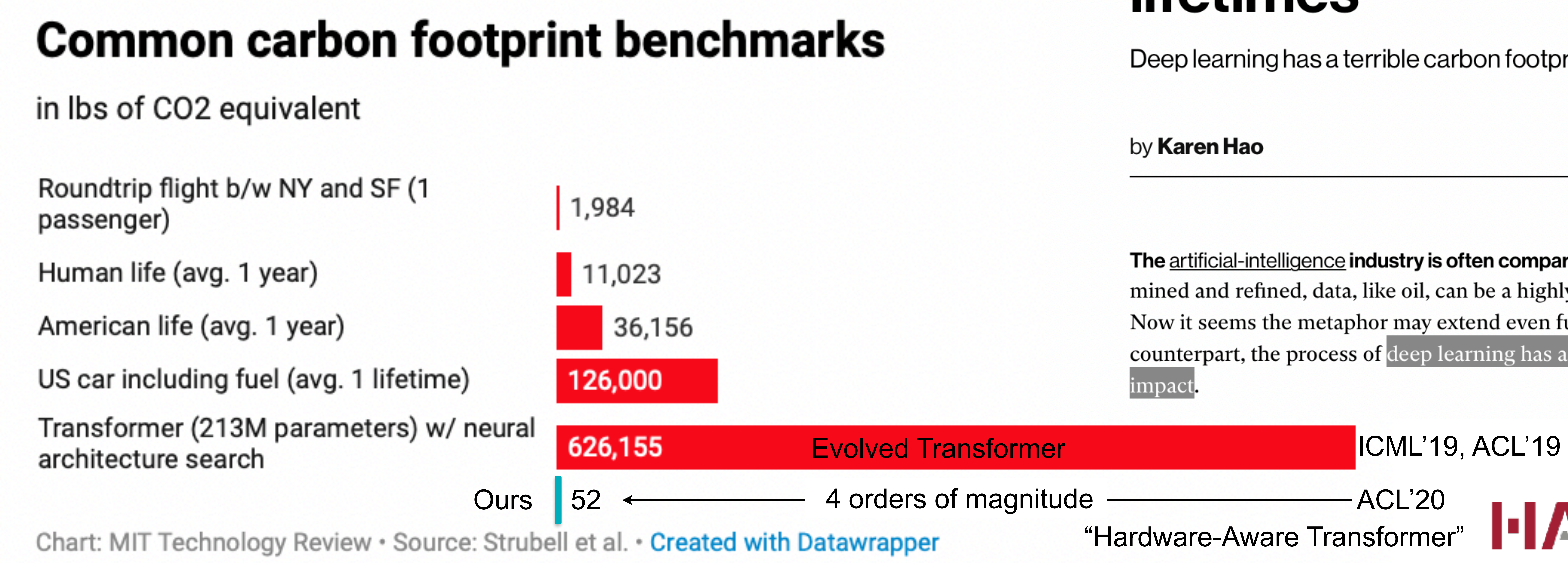
Artificial intelligence / Machine learning

Training a single AI model can emit as much carbon as five cars in their lifetimes

Deep learning has a terrible carbon footprint.

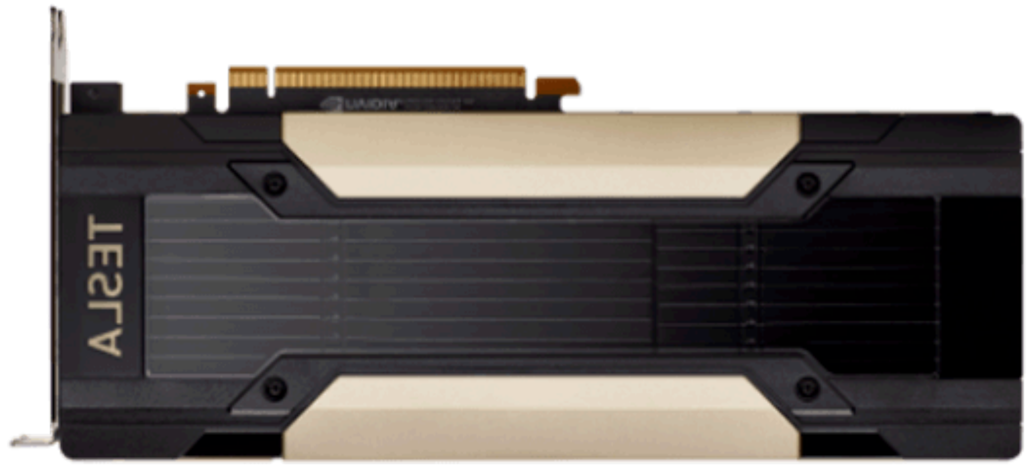
by **Karen Hao** June 6, 2019

The artificial-intelligence industry is often compared to the oil industry: once mined and refined, data, like oil, can be a highly lucrative commodity. Now it seems the metaphor may extend even further. Like its fossil-fuel counterpart, the process of deep learning has an outsize environmental impact.



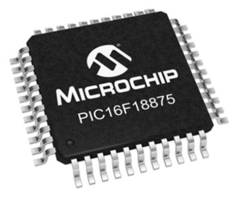
Challenge: Efficient Inference on Diverse Hardware Platforms

Diverse Hardware Platforms



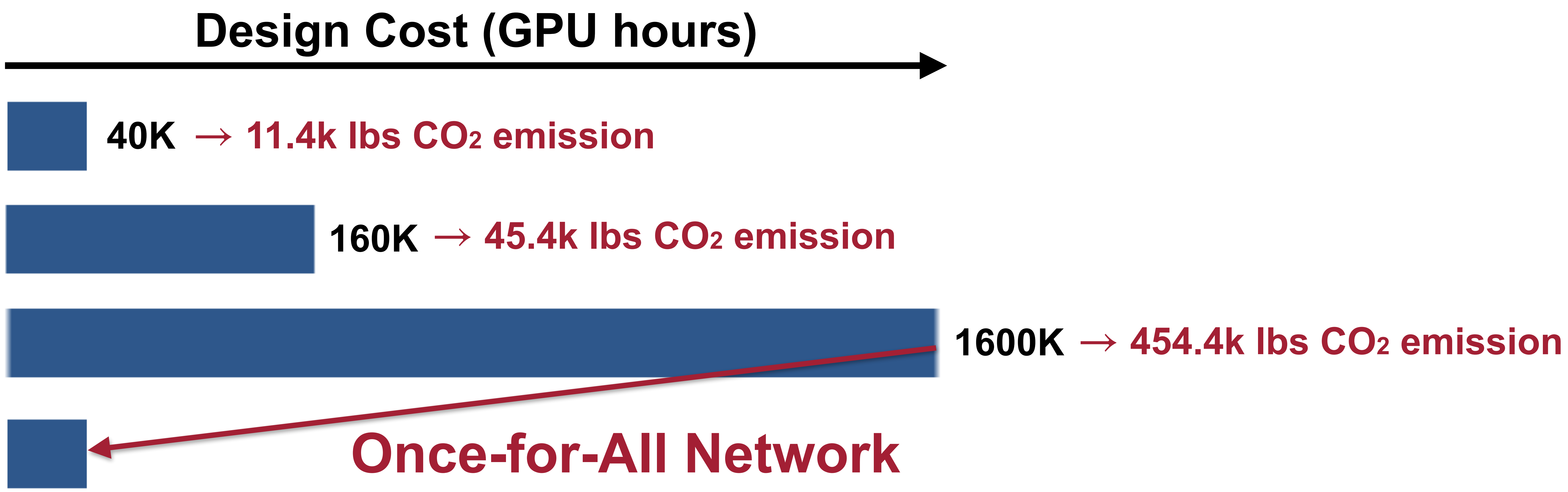
Cloud AI (10^{12} FLOPS)

Mobile AI (10^{9} FLOPS)



...

Tiny AI (10^{6} FLOPS)



1 GPU hour translates to 0.284 lbs CO_2 emission according to
Strubell, Emma, et al. "Energy and policy considerations for deep learning in NLP." *ACL.* 2019.

OFA: Decouple Training and Search

Conventional NAS with meta controller

For devices:
  For search episodes: // meta controller
    For training iterations:
      forward-backward(); **Expensive**
    If good_model: **break;**
  For post-search training iterations:
    forward-backward(); **Expensive**

=>

Once-for-All:

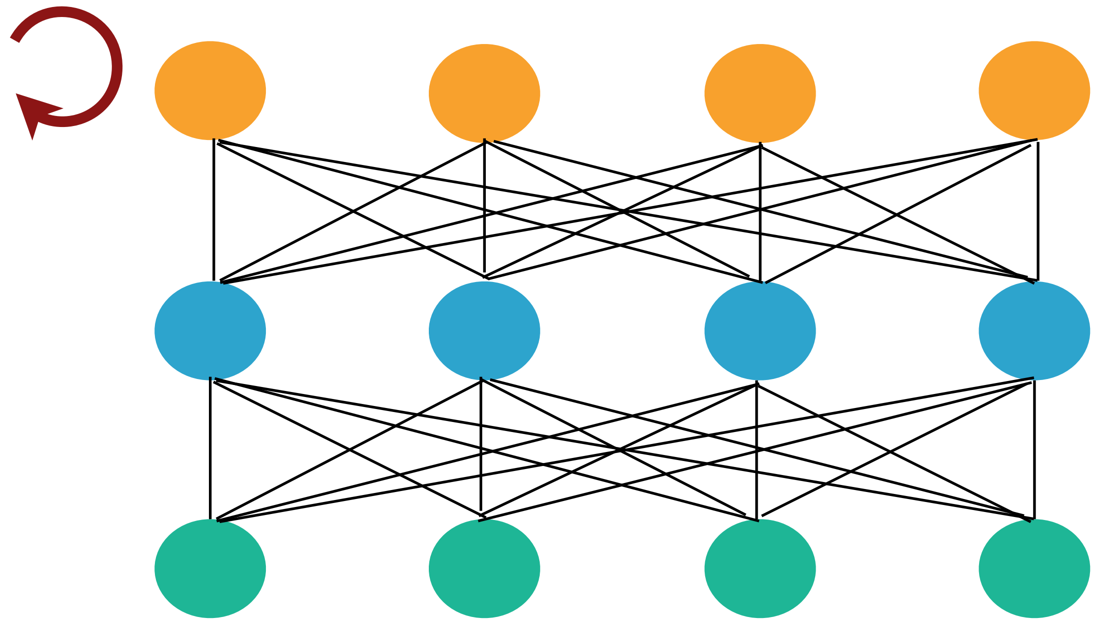
For OFA training iterations: **Expensive**
  forward-backward(); **training**

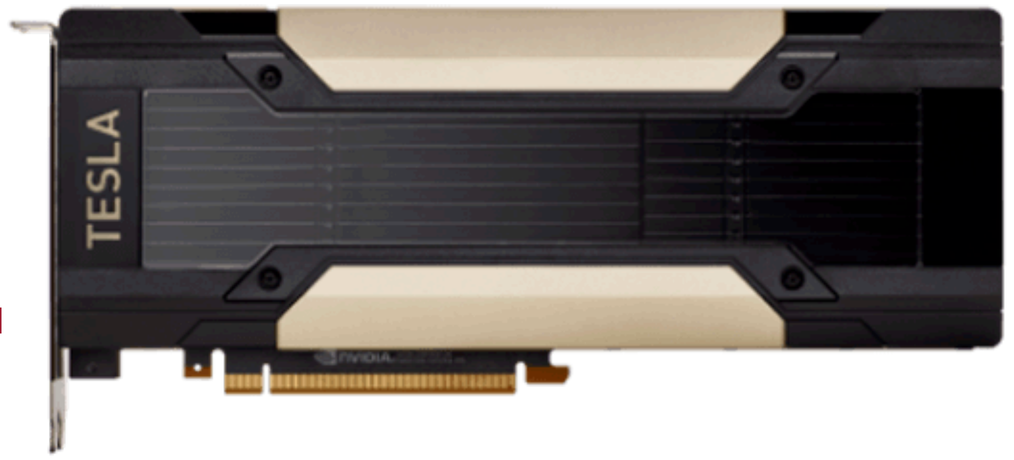
decouple

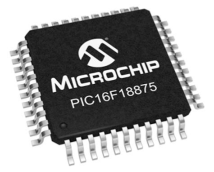
For devices: **search**
  For search episodes:
    sample from OFA; **Light-Weight**
    If good_model: **break;**
  directly deploy without training; **Light-Weight**

Once-for-All Network:
Decouple Model Training and Architecture Design

once-for-all network



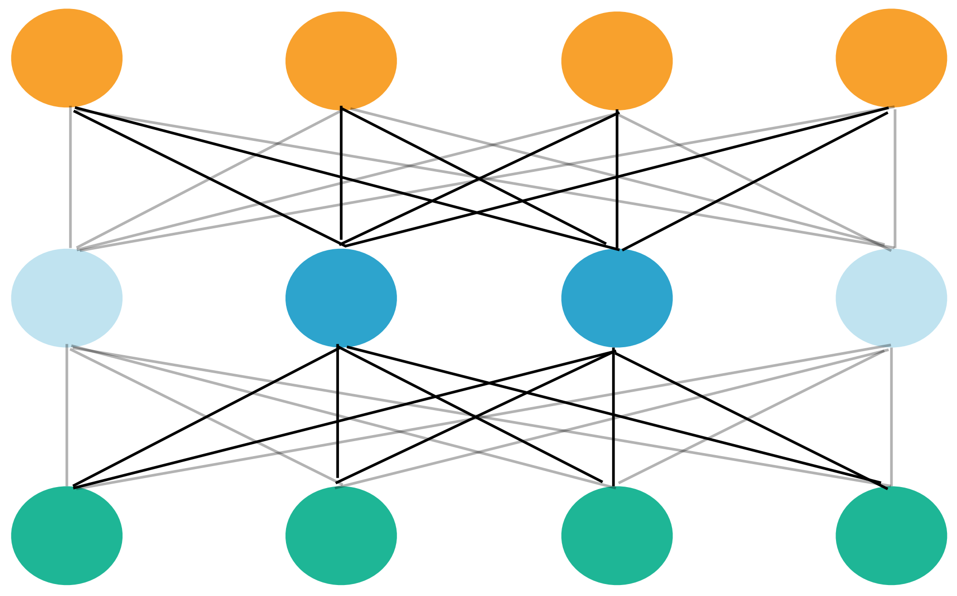


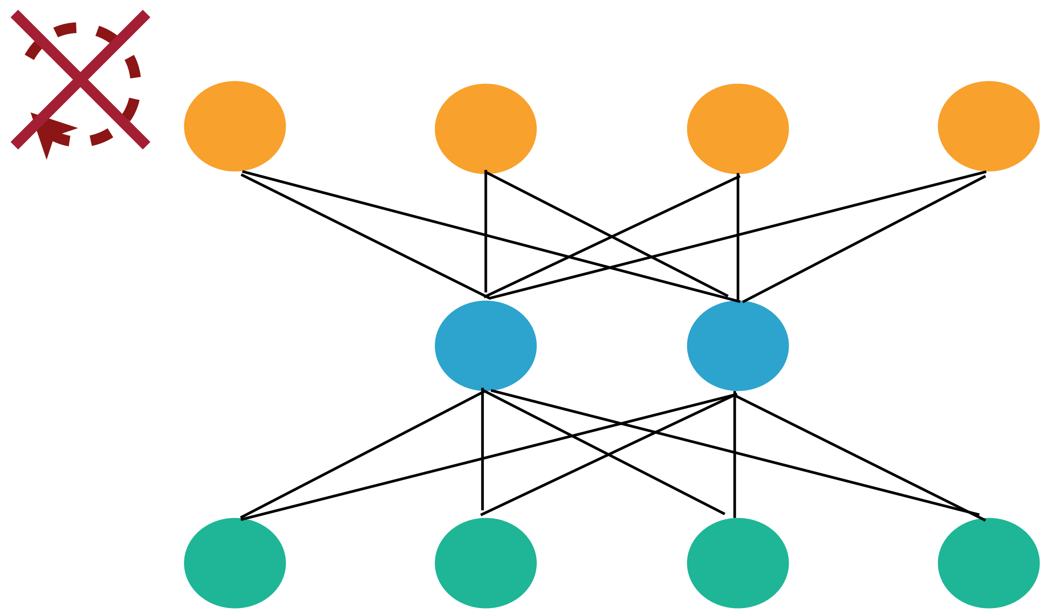


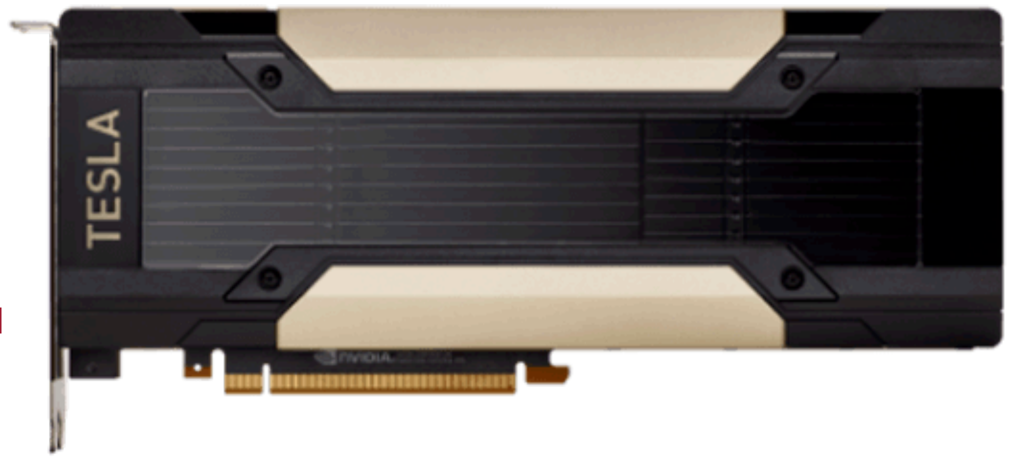
Once-for-all, ICLR'20

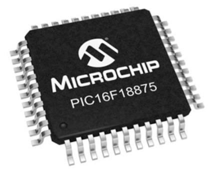
Once-for-All Network:
Decouple Model Training and Architecture Design

