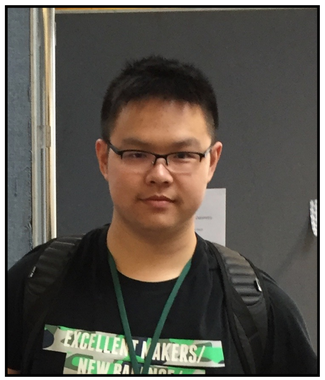
TinyTL: Reduce Memory, not Parameters for Efficient On-Device Learning

Han Cai[1]

Chuang Gan[2]



Ligeng Zhu[1]

Song Han[1]

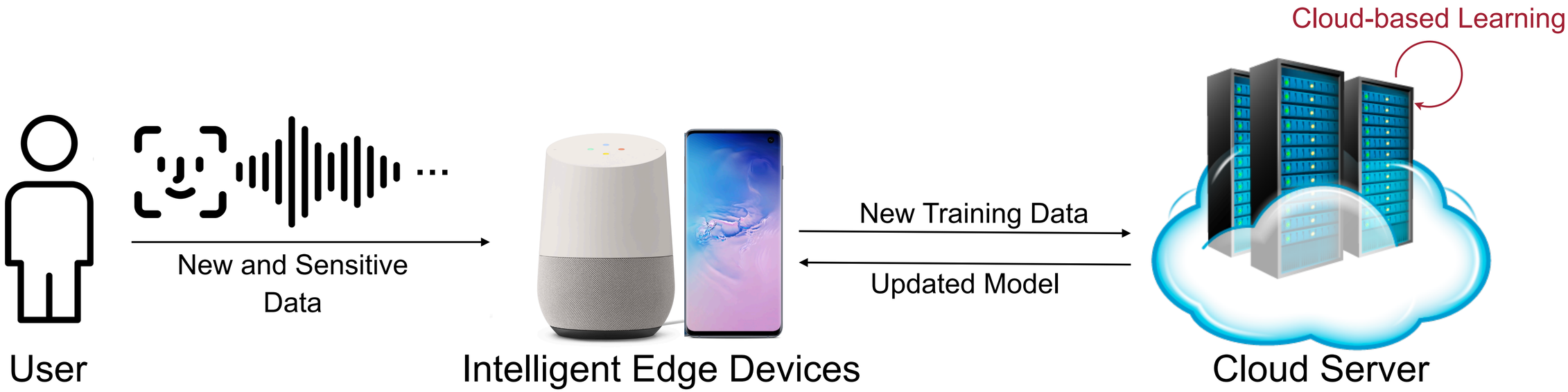
[1]MIT [2]MIT-IBM Watson AI Lab

NeurIPS 2020

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

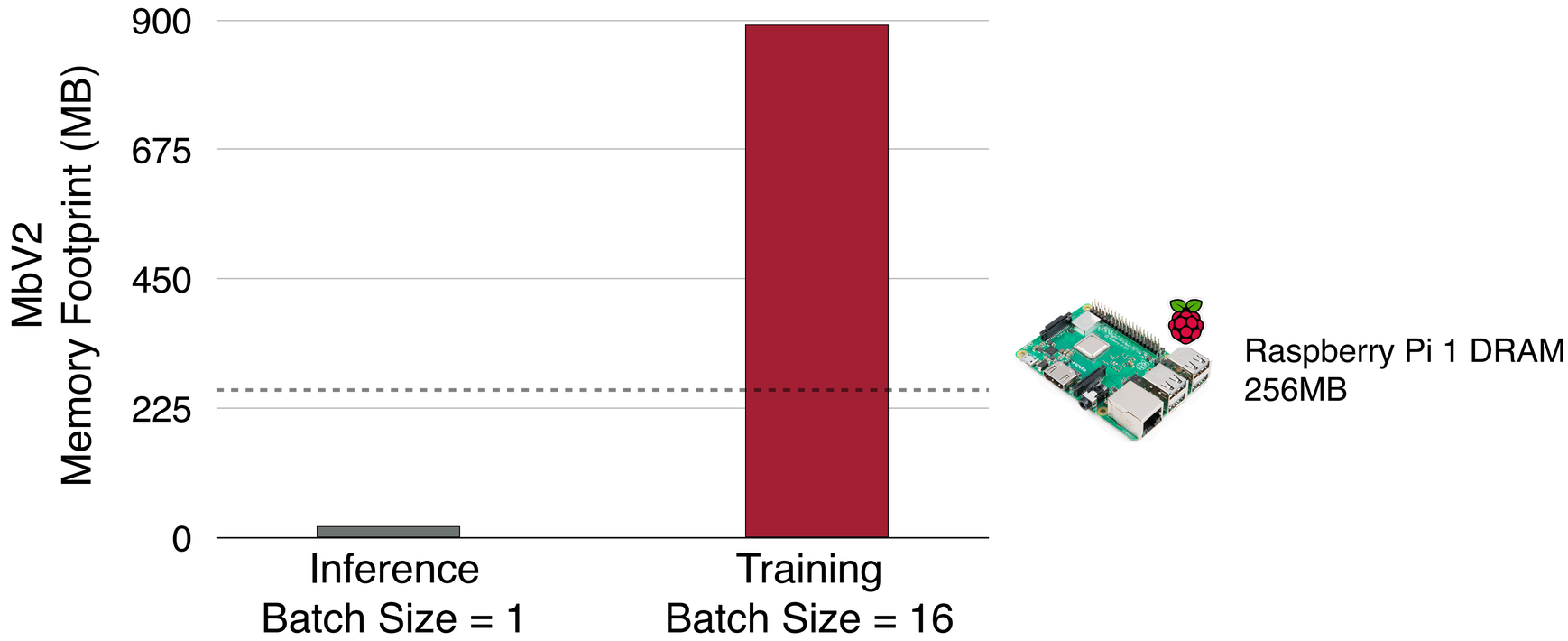


- Customization: AI systems need to continually adapt to new data collected from the sensors.

On-device Learning

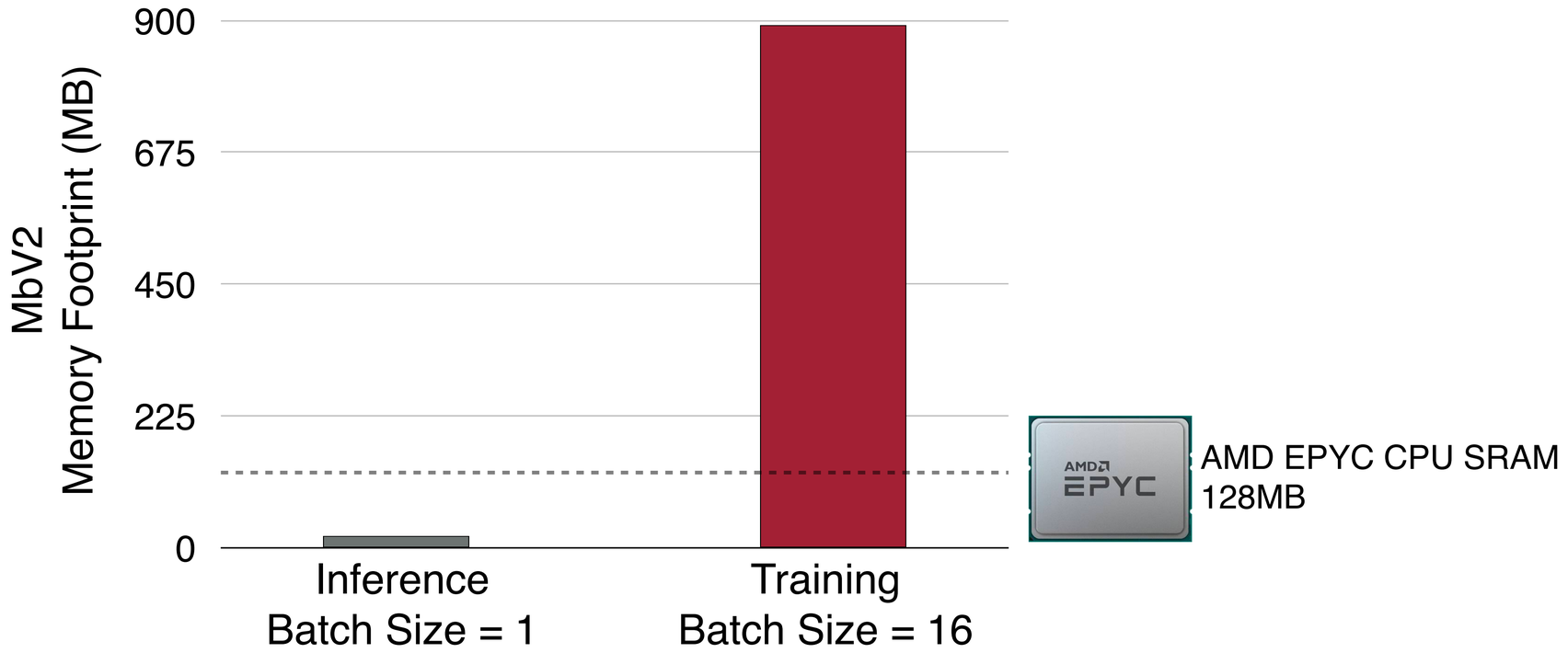
- Customization: AI systems need to continually adapt to new data collected from the sensors.
- Security: Data cannot leave devices because of security and regularization.

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.

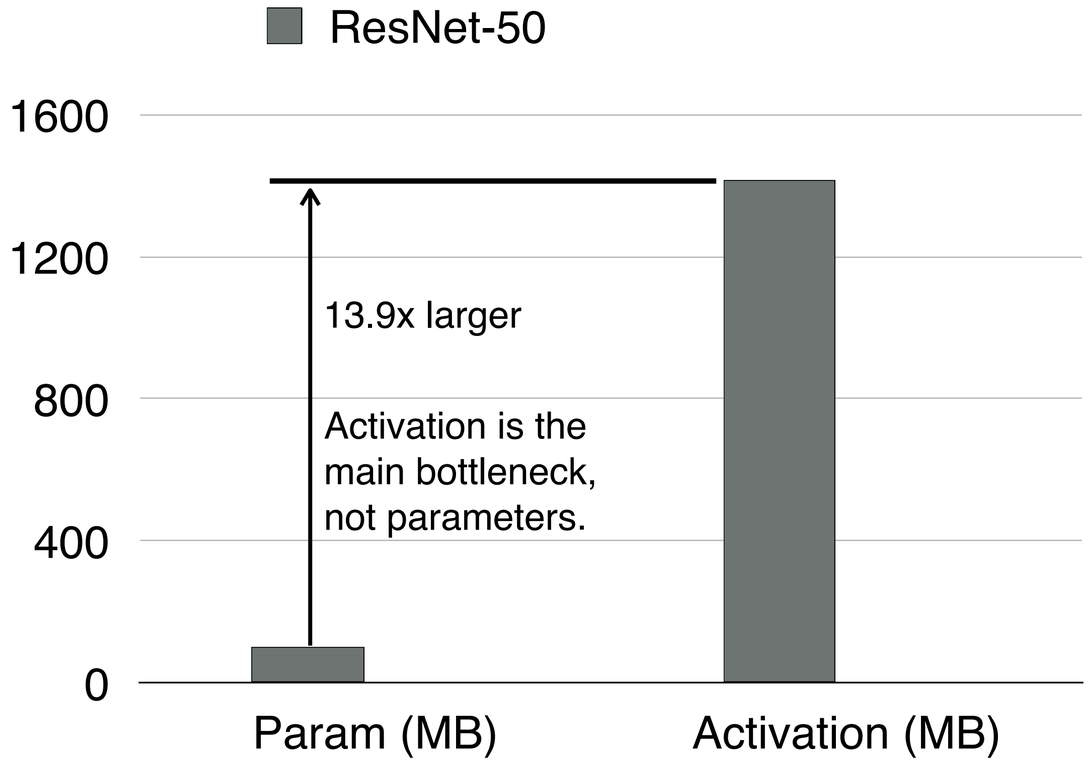
Training Memory is much Larger than Inference



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

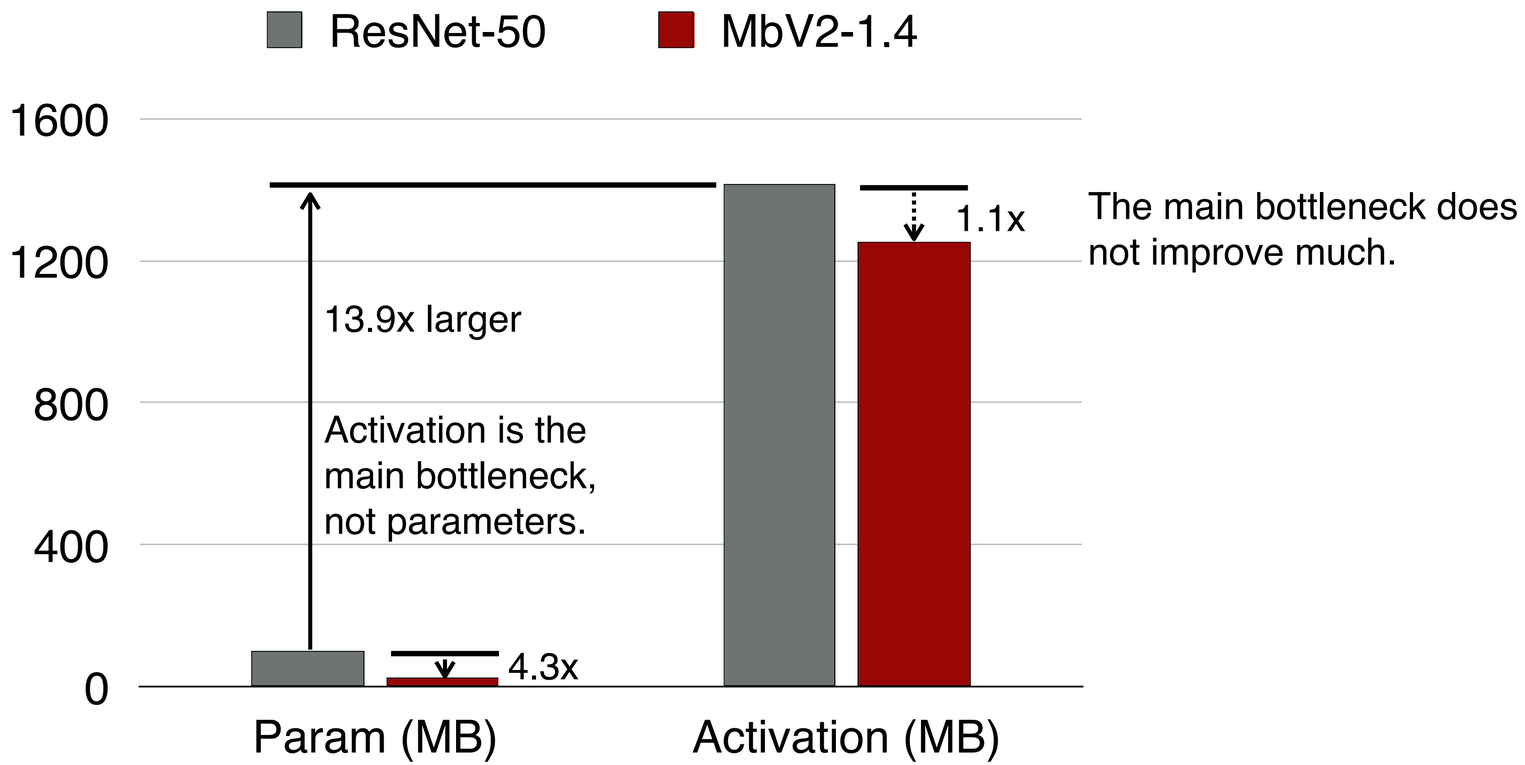
- Edge devices are energy-constrained. Failing to fit the training process into the energy-efficient on-chip SRAM will significantly increase the energy cost.