once-for-all network





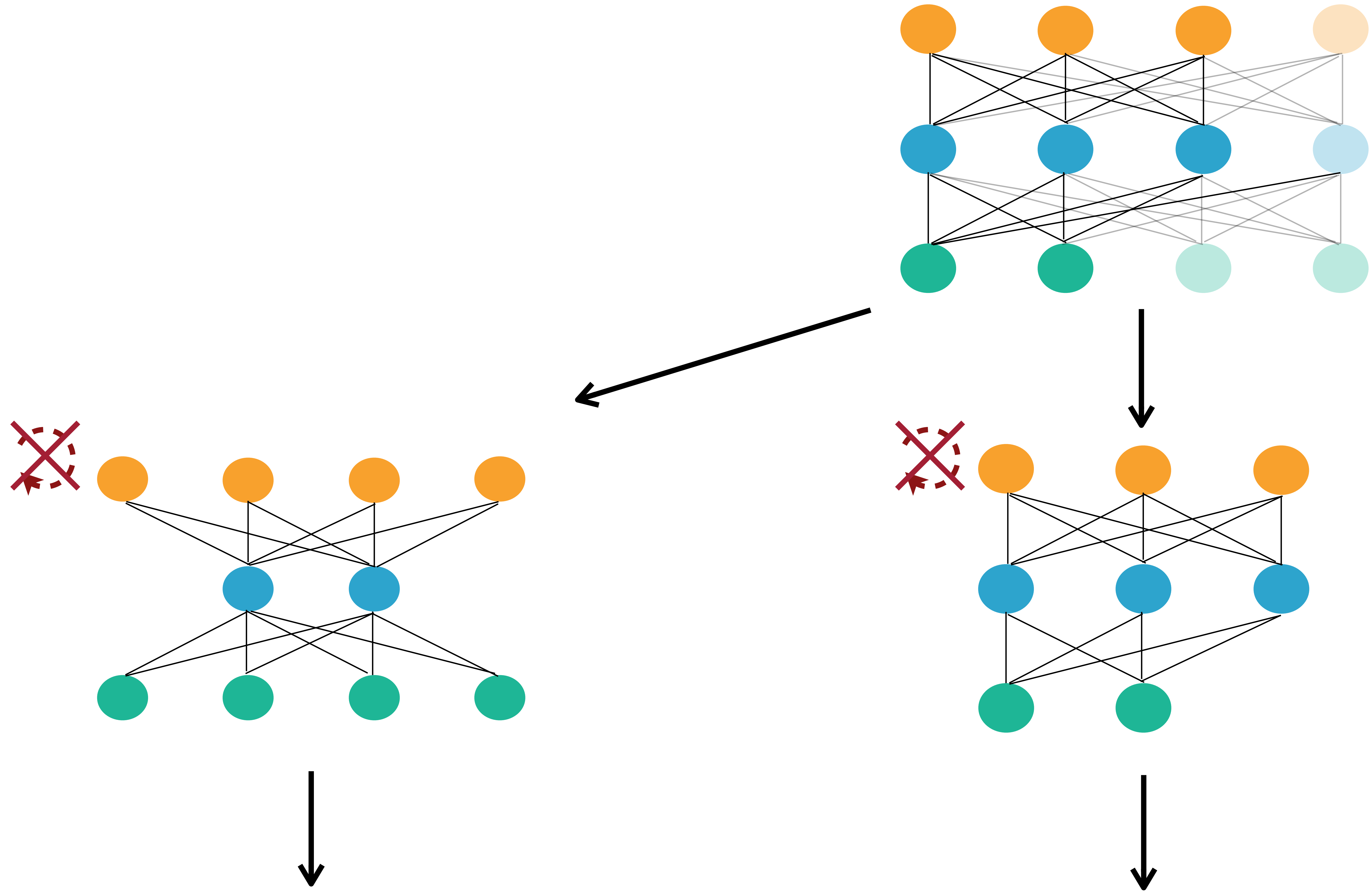


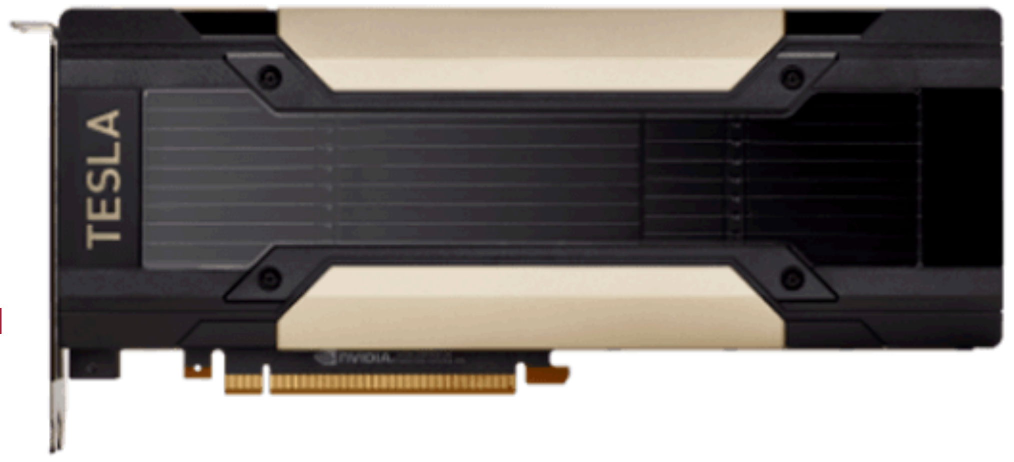


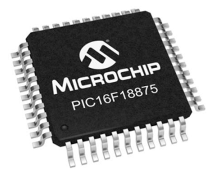
[Once-for-all](), ICLR'20

Once-for-All Network:
Decouple Model Training and Architecture Design

once-for-all network



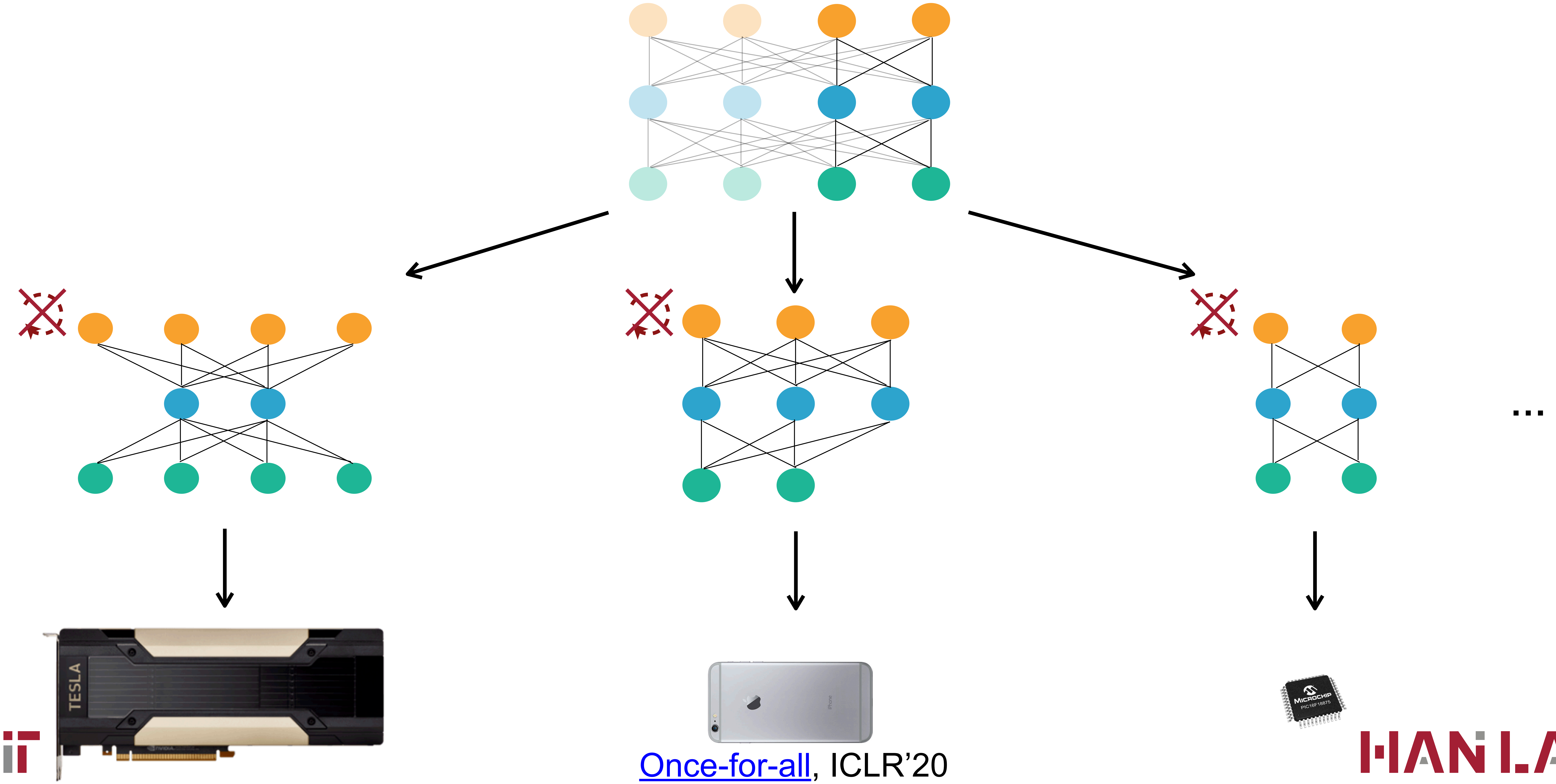




[Once-for-all](), ICLR'20

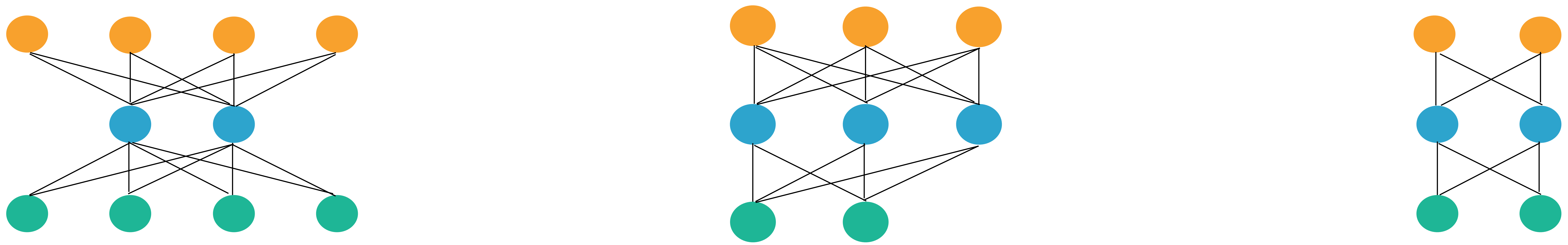
Once-for-All Network: Decouple Model Training and Architecture Design

once-for-all network



[Once-for-all](), ICLR'20

Challenge: how to prevent different subnetworks from interfering with each other?



Once-for-all, ICLR'20

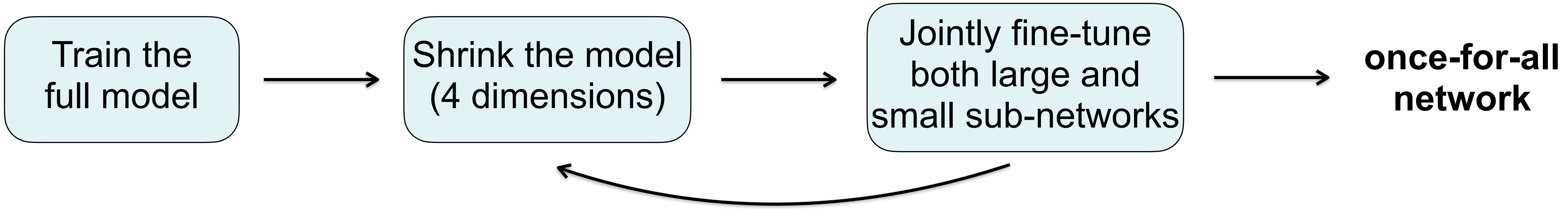
Solution: Progressive Shrinking

- Training once-for-all network is much more challenging than training a normal neural network given so many sub-networks to support.
- Progressive Shrinking can support more than 10^{19} **different sub-networks** in a single once-for-all network, covering 4 different dimensions: **resolution**, **kernel size**, **depth**, **width**.

Solution: Progressive Shrinking

- Training once-for-all network is much more challenging than training a normal neural network given so many sub-networks to support.
- Progressive Shrinking can support more than 10^{19} **different sub-networks** in a single once-for-all network, covering 4 different dimensions: **resolution**, **kernel size**, **depth**, **width**.

Progressive Shrinking



- Small sub-networks are nested in large sub-networks.
- Cast the training process of the once-for-all network as a progressive shrinking and joint fine-tuning process.

Connection to Network Pruning

Network Pruning

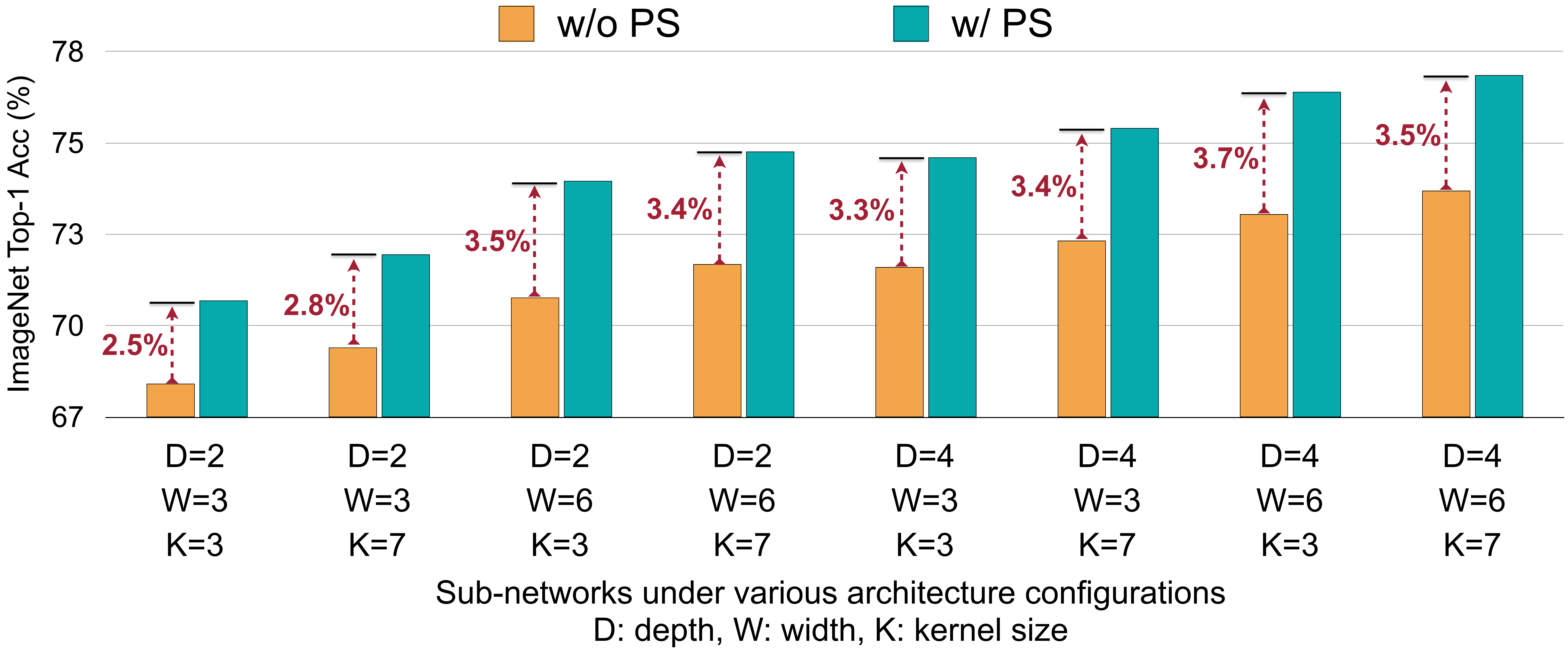
Train the full model → Shrink the model (only width) → Fine-tune the small net → **single pruned network**

Progressive Shrinking

Train the full model → Shrink the model (4 dimensions) → Fine-tune both large and small sub-nets → **once-for-all network**

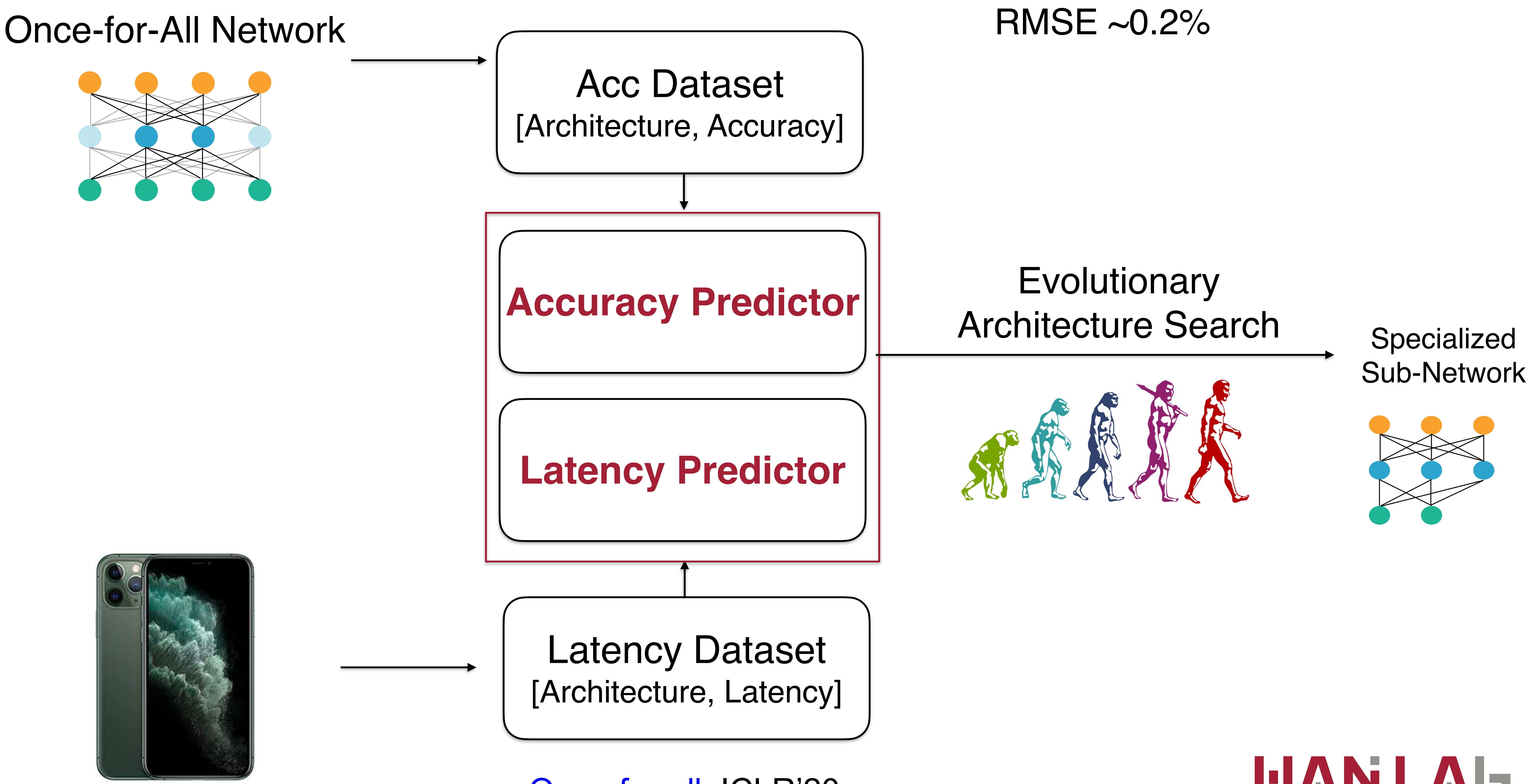
- Progressive shrinking can be viewed as a generalized network pruning with much higher flexibility across 4 dimensions.