Activation is the Memory Bottleneck, not Parameters



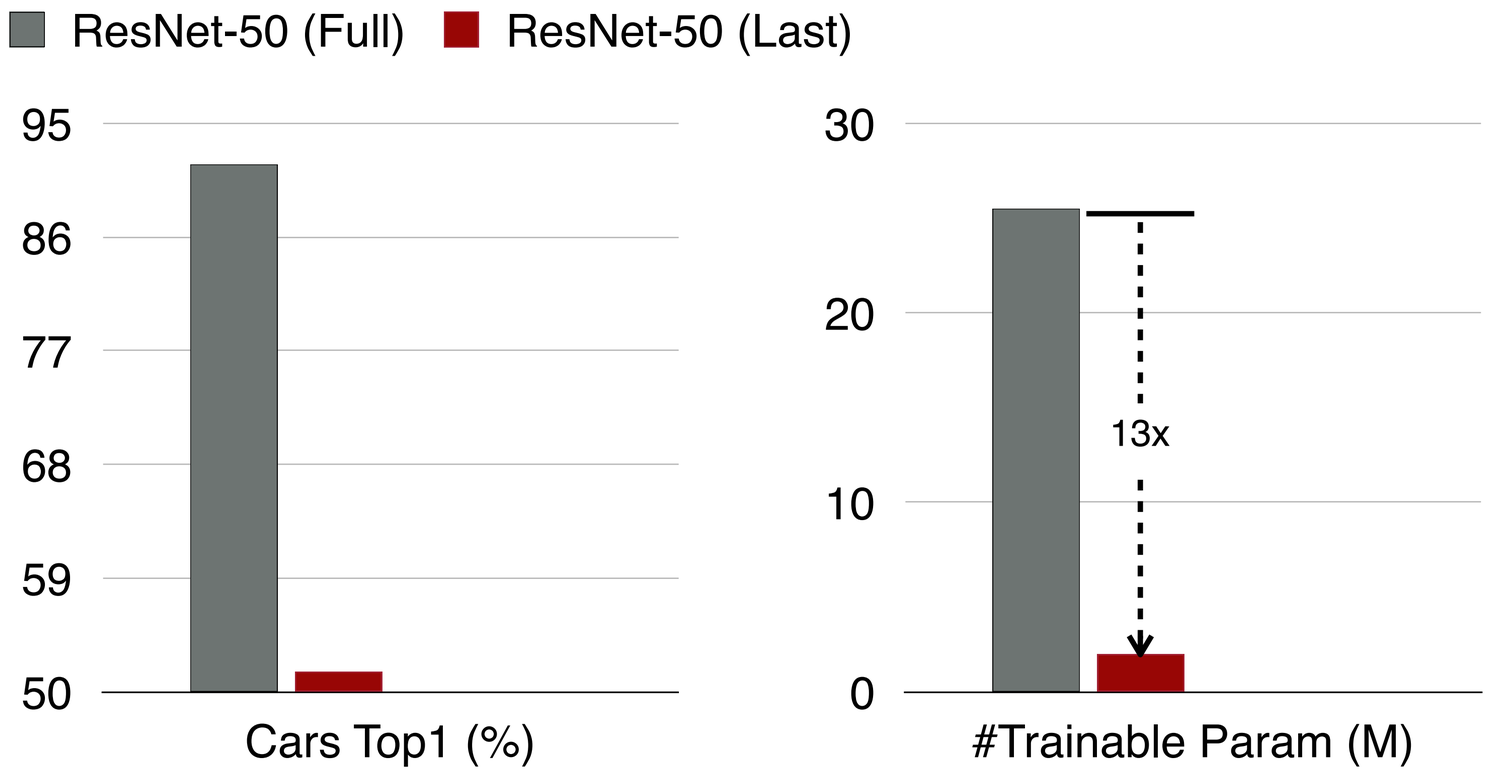
- Activation is the main bottleneck for on-device learning, not parameters.

Activation is the Memory Bottleneck, not Parameters



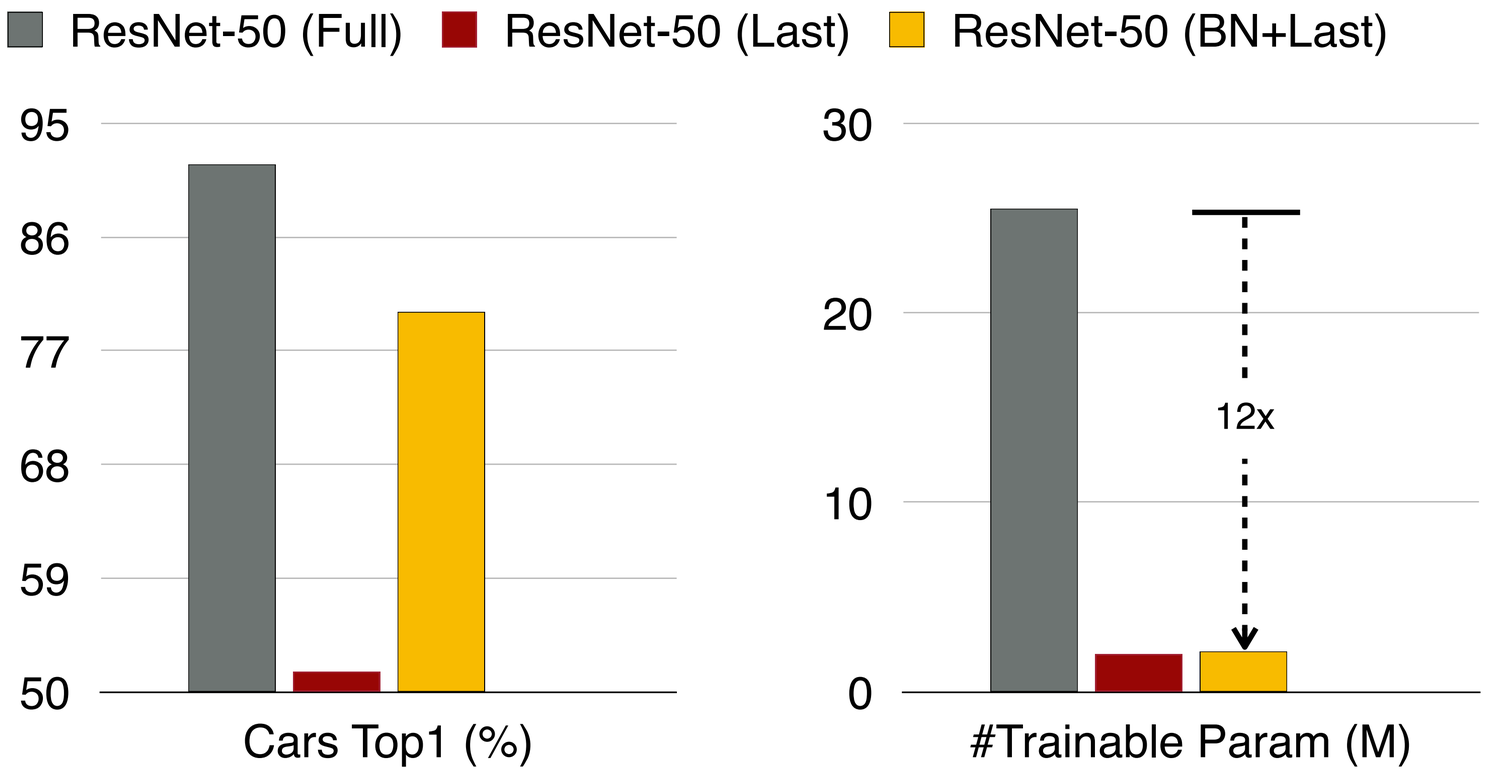
- Activation is the main bottleneck for on-device learning, not parameters.
- Previous methods focus on reducing the number of parameters or FLOPs, while the main bottleneck does not improve much.