Performances of Sub-networks on ImageNet



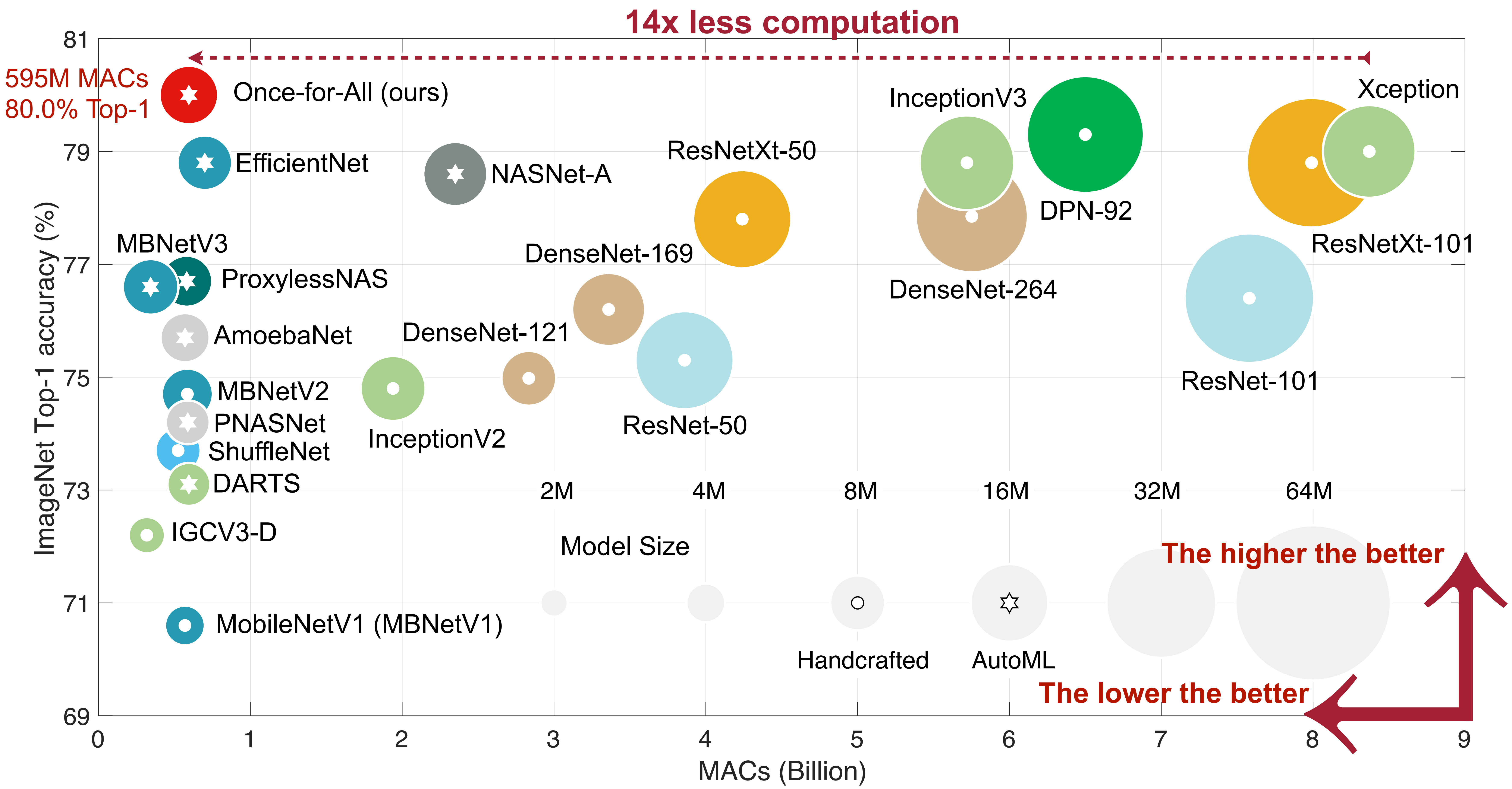
- Progressive shrinking consistently improves accuracy of sub-networks on ImageNet.

Accuracy / Latency Predictor



[Once-for-all](), ICLR'20

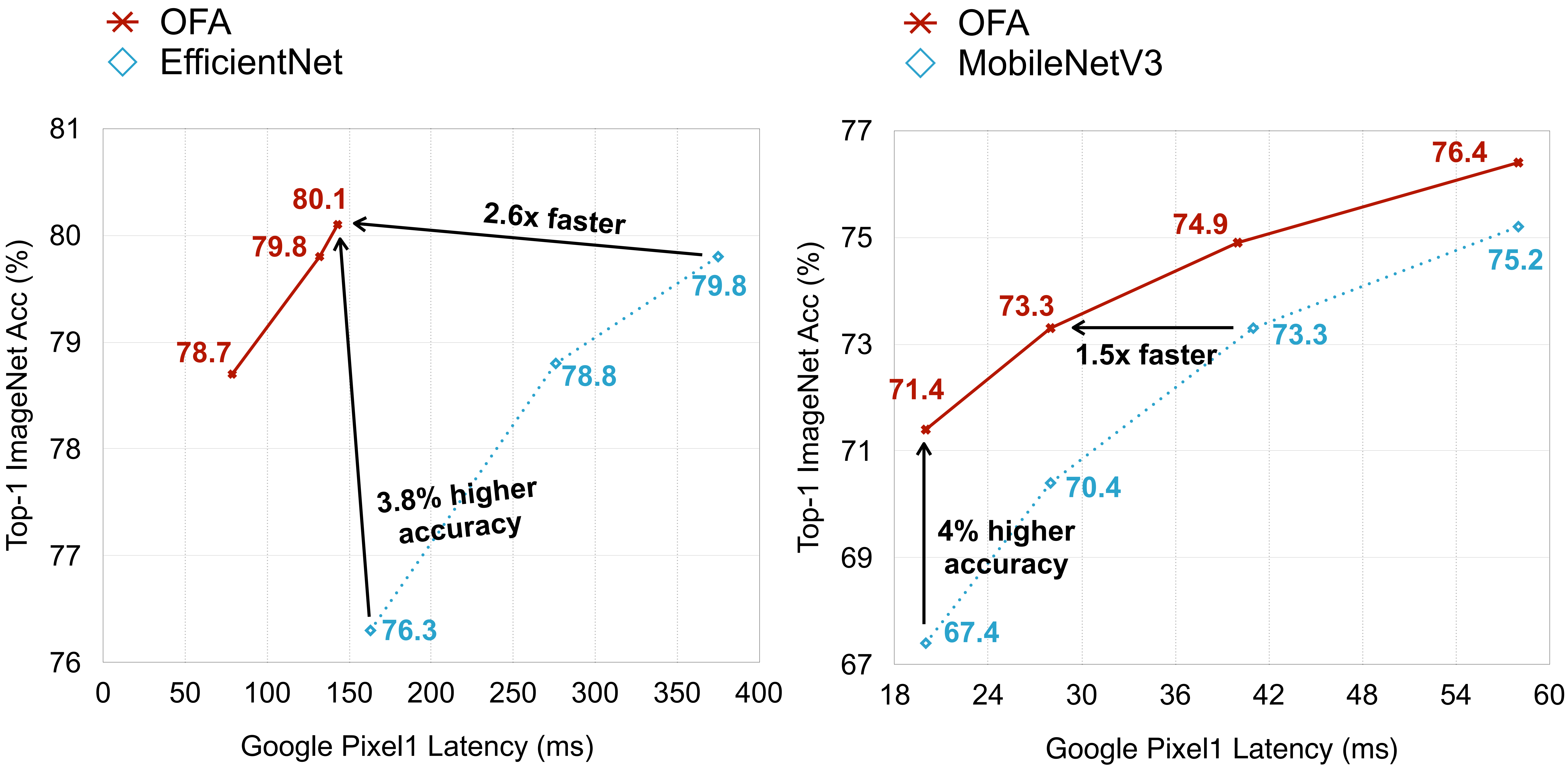
OFA: 80% Top-1 Accuracy on ImageNet



- Once-for-all sets a new state-of-the-art **80% ImageNet top-1 accuracy** under the mobile vision setting (< 600M MACs).

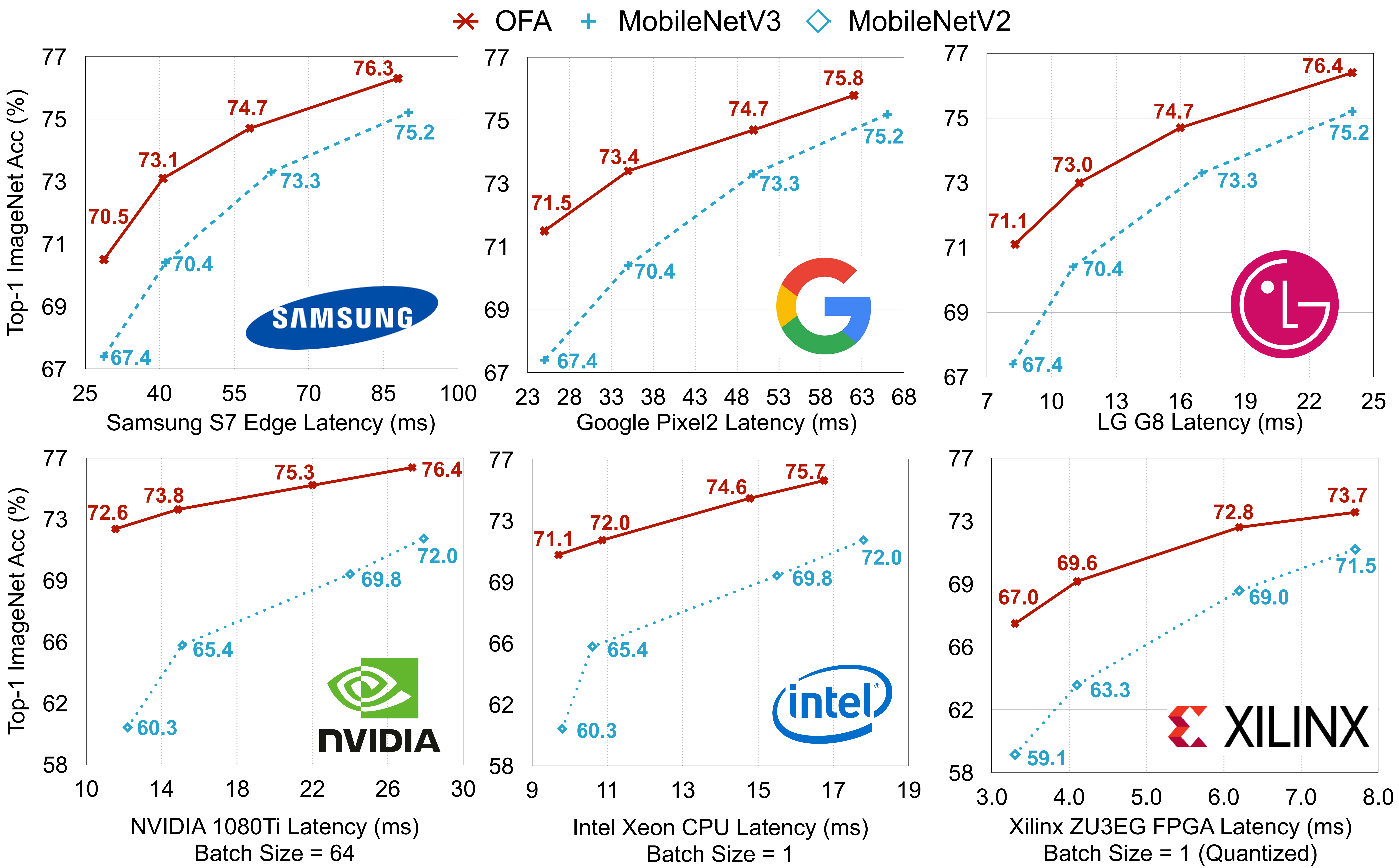
Once-for-all, ICLR’20

Accuracy & Latency Improvement



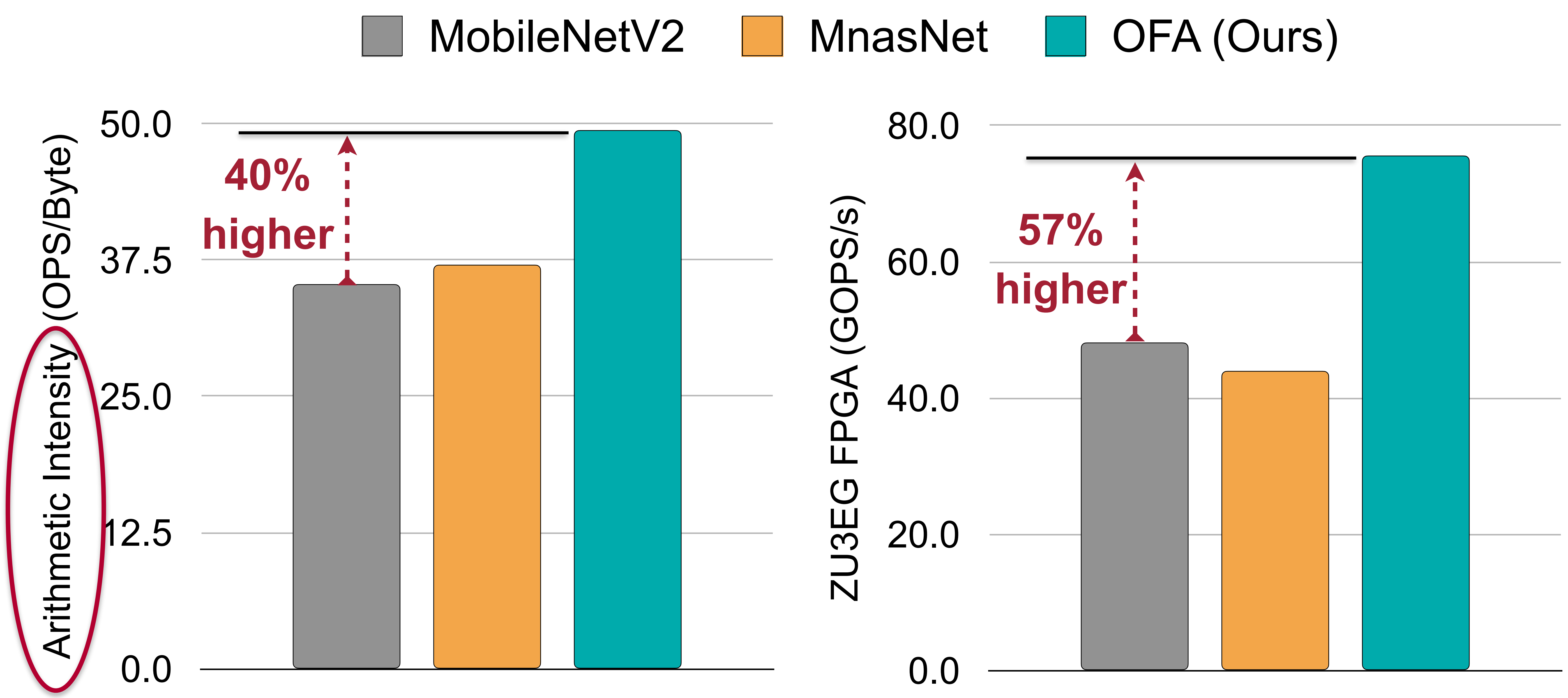
- Training from scratch cannot achieve the same level of accuracy

OFA Enables Fast Specialization on Diverse Hardware Platforms



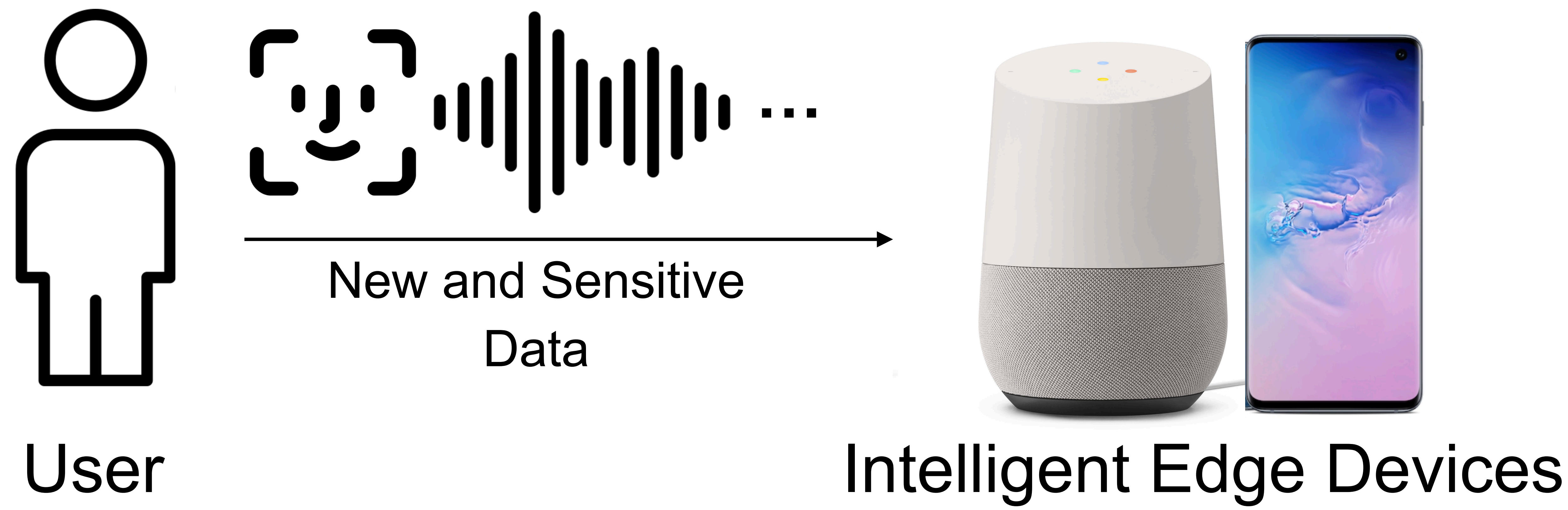
OFA for FPGA

Specialized NN architecture on specialized hardware architecture



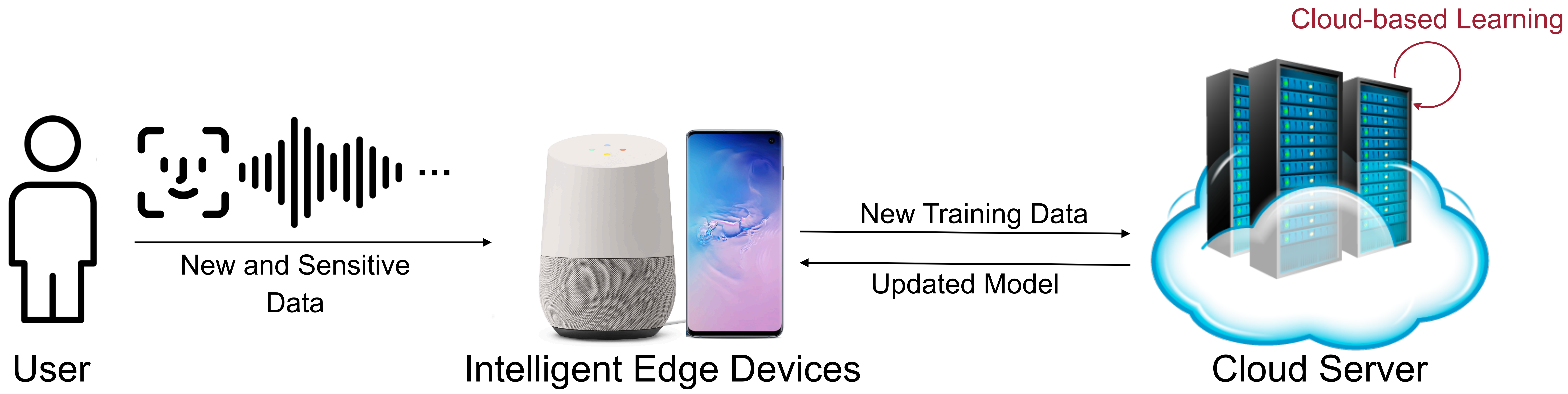
Measured results on XILINX FPGA

Adapt to Newly Collected Data on the Edge



- Customization: AI systems need to continually adapt to new data collected from the sensors.

Cloud-based Learning

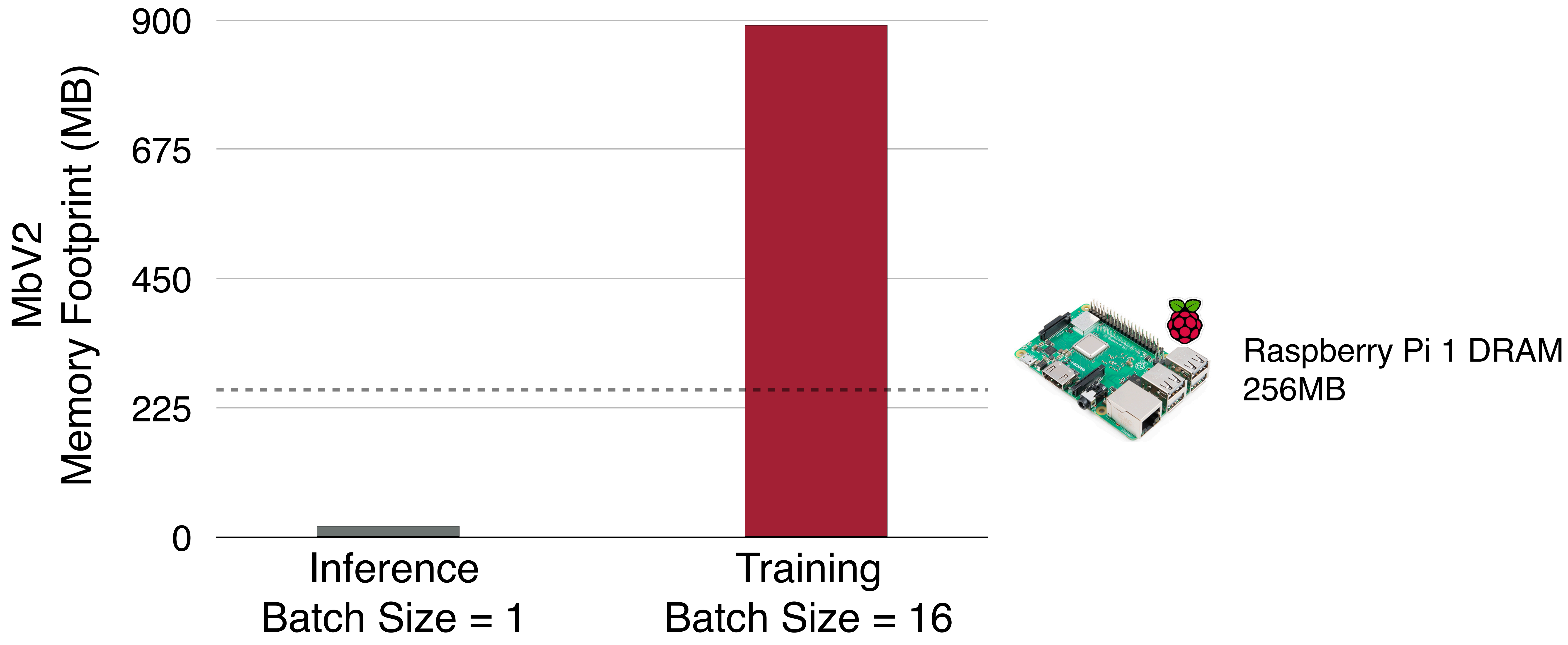


- Customization: AI systems need to continually adapt to new data collected from the sensors.