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.

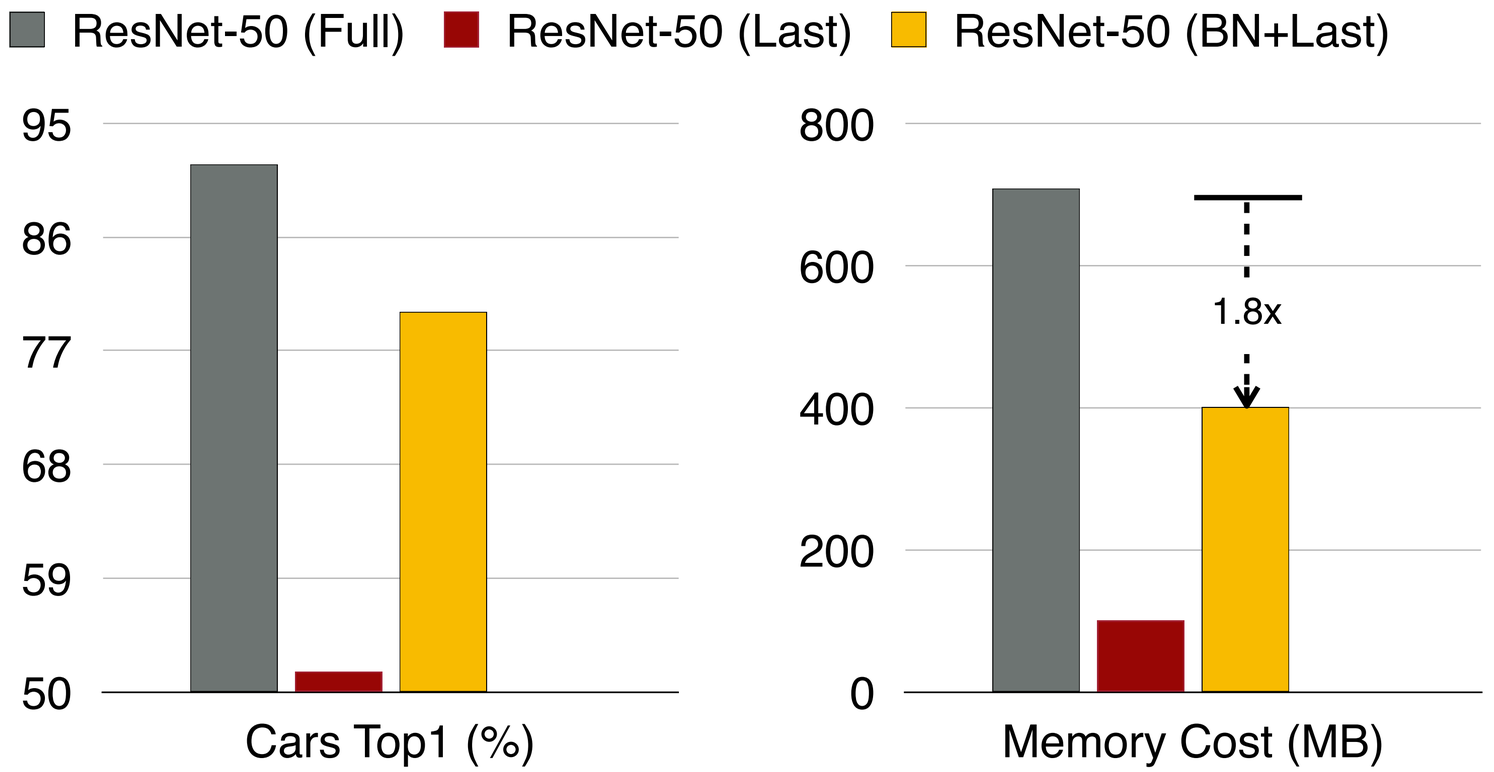
Related Work: Parameter-Efficient Transfer Learning



- 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.

- Mudrakarta, Pramod Kaushik, et al. "K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning." *ICLR 2019.*

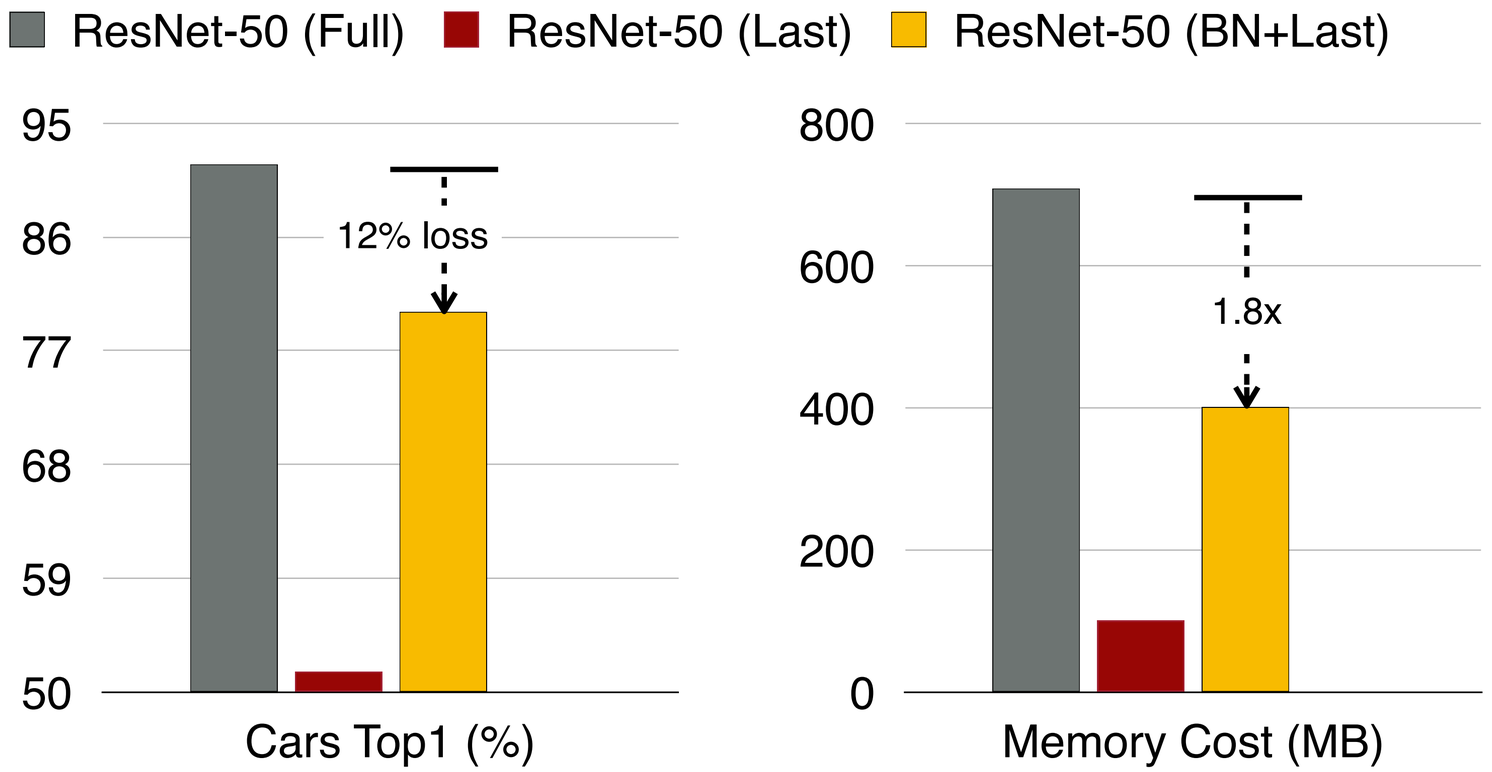
Related Work: Parameter-Efficient Transfer Learning