On-device Learning

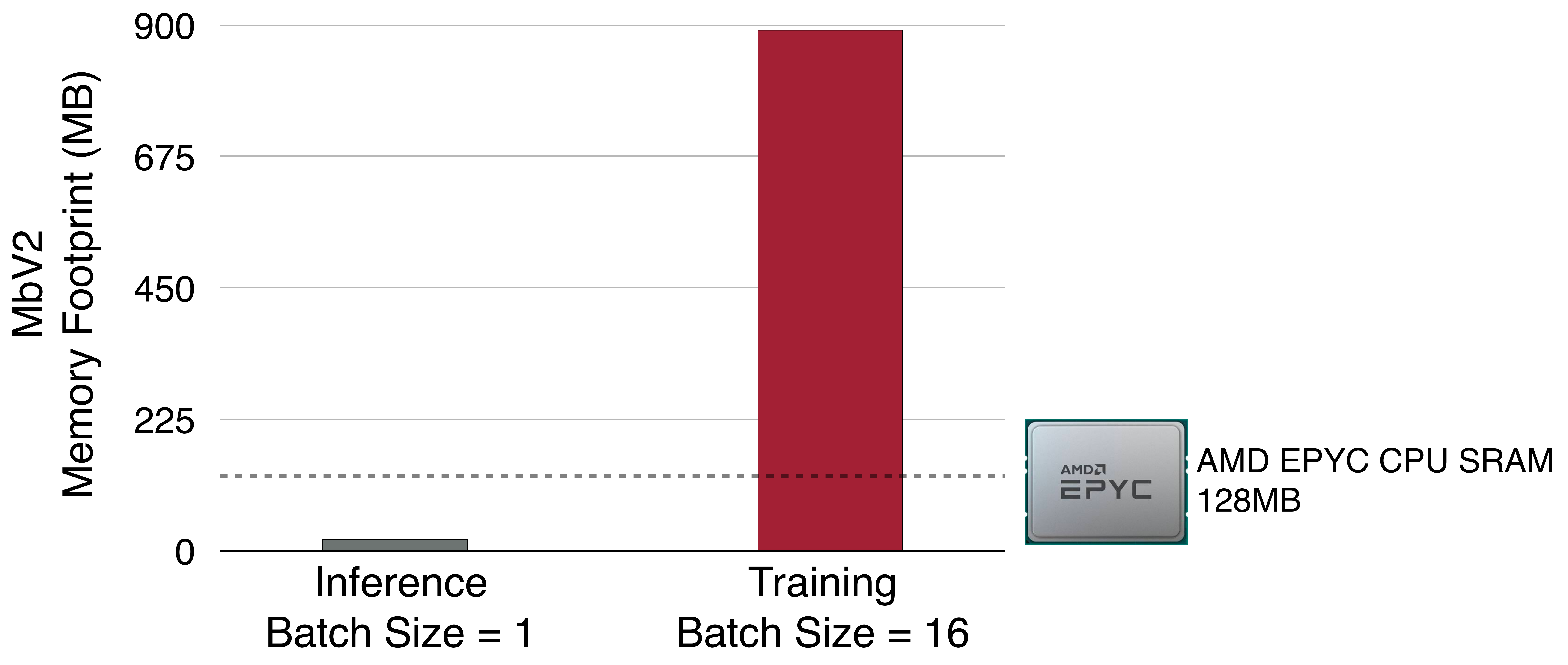
- Customization: AI systems need to continually adapt to new data collected from the sensors.
- Security: Data cannot leave devices because of security and regularization.

Training Memory is much Larger than Inference



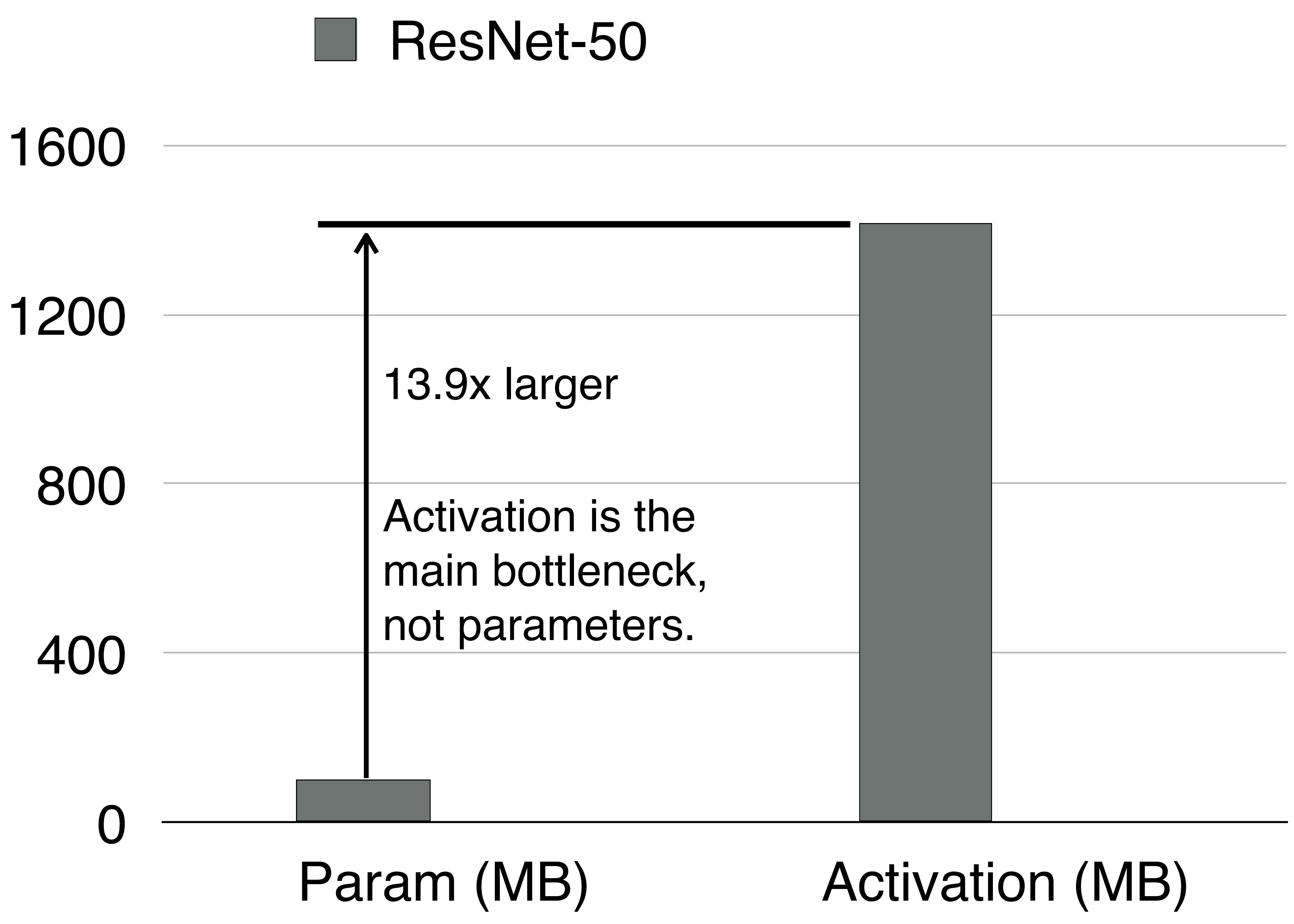
- Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.

Training Memory is much Larger than Inference



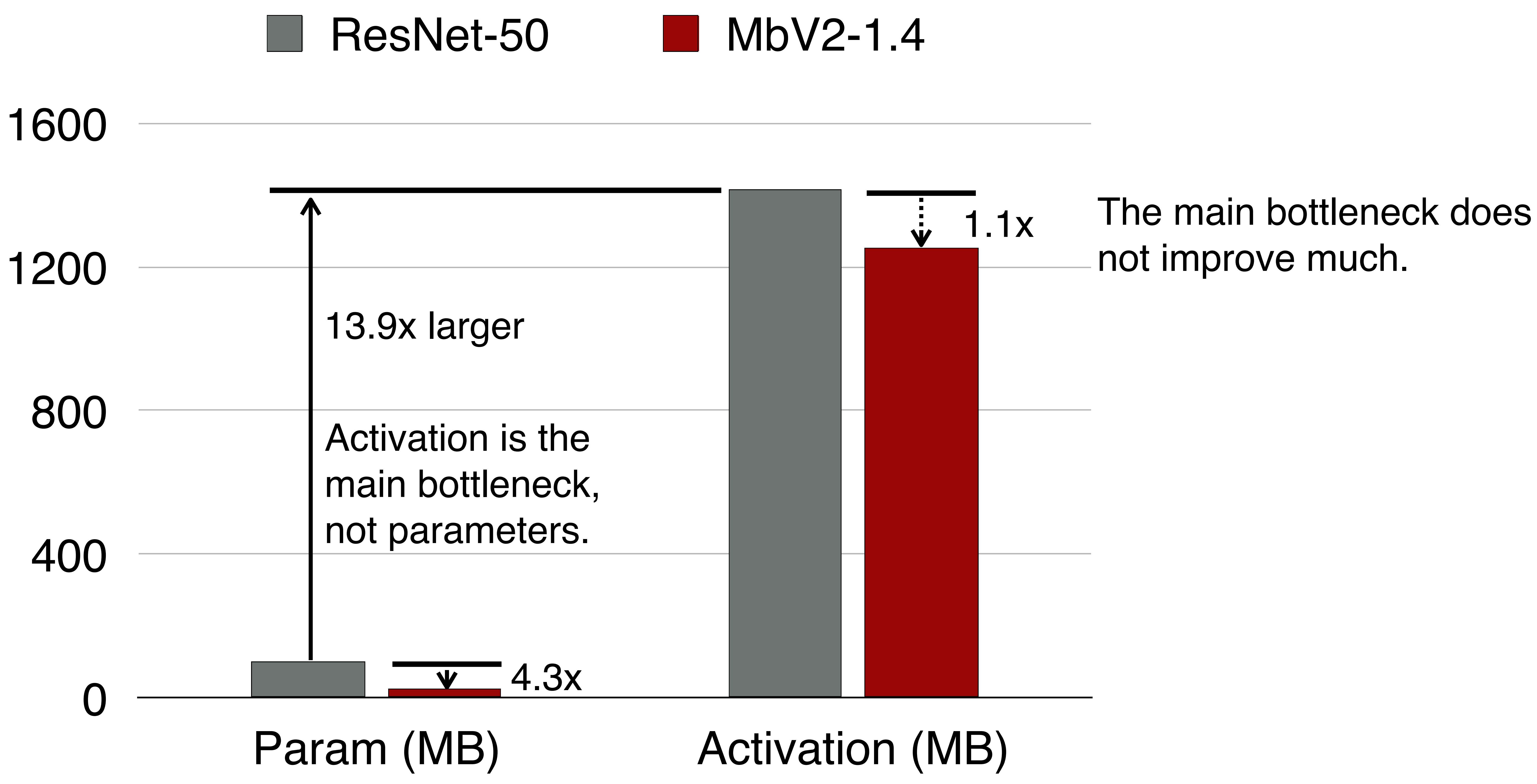
- Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.
- Edge devices are energy-constrained. Failing to fit the training process into the energy-efficient on-chip SRAM will significantly increase the energy cost.

Activation is the Memory Bottleneck, not Parameters



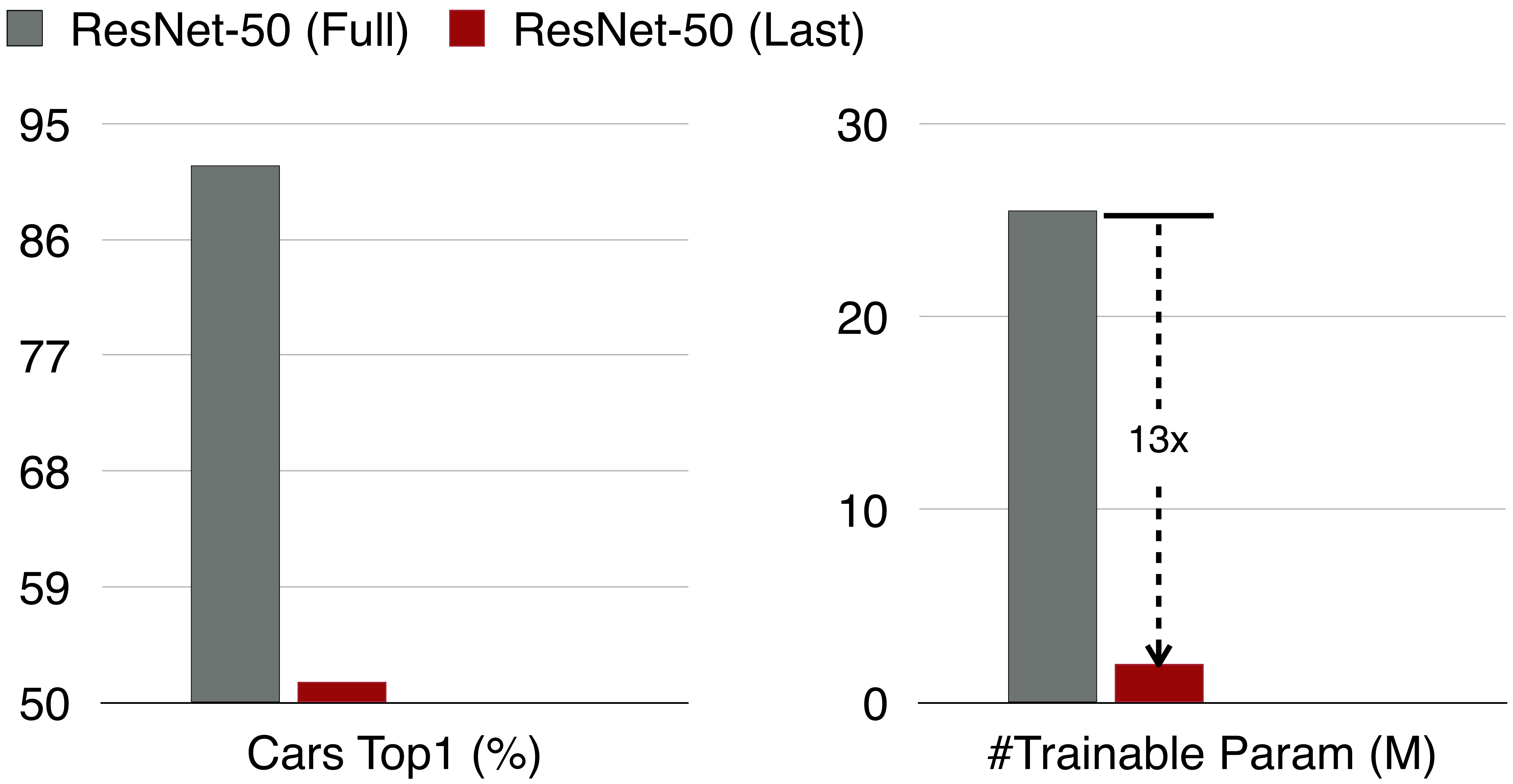
- Activation is the main bottleneck for on-device learning, not parameters.

Activation is the Memory Bottleneck, not Parameters



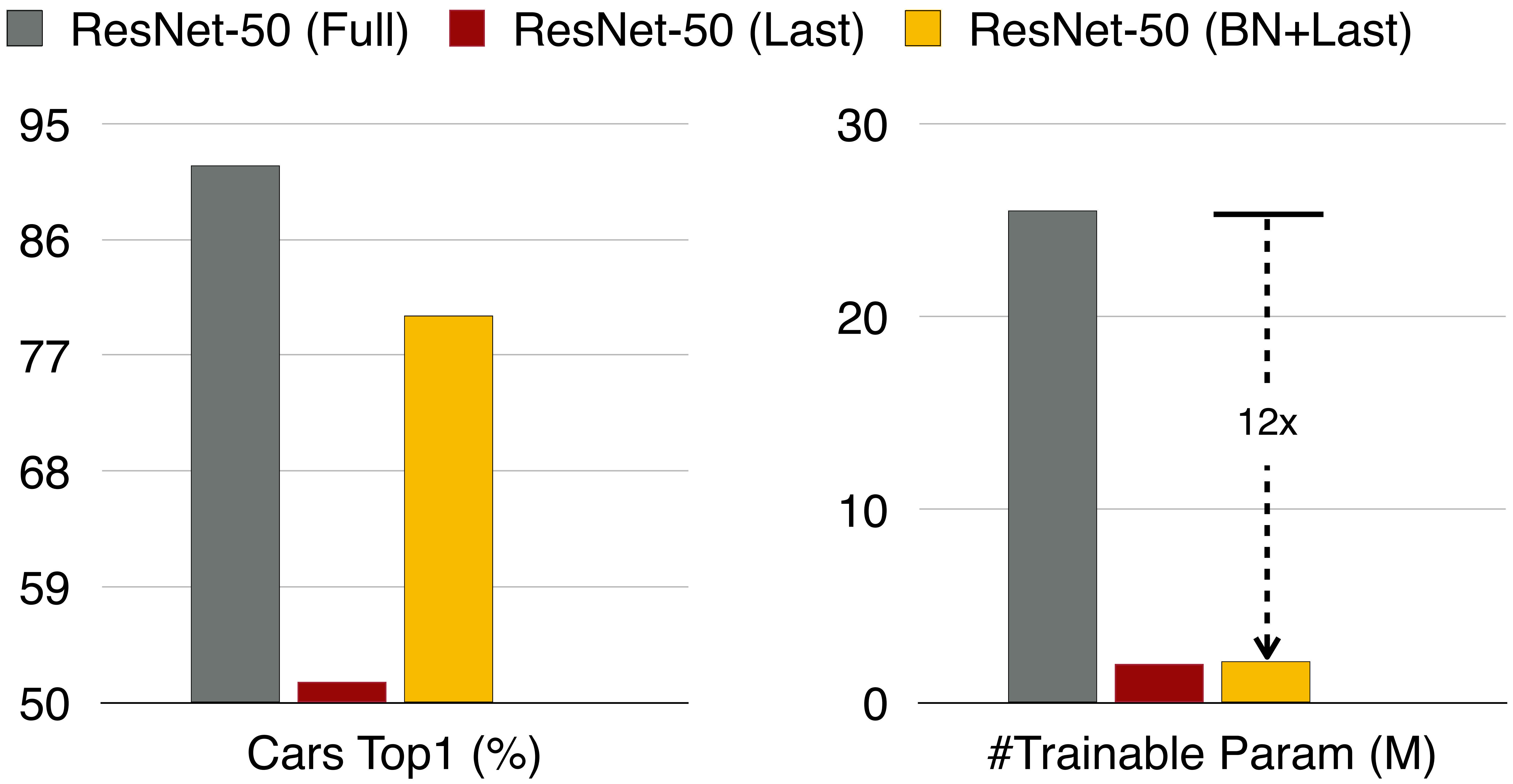
- Activation is the main bottleneck for on-device learning, not parameters.
- Previous methods focus on reducing the number of parameters or FLOPs, while the main bottleneck does not improve much.

Related Work: Parameter-Efficient Transfer Learning



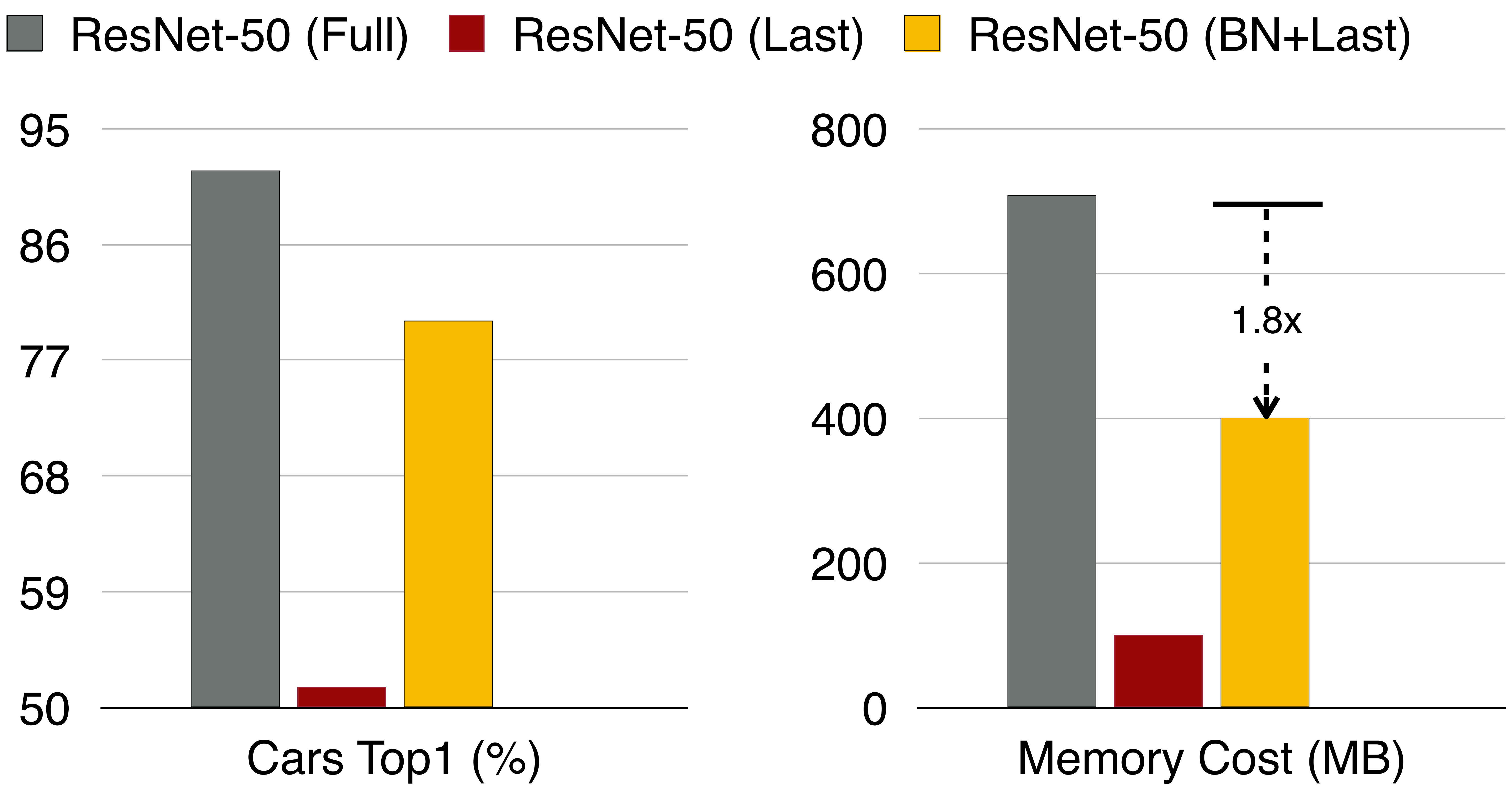
- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.

Related Work: Parameter-Efficient Transfer Learning



- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency.

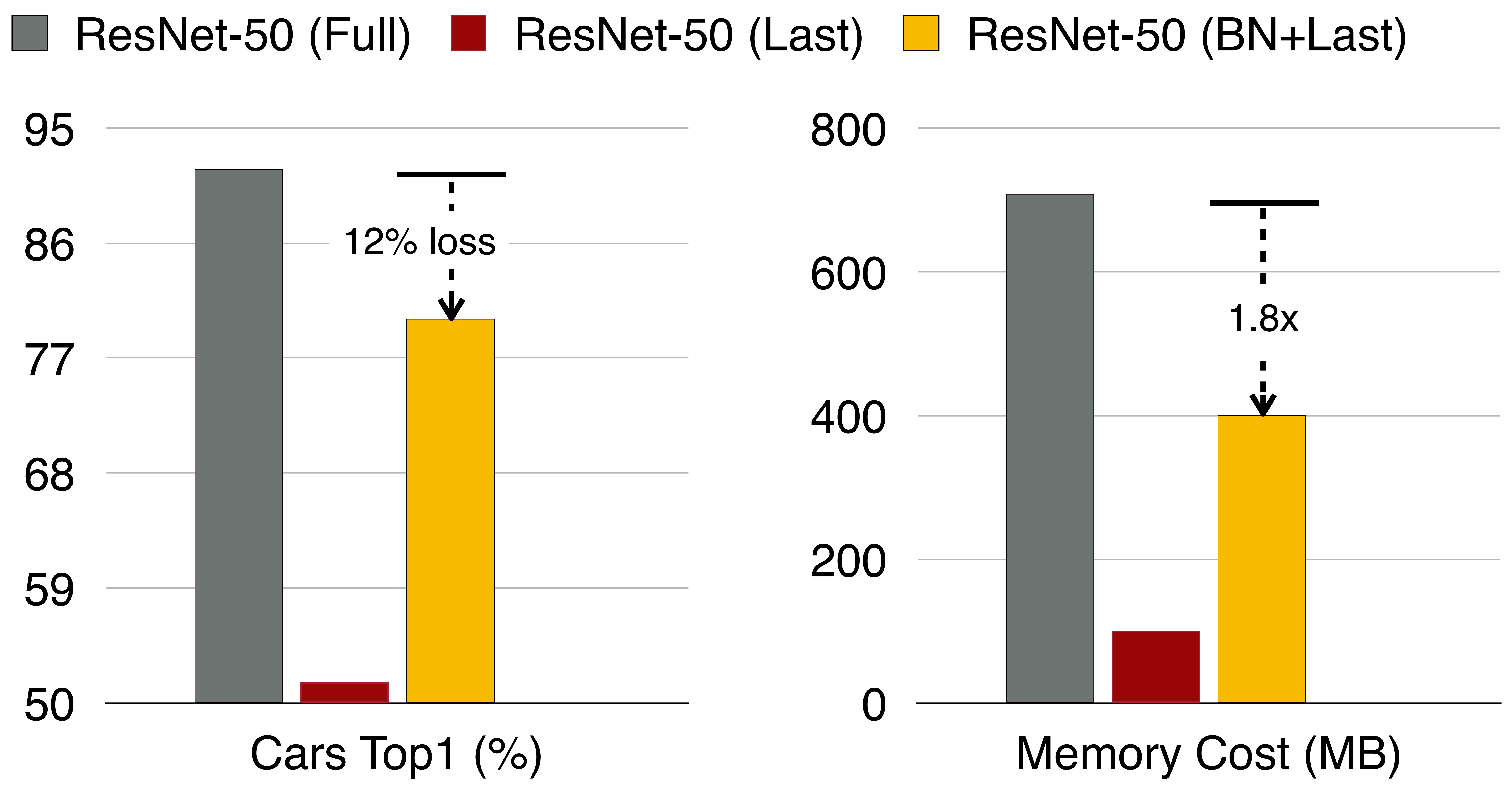
Related Work: Parameter-Efficient Transfer Learning



Parameter-efficiency does not directly translate to memory-efficiency

- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the memory saving is limited.**

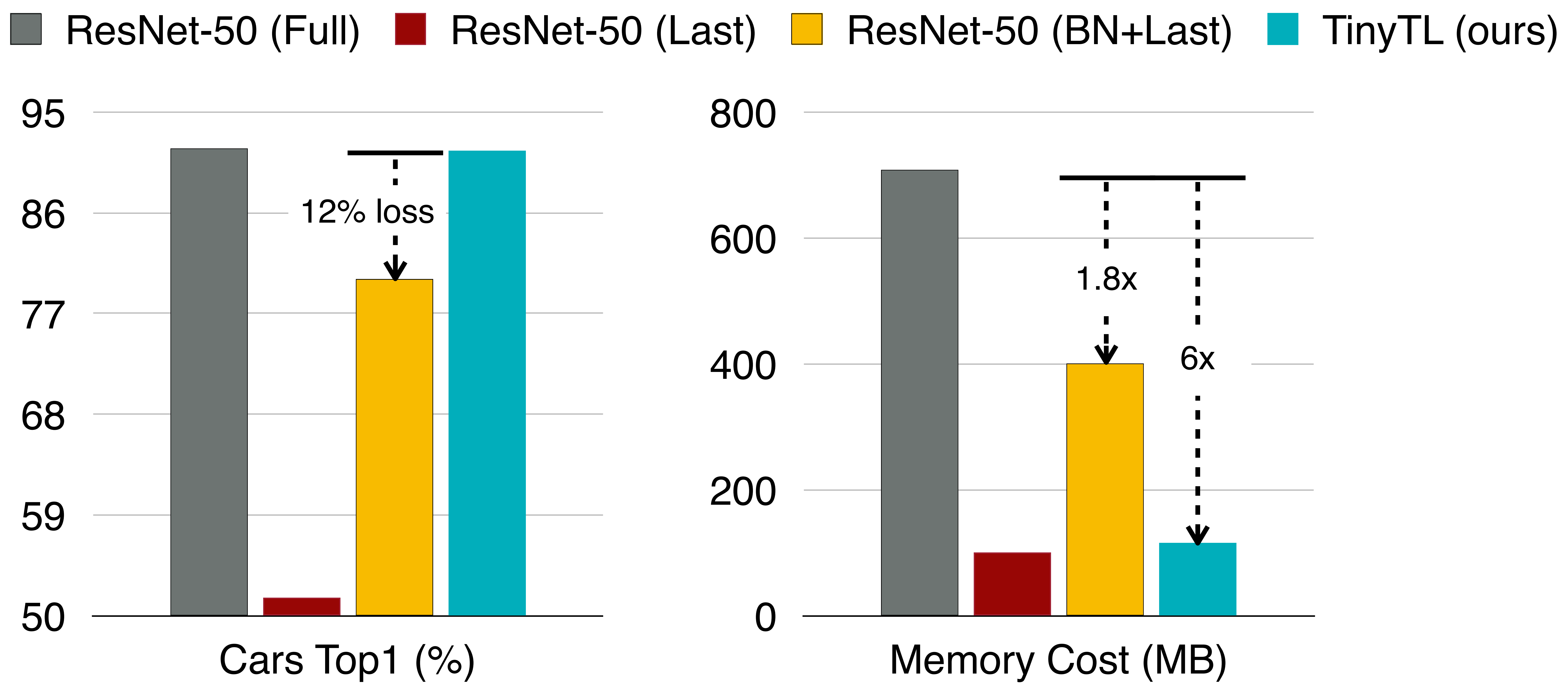
Related Work: Parameter-Efficient Transfer Learning



Parameter-efficiency does not directly translate to memory-efficiency

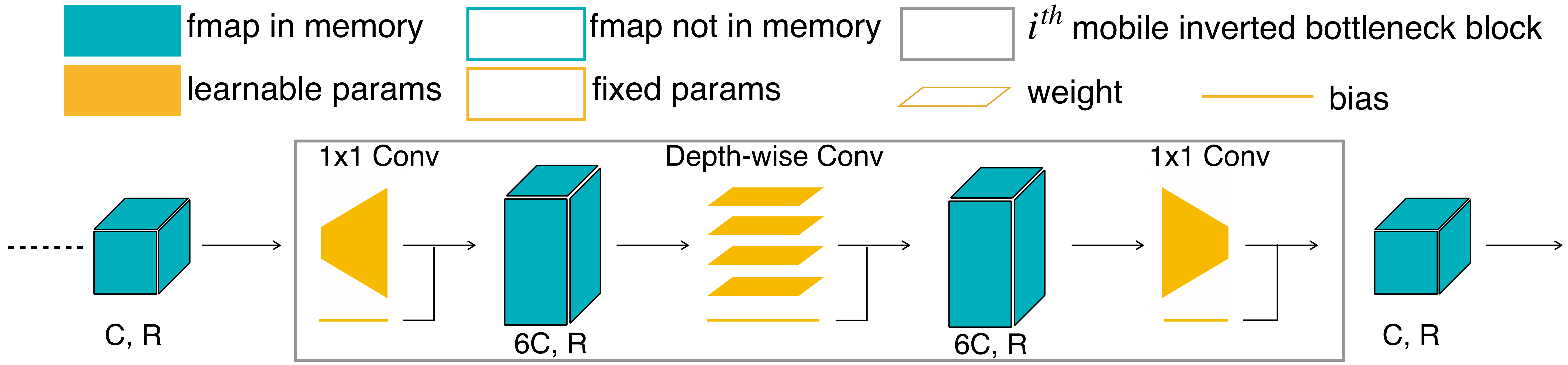
- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the memory saving is limited. The accuracy loss is still significant.**

TinyTL: Memory-Efficient Transfer Learning



- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the memory saving is limited. The accuracy loss is still significant.**

Updating Weights is Memory-expensive While Updating Biases is Memory-efficient



Fine-tune the full network (Conventional)

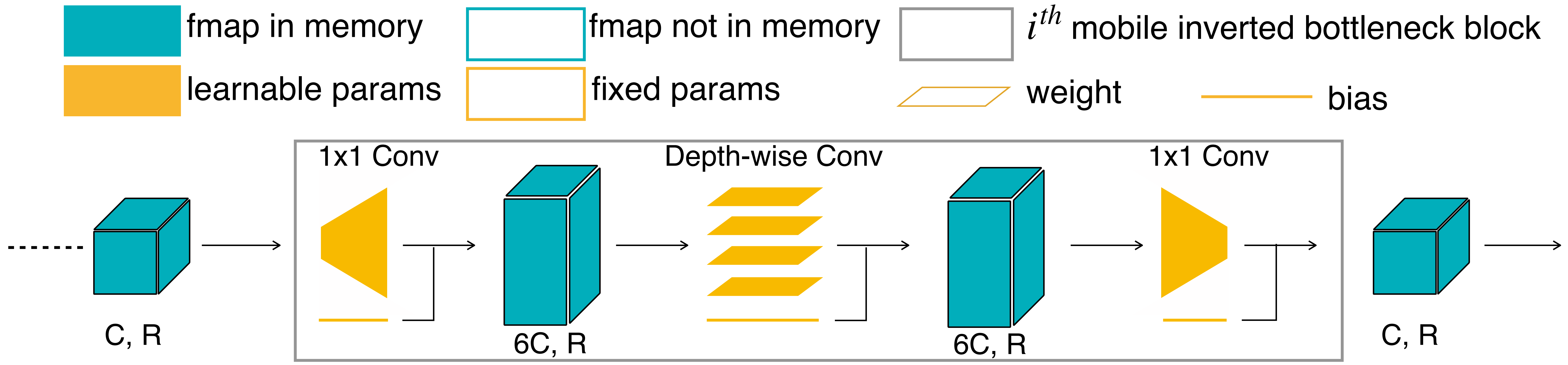
Linear Layer

Forward: $\mathbf{a}_{i+1} = \mathbf{a}_i \mathbf{W}_i + \mathbf{b}_i$

Backward: $\frac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \frac{\partial L}{\partial \mathbf{a}_{i+1}}, \qquad \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial \mathbf{a}_{i+1}} = \frac{\partial L}{\partial \mathbf{a}_{i+2}} \mathbf{W}_{i+1}^T$

Updating weights requires storing intermediate activations
Updating biases does not

Updating Weights is Memory-expensive While Updating Biases is Memory-efficient



Fine-tune the full network (Conventional)

Linear Layer

Forward: $\mathbf{a}_{i+1} = \mathbf{a}_i \mathbf{W}_i + \mathbf{b}_i$

Backward: $\frac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \frac{\partial L}{\partial \mathbf{a}_{i+1}}, \quad \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial \mathbf{a}_{i+1}} = \frac{\partial L}{\partial \mathbf{a}_{i+2}} \mathbf{W}_{i+1}^T$

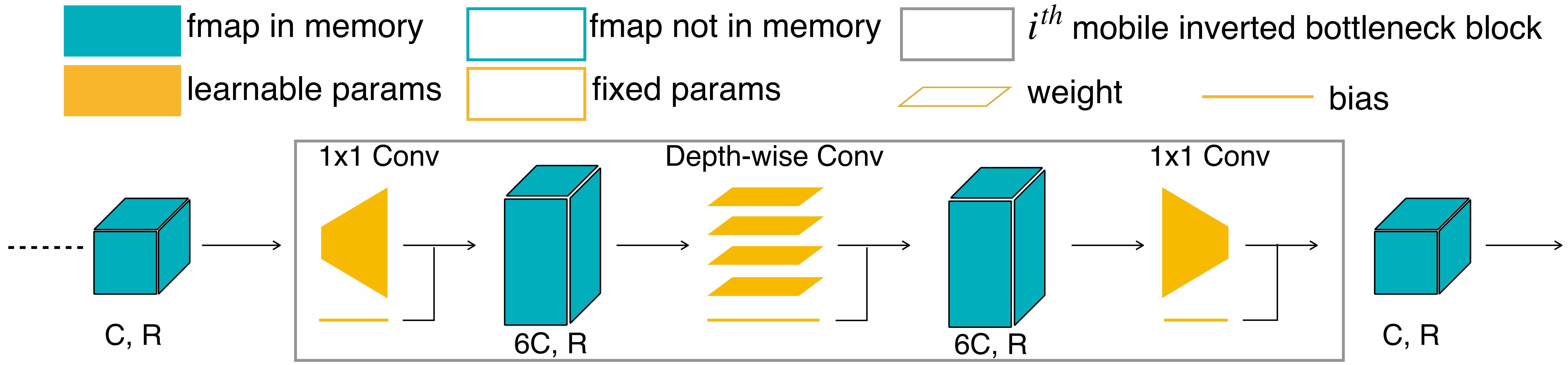
Updating weights requires storing intermediate activations

Updating biases does not

- Convolution layers and normalization layers (e.g., BN) can be viewed as special types of linear layers. Thus, this property is also applicable to them.