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.**

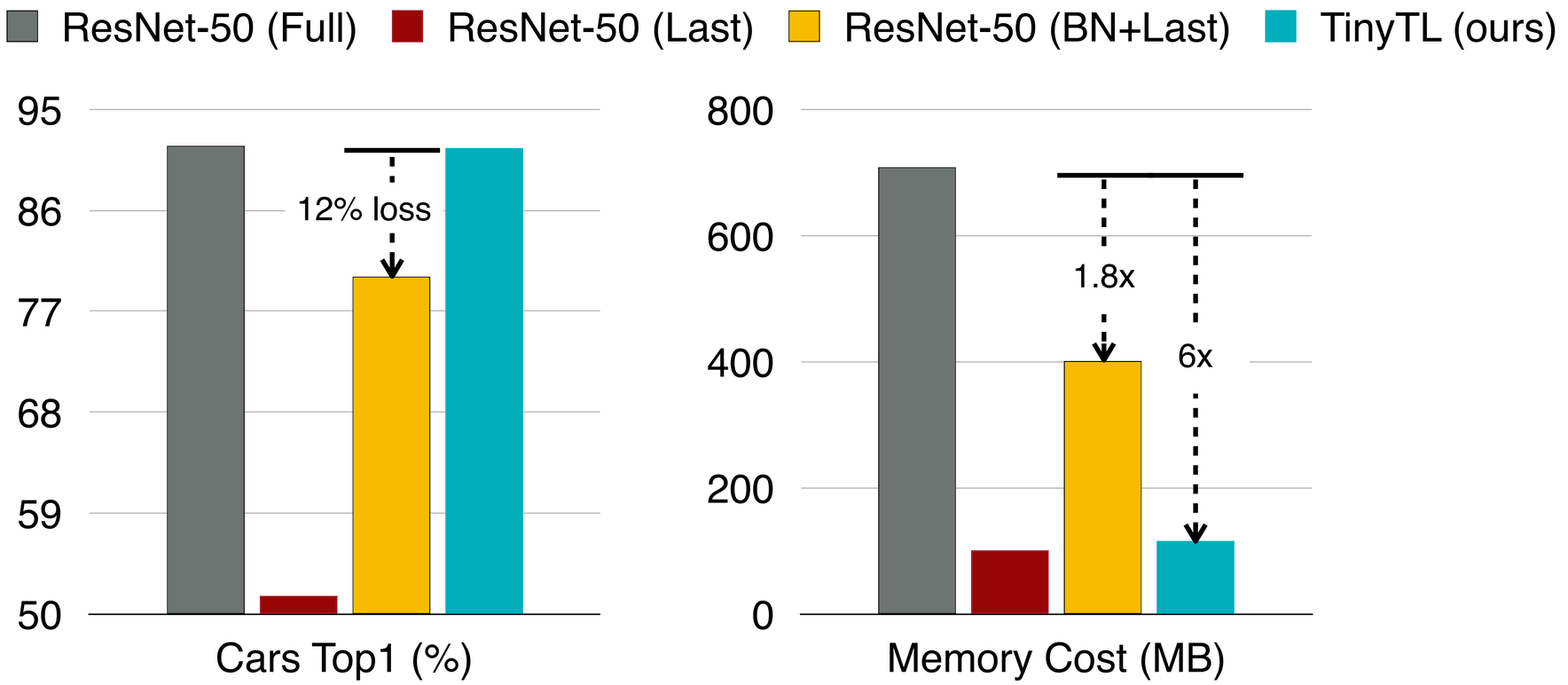
Related Work: Parameter-Efficient Transfer Learning



Parameter-efficiency does not directly translate to memory-efficiency

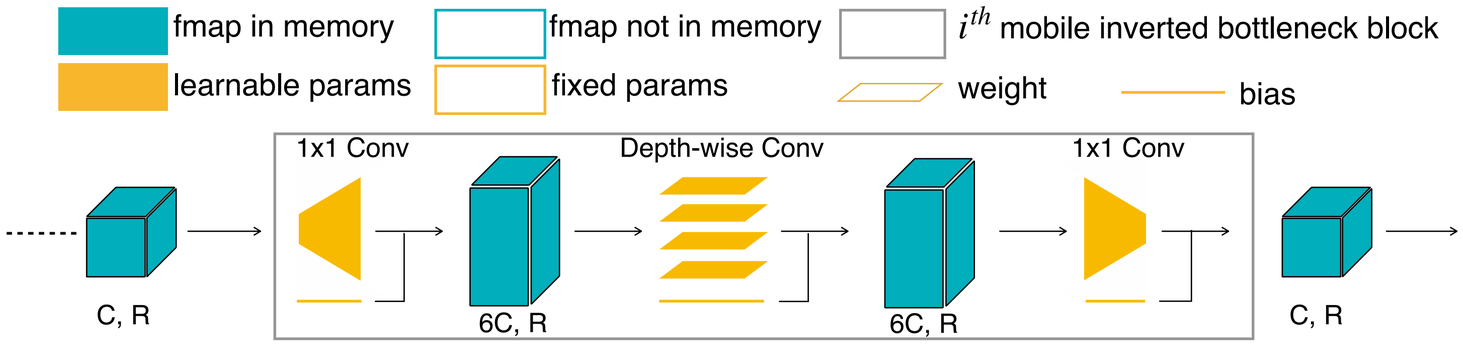
- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the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



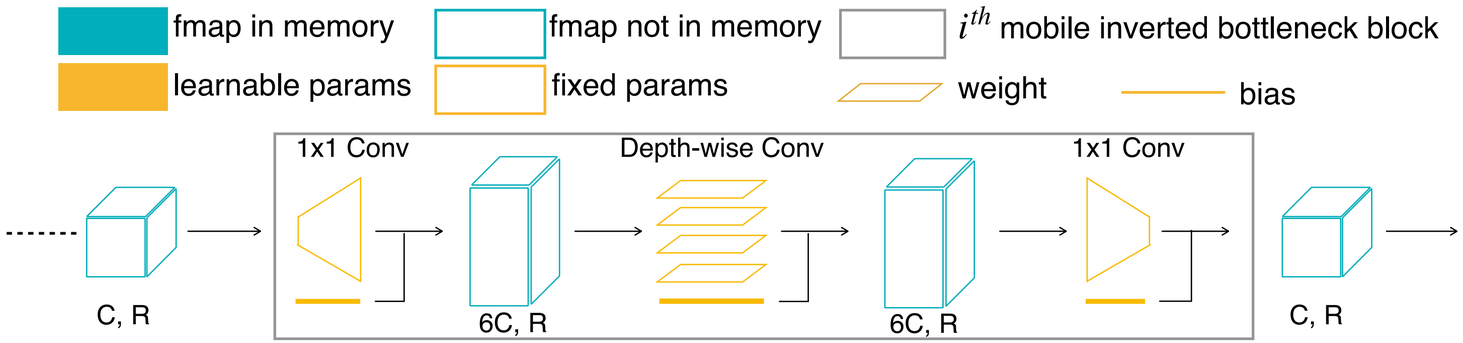
Fine-tune the full network (Conventional)

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

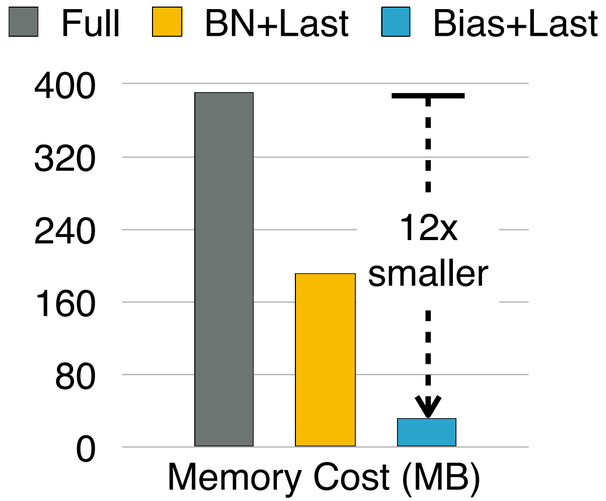
Backward: $\frac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \frac{\partial L}{\partial \mathbf{a}_{i+1}}, \qquad \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial \mathbf{a}_{i+1}} = \frac{\partial L}{\partial \mathbf{a}_{i+2}} \mathbf{W}_{i+1}^T$

- Updating weights requires storing intermediate activations
- Updating biases does not