TinyTL, NeurIPS'20

Updating Weights is Memory-expensive While Updating Biases is Memory-efficient



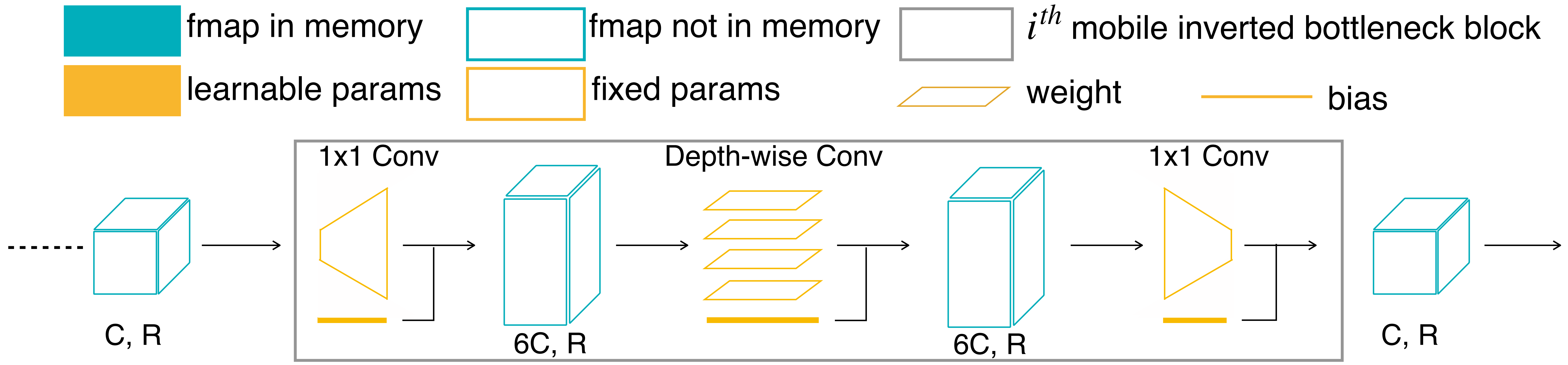
Fine-tune the full network (Conventional)

Layer Type	Forward	Backward	Memory Cost
ReLU	$\mathbf{a}_{i+1} = \max(0, \mathbf{a}_i)$	$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \circ \mathbf{1}_{\mathbf{a}_i \geq 0}$	$\lvert\mathbf{a}_i\rvert$ bits
sigmoid	$\mathbf{a}_{i+1} = \sigma(\mathbf{a}_i) = \frac{1}{1+\exp(-\mathbf{a}_i)}$	$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \circ \sigma(\mathbf{a}_i) \circ (1 - \sigma(\mathbf{a}_i))$	32 $\lvert\mathbf{a}_i\rvert$ bits
h-swish [8]	$\mathbf{a}_{i+1} = \mathbf{a}_i \circ \frac{\mathrm{ReLU6}(\mathbf{a}_i+3)}{6}$	$\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{i+1}} \circ (\frac{\mathrm{ReLU6}(\mathbf{a}_i+3)}{6} + \mathbf{a}_i \circ \frac{\mathbf{1}_{-3 \leq \mathbf{a}_i \leq 3}}{6})$	32 $\lvert\mathbf{a}_i\rvert$ bits

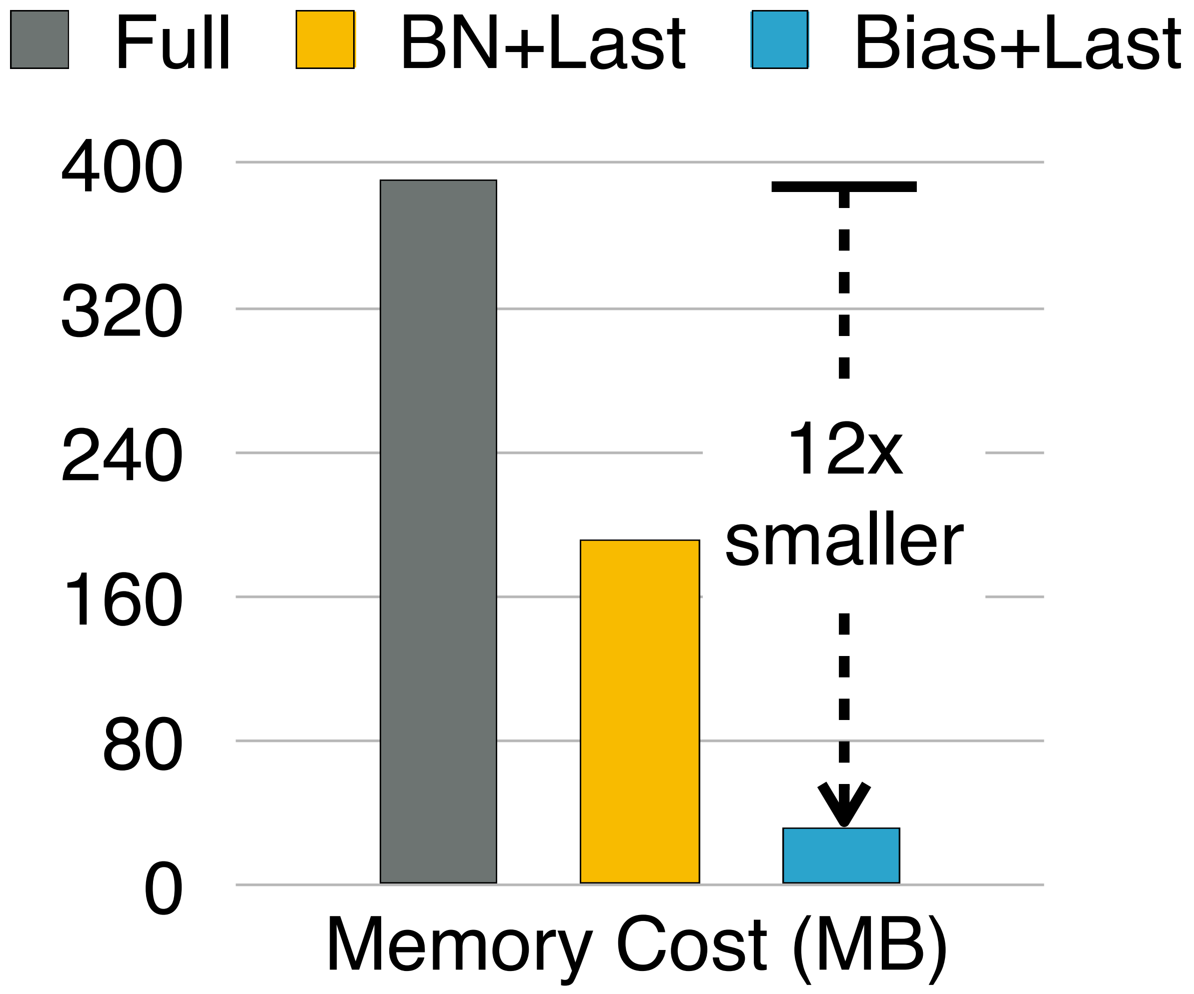
ReLU is memory-efficient

Smooth activation functions (e.g., sigmoid, swish, hard-swish) are memory-expensive