TinyTL: Fine-tune Bias Only



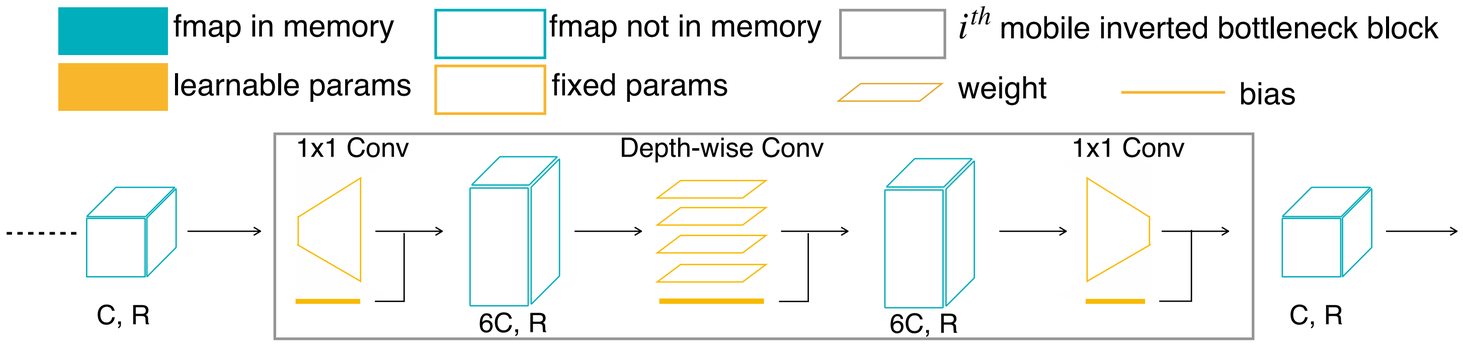
Fine-tune bias only



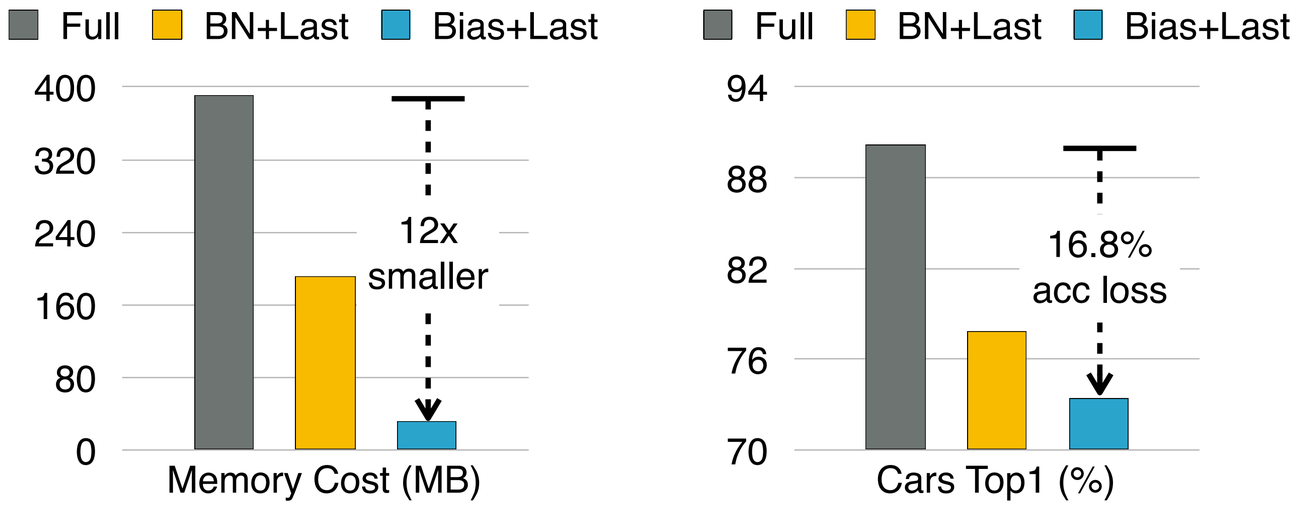
Freeze weights, only fine-tune biases

=> save 12x memory

TinyTL: Fine-tune Bias Only



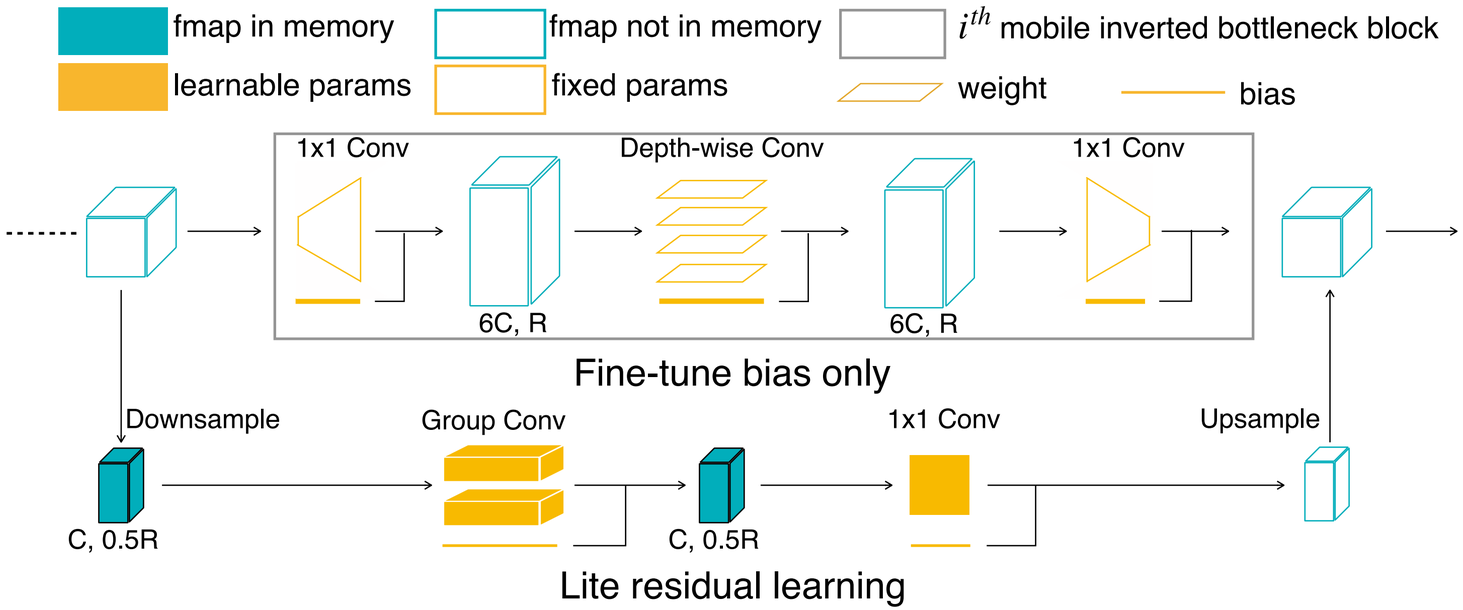
Fine-tune bias only



Freeze weights, only fine-tune biases

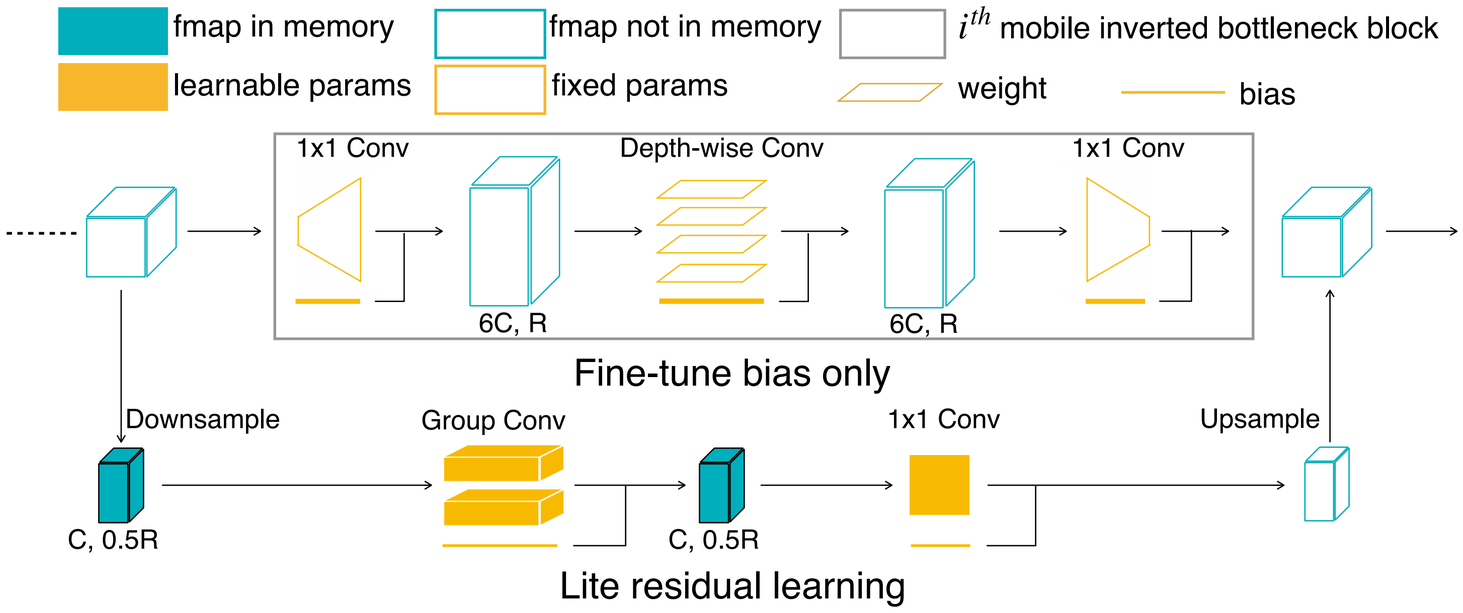
=> save 12x memory, but also hurt the accuracy