TinyTL: Fine-tune Bias Only



Fine-tune bias only

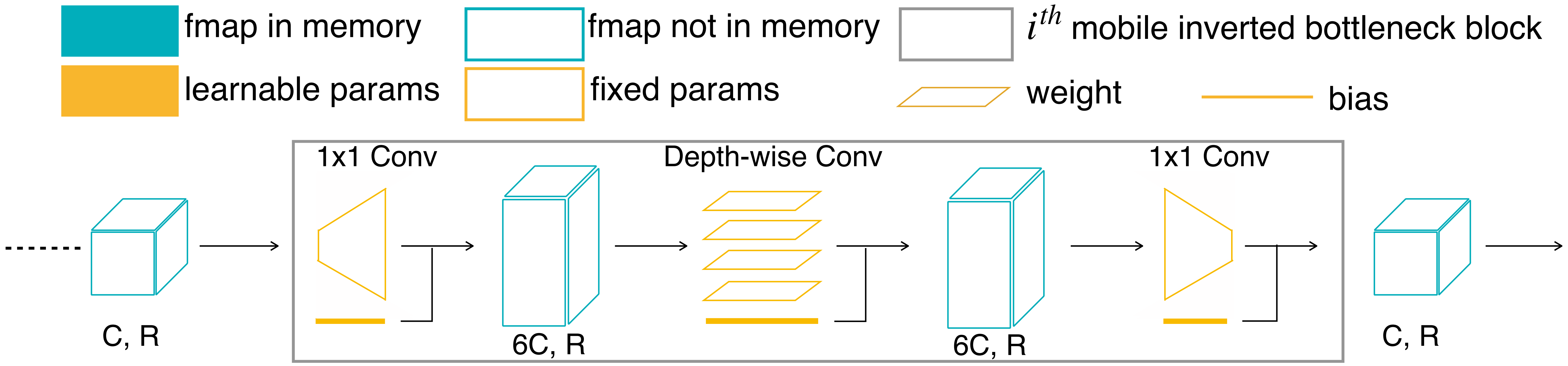


Freeze weights, only fine-tune biases

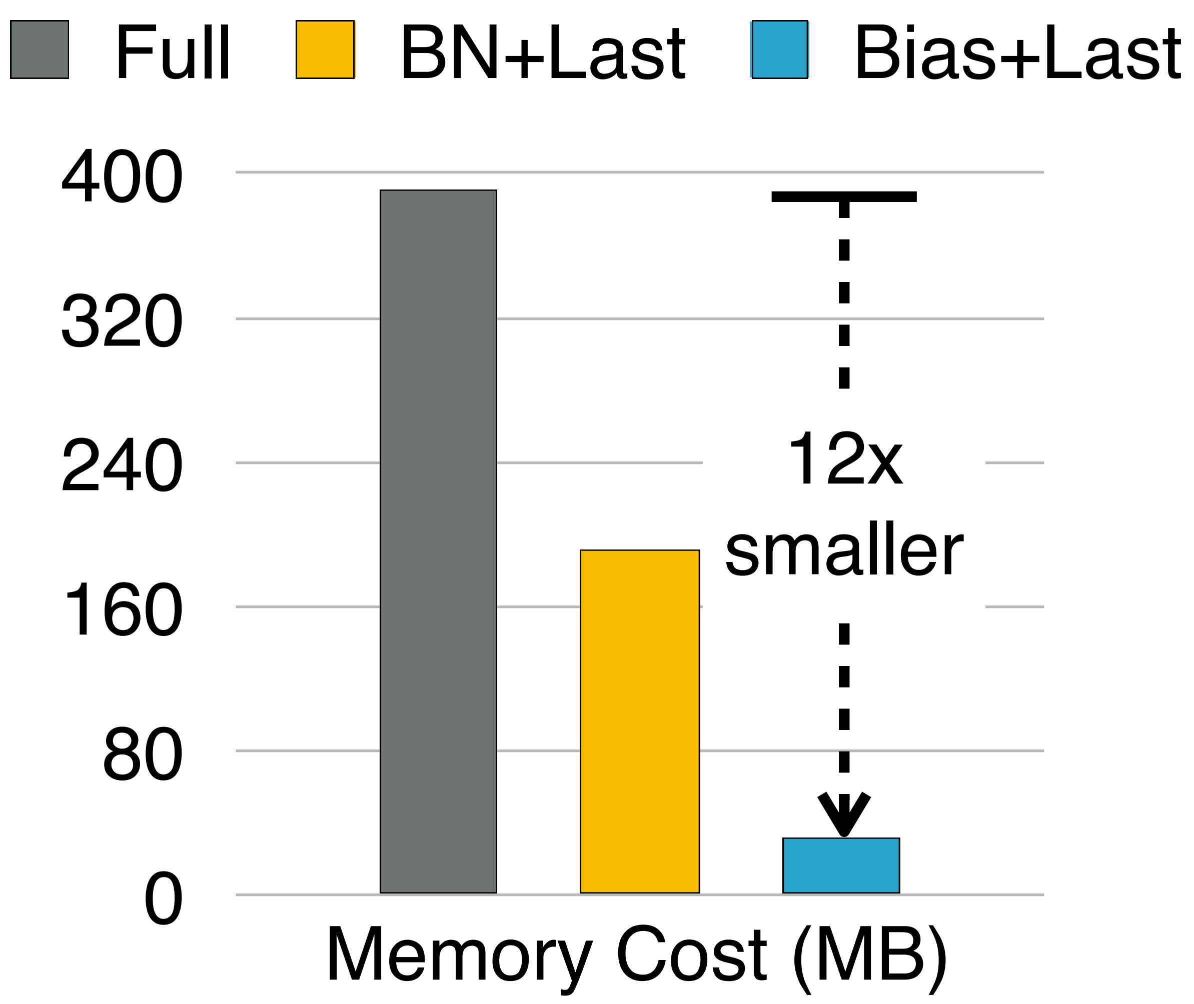
=> save 12x memory

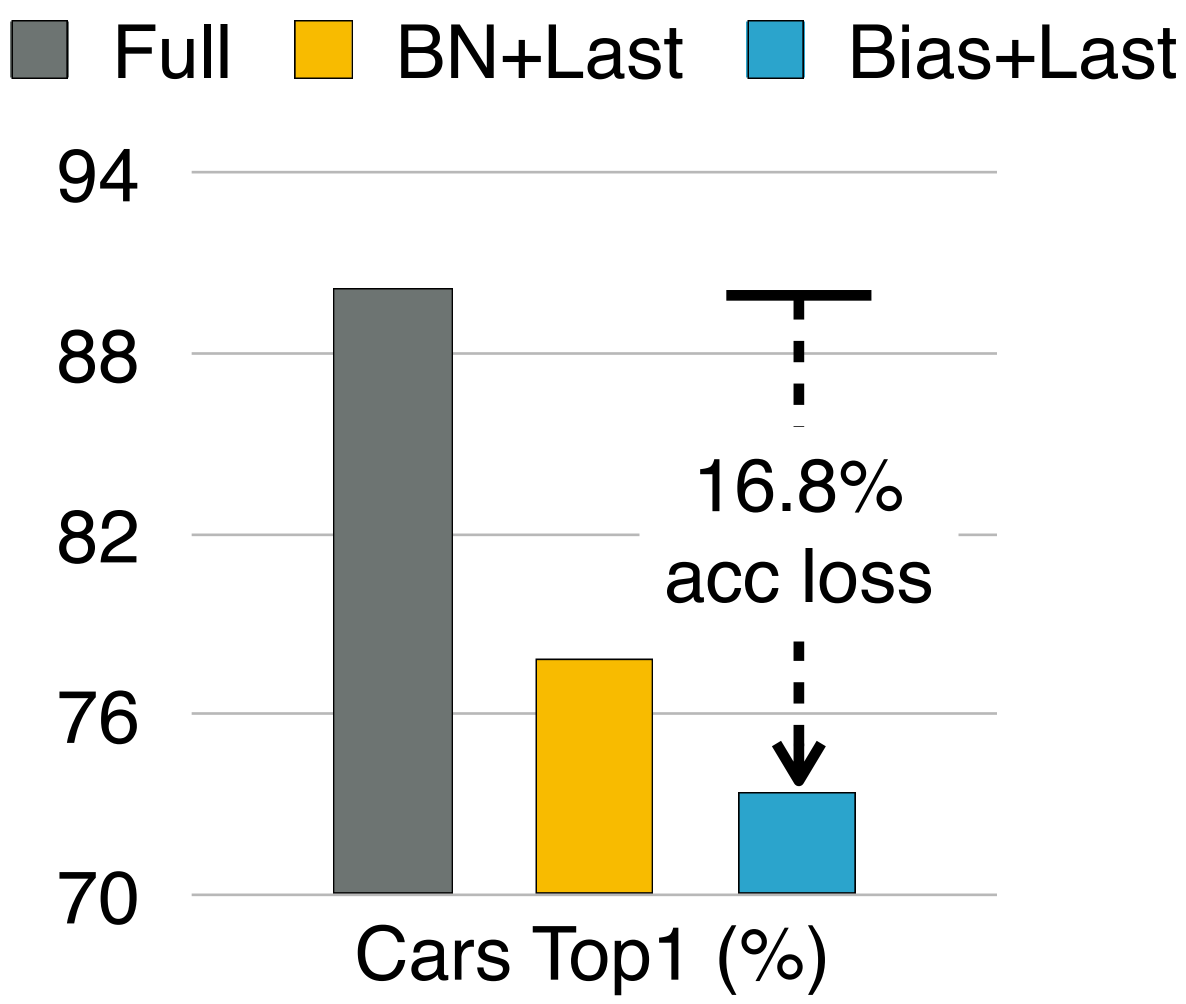
[TinyTL](), NeurIPS'20

TinyTL: Fine-tune Bias Only



Fine-tune bias only



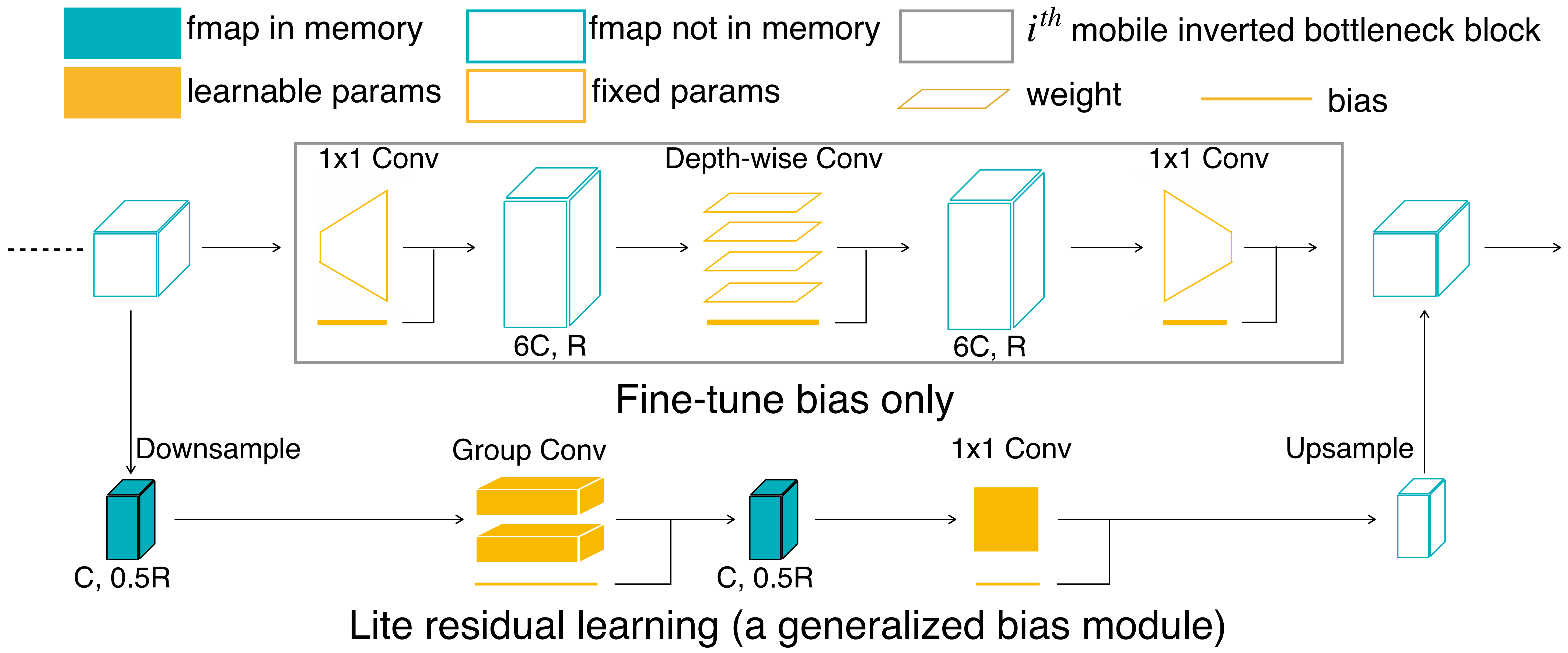


Freeze weights, only fine-tune biases

=> save 12x memory, but also hurt the accuracy

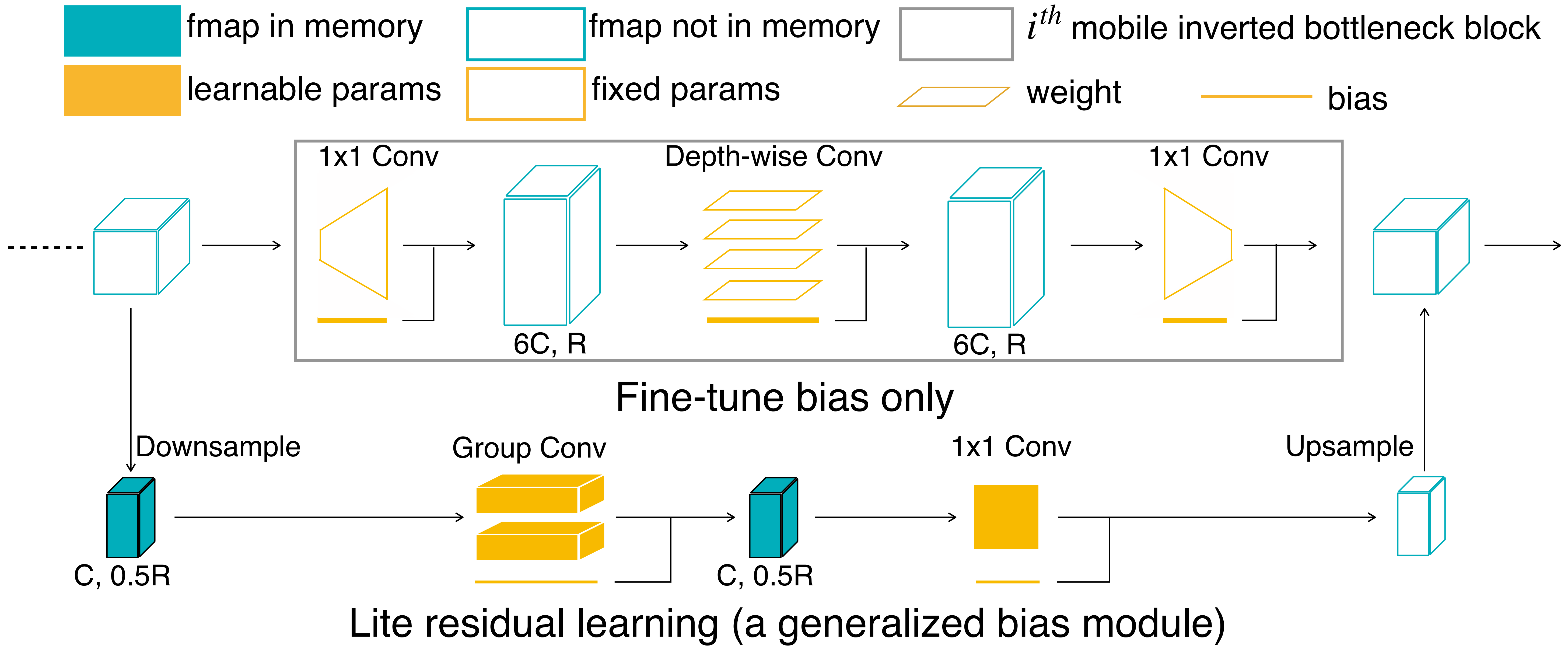
TinyTL, NeurIPS'20

TinyTL: Lite Residual Learning



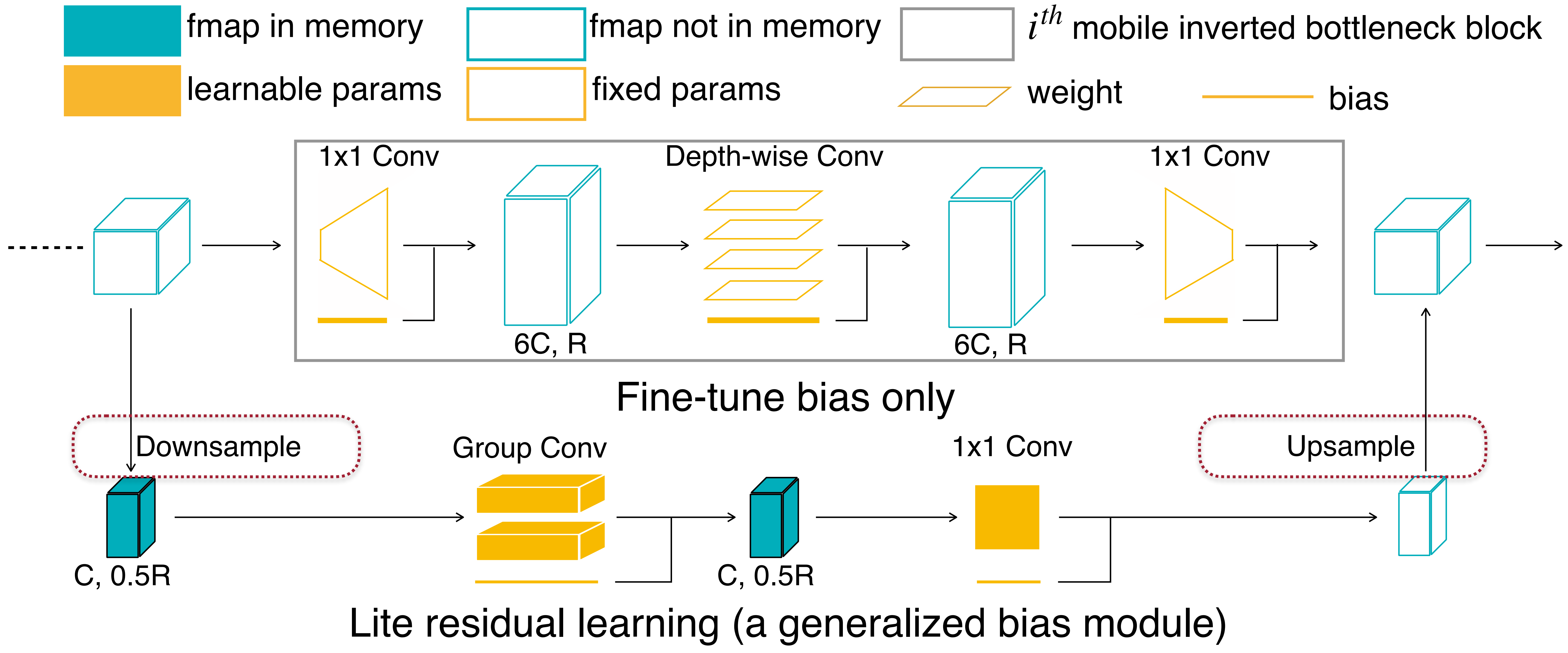
- Add lite residual modules to increase model capacity

TinyTL: Lite Residual Learning



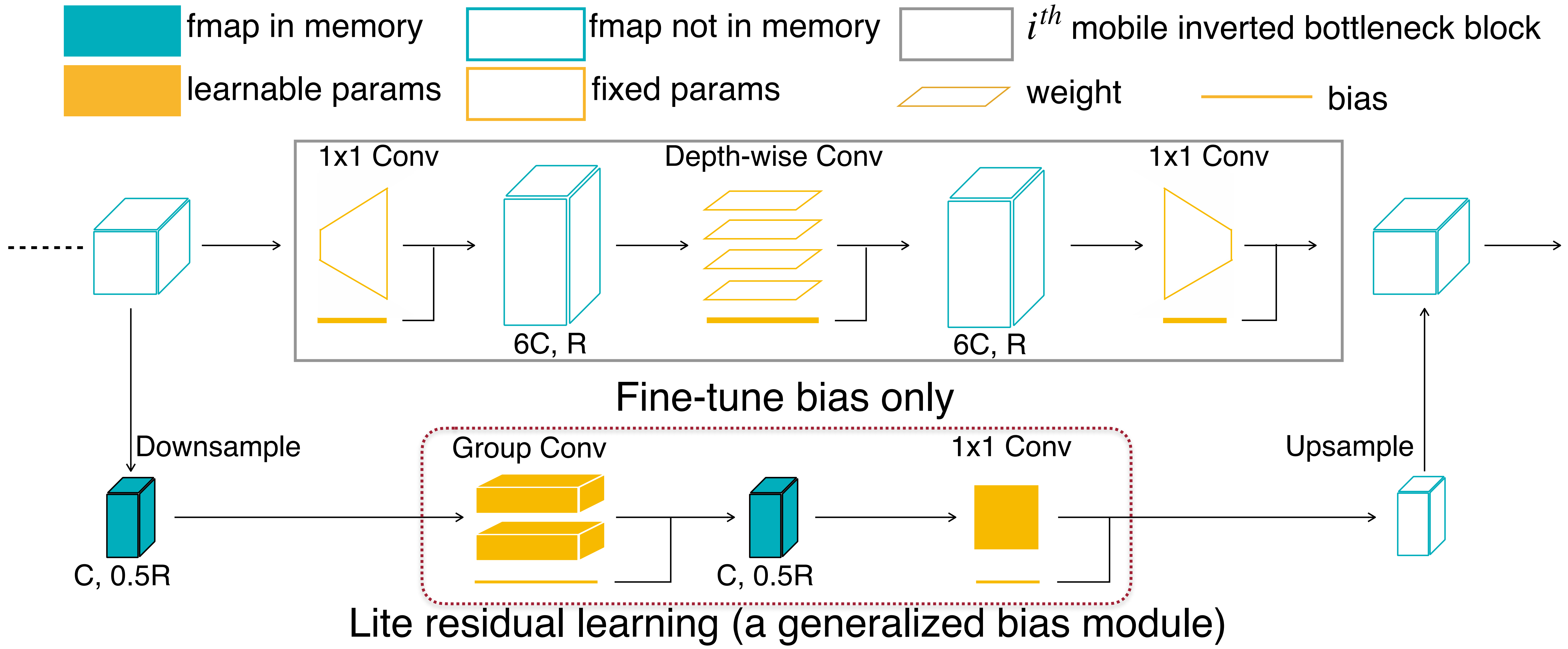
- Add lite residual modules to increase model capacity
- Key principle - keep activation size small

TinyTL: Lite Residual Learning



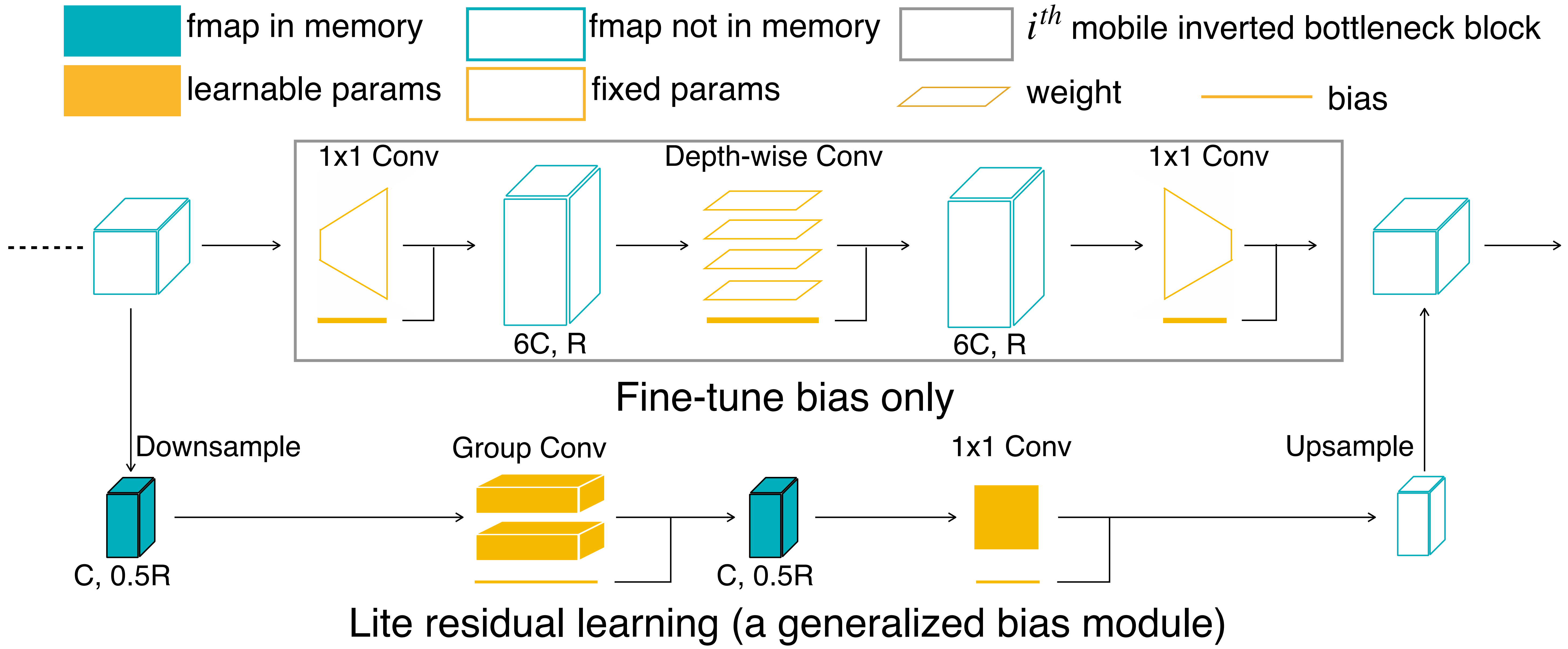
- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
    1. Reduce the resolution

TinyTL: Lite Residual Learning



- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
    1. Reduce the resolution
    2. Avoid inverted bottleneck

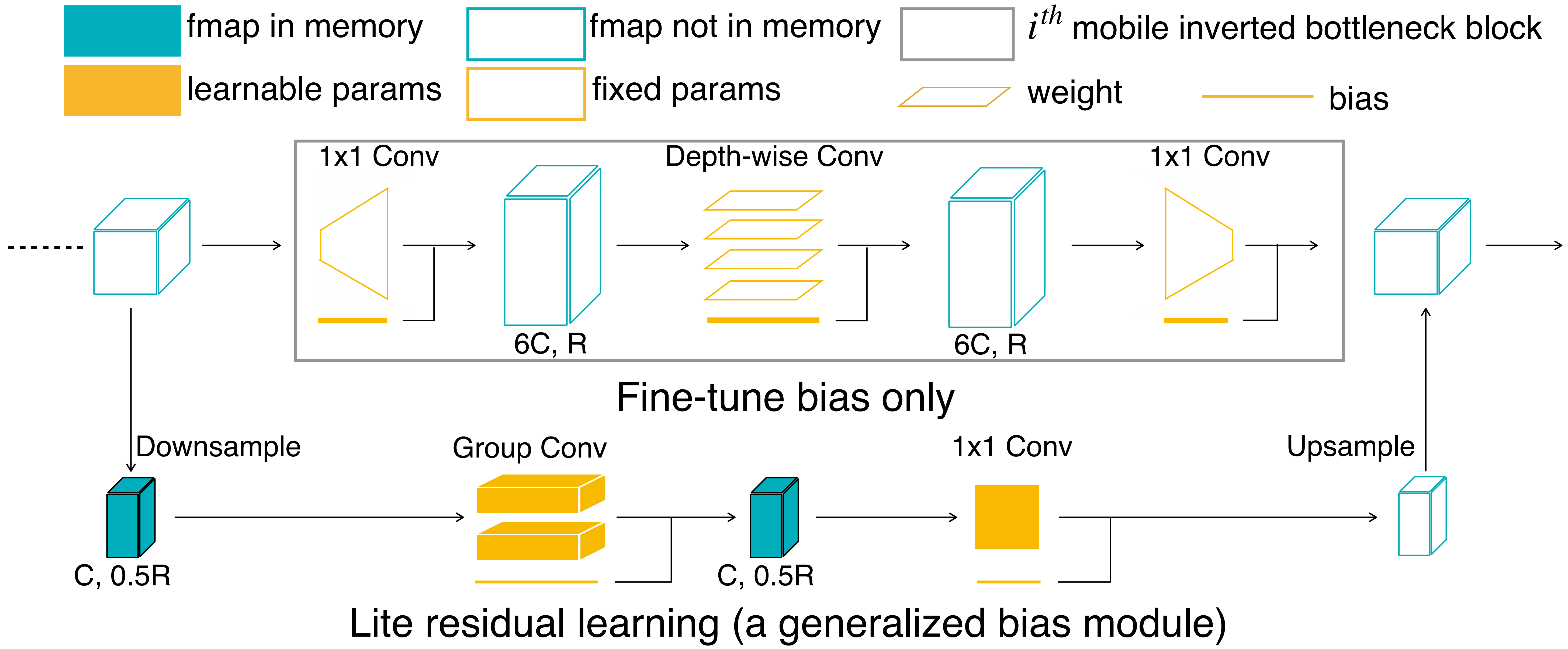
TinyTL: Lite Residual Learning

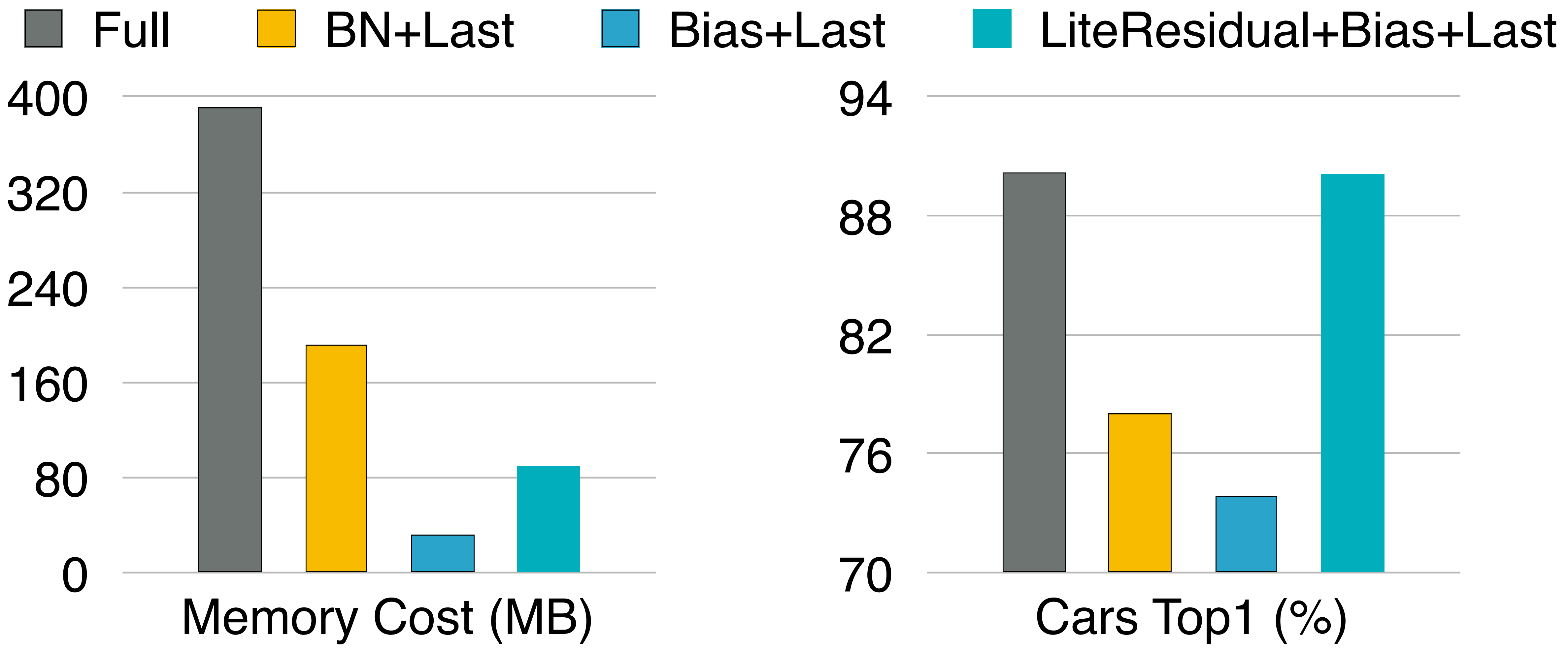


- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
    1. Reduce the resolution
    2. Avoid inverted bottleneck

(1/6 channel, 1/2 resolution, 2/3 depth => ~4% activation size)

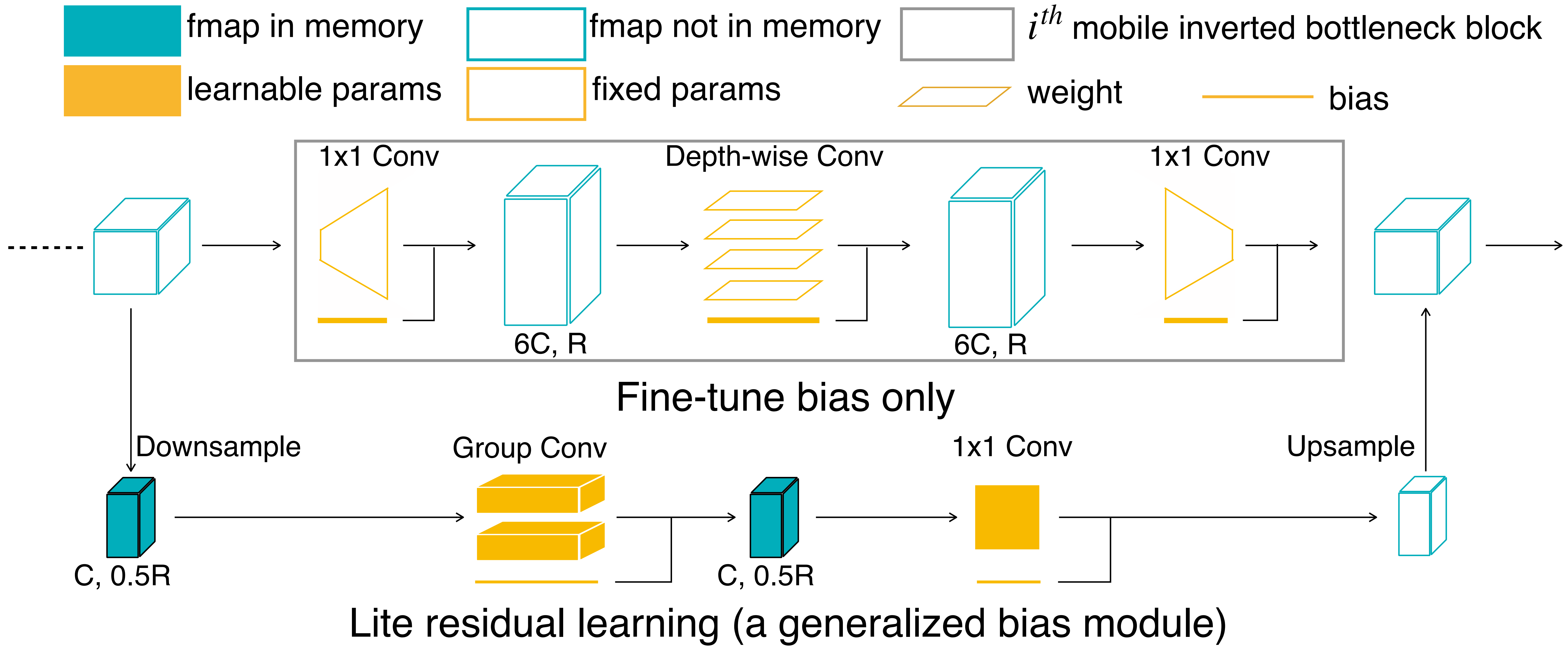
TinyTL: Lite Residual Learning





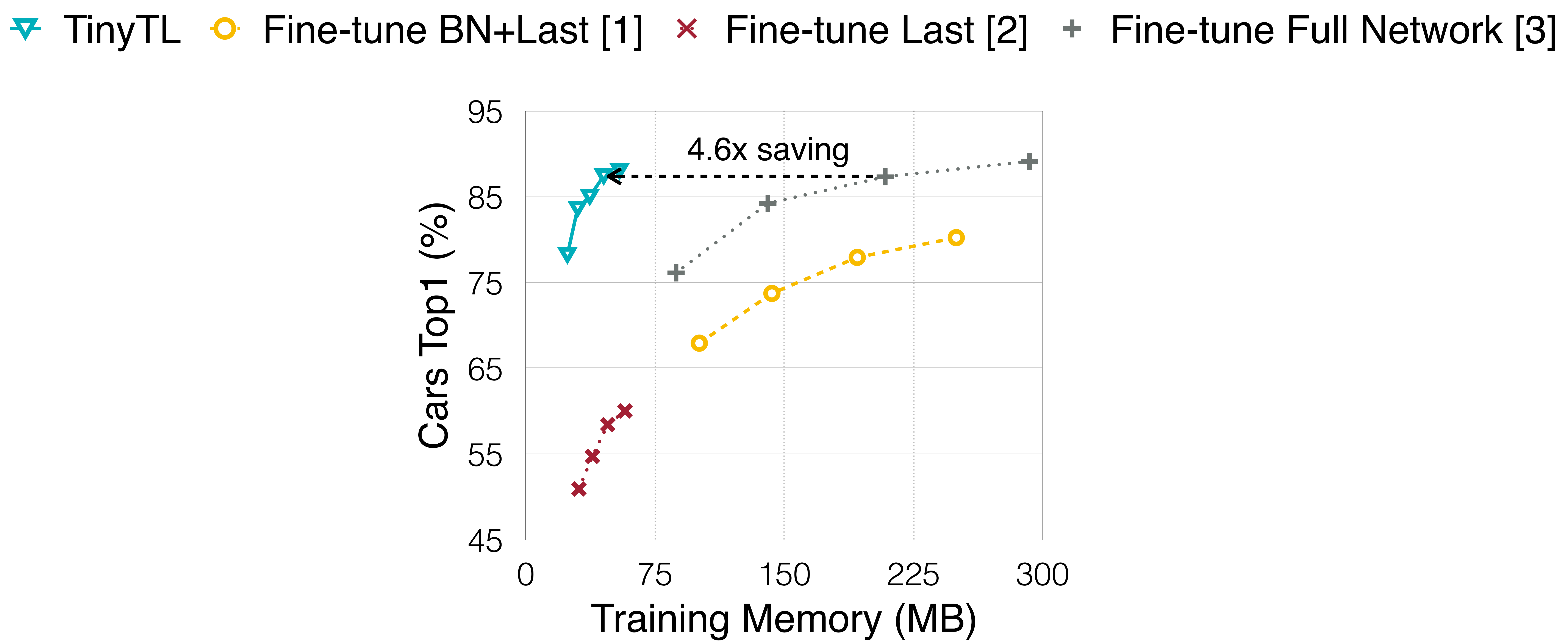
TinyTL, NeurIPS'20

Model Compression on Fixed Parameters



- Apply model compression (pruning, quantization) to reduce the parameter size for fixed parameters.

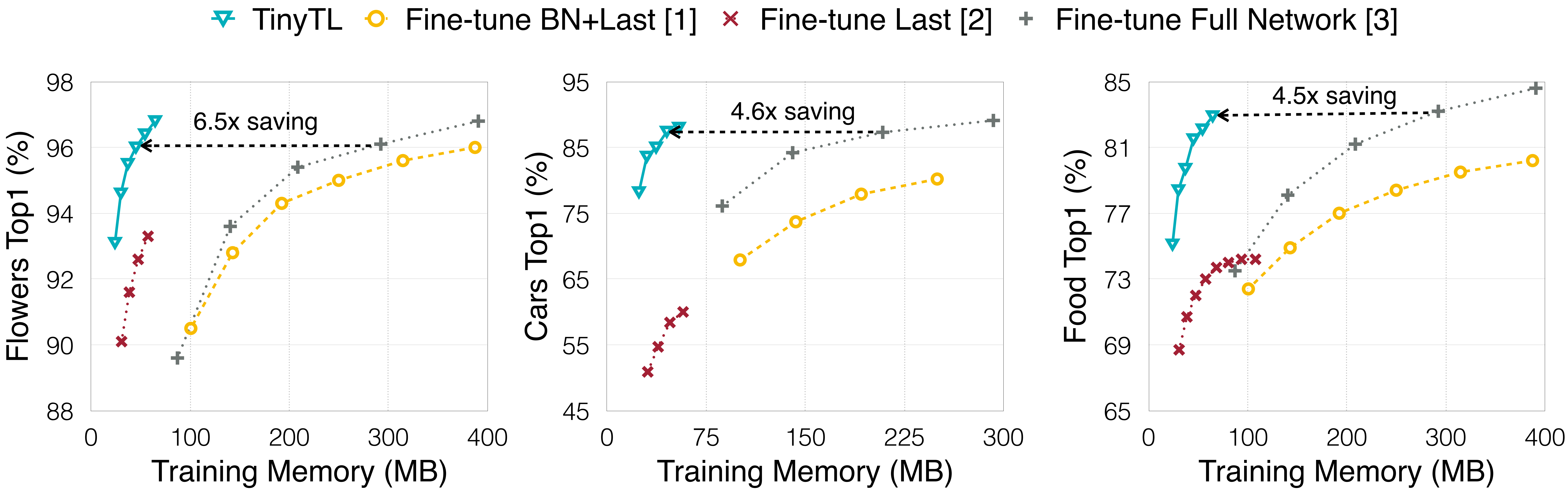
Memory Saving



- TinyTL provides **4.6x** memory saving **without accuracy loss**.

- [1] Chatfield, Ken, et al. "Return of the devil in the details: Delving deep into convolutional nets." *BMVC 2014.*
- [2] Mudrakarta, Pramod Kaushik, et al. "K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning." *ICLR 2019.*
- [3] Kornblith, Simon, Jonathon Shlens, and Quoc V. Le. "Do better imagenet models transfer better?." *CVPR* 2019.

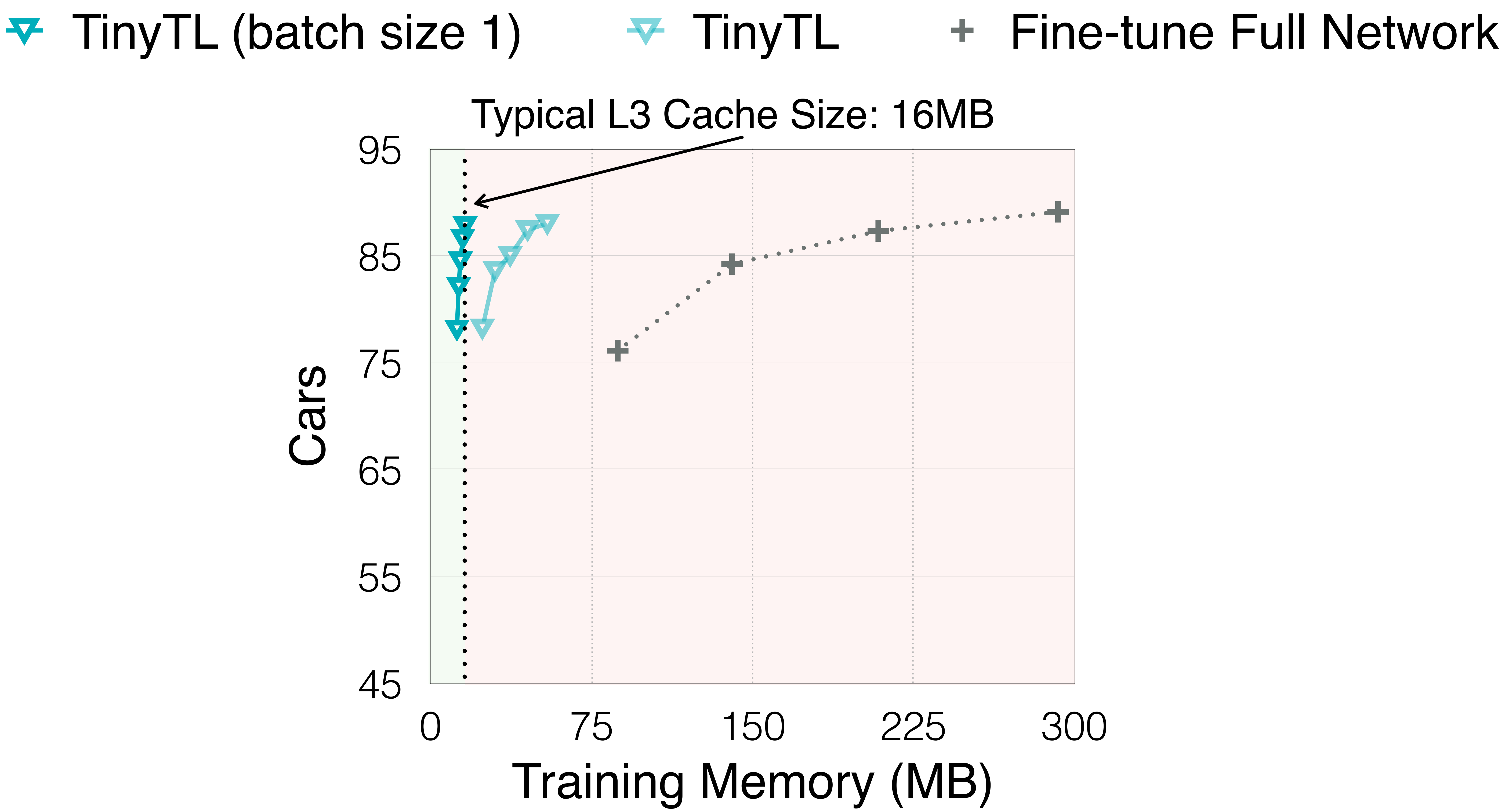
Memory Saving



- On different datasets, TinyTL provides up to **6.5x** memory saving **without accuracy loss**.

- [1] Chatfield, Ken, et al. "Return of the devil in the details: Delving deep into convolutional nets." *BMVC 2014.*
- [2] Mudrakarta, Pramod Kaushik, et al. "K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning." *ICLR 2019.*
- [3] Kornblith, Simon, Jonathon Shlens, and Quoc V. Le. "Do better imagenet models transfer better?." *CVPR* 2019.

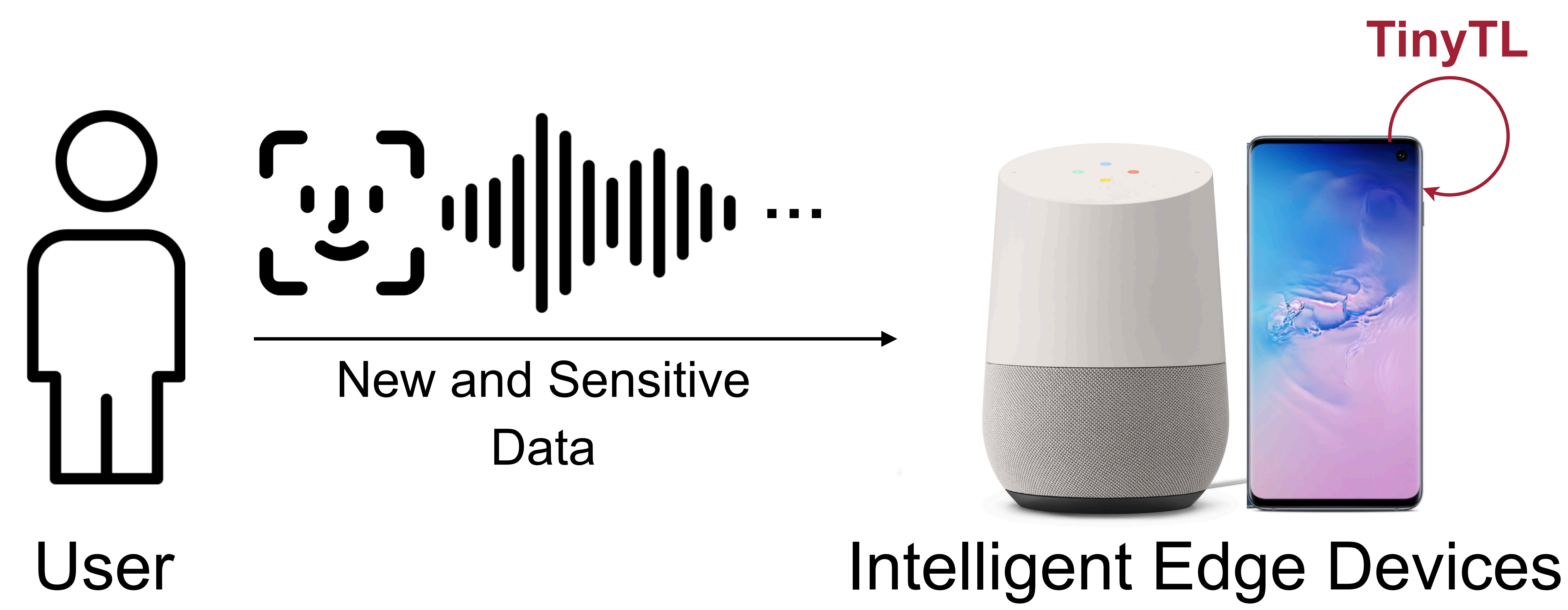
TinyTL enables in-cache training

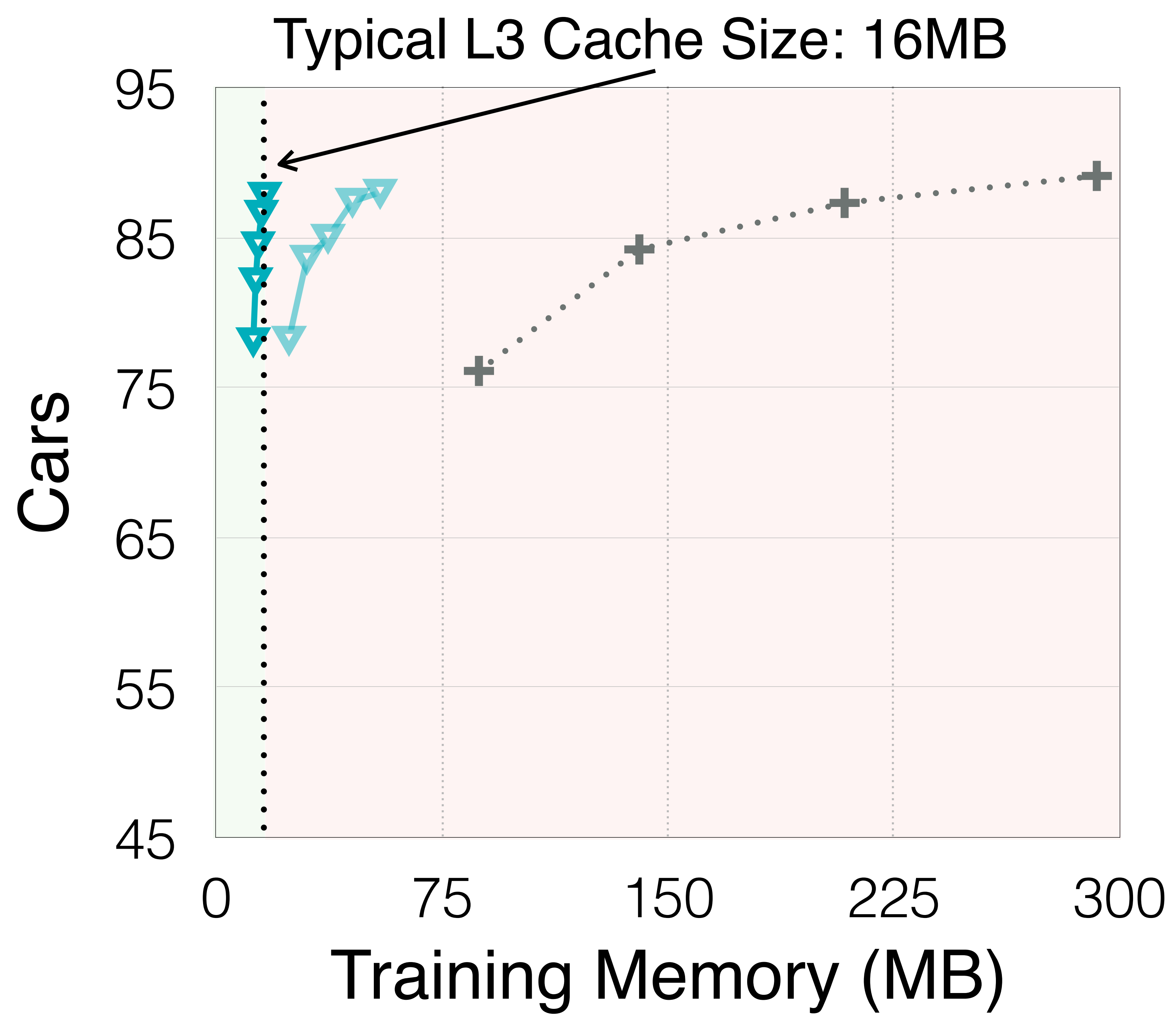


- TinyTL (tiny transfer learning) supports batch 1 training by **group normalization**.
- Together with the lite residual model, it further reduces the training memory cost to 16MB (fits L3 cache), enabling fitting the training process into cache, which is much more energy-efficient than training on DRAM.

TinyTL, NeurIPS'20

TinyTL: Reduce Memory, not Parameters for Efficient On-Device Learning





Project Page: **http://tinyml.mit.edu**