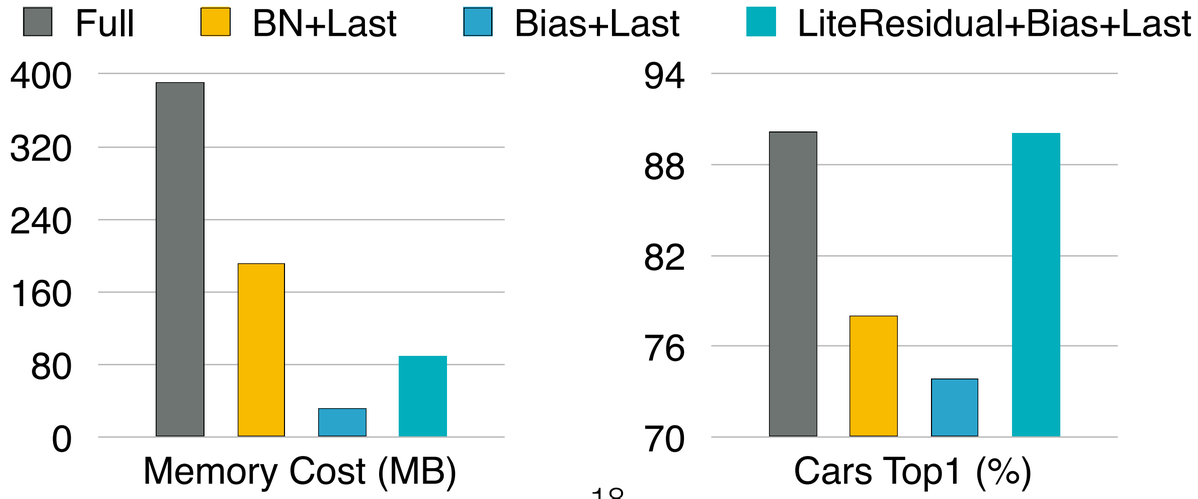
TinyTL: Lite Residual Learning



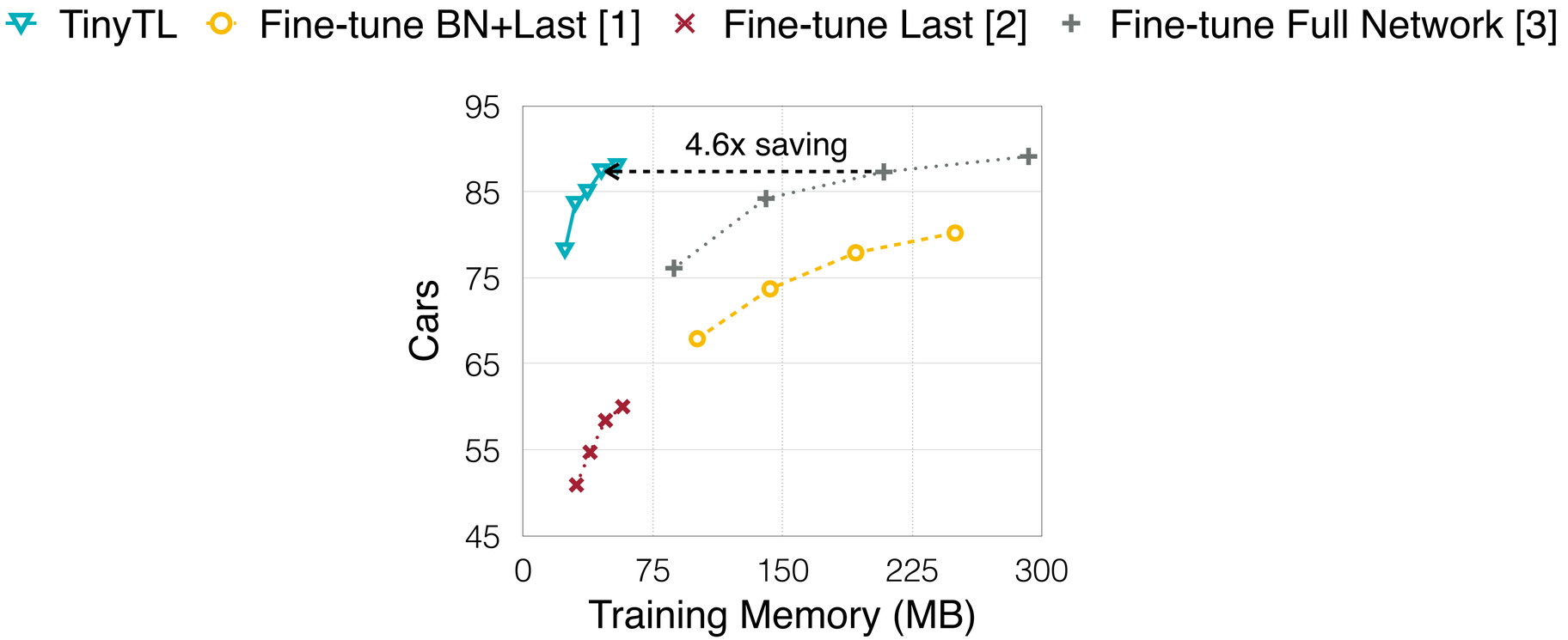
- Add lite residual modules (only 4% memory overhead) to increase model capacity
  (1/6 channel, 1/2 resolution, 2/3 depth => ~4%)

TinyTL: Lite Residual Learning





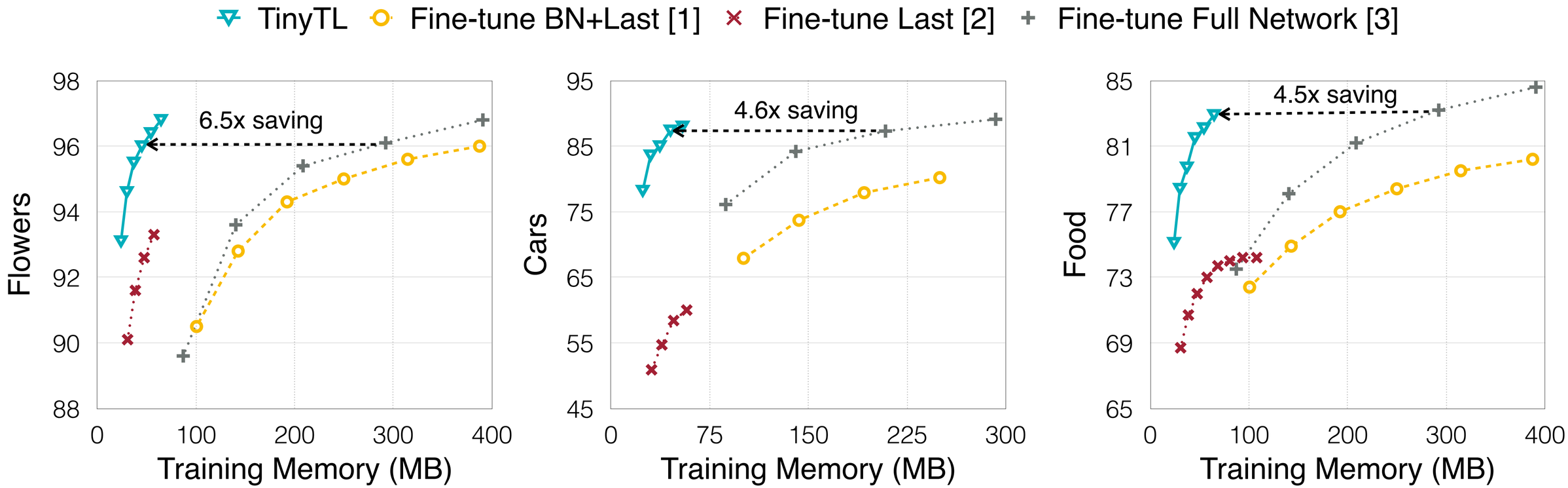
TinyTL: Same Accuracy, 4.6x Memory Saving



- Using the same pre-trained model (ProxylessNAS-Mobile), 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.

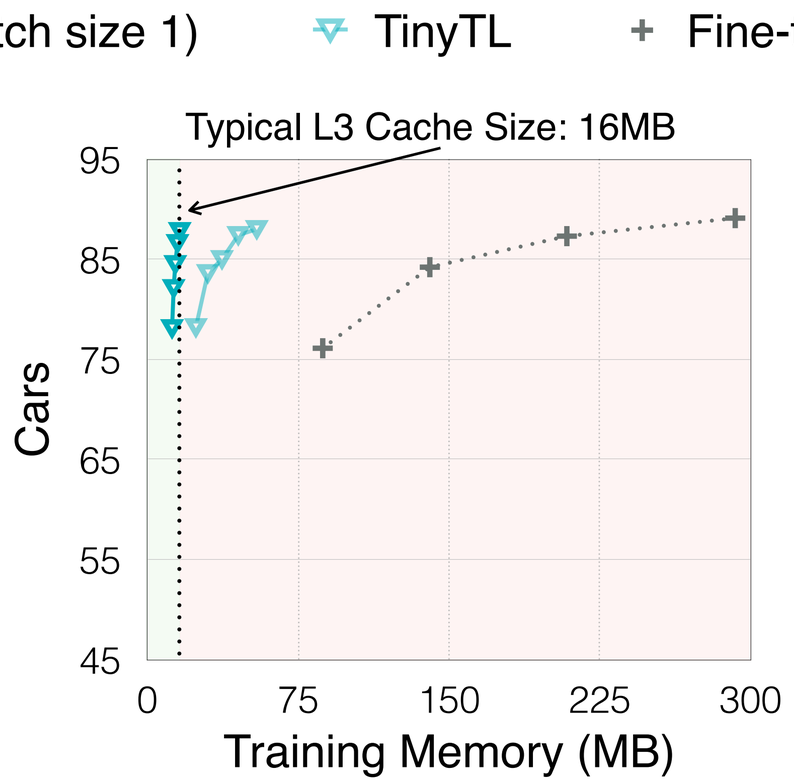
TinyTL: Same Accuracy, Up to 6.5x Memory Saving



- Using the same pre-trained model (ProxylessNAS-Mobile), TinyTL provides up to **6.5x** memory saving **without accuracy loss**.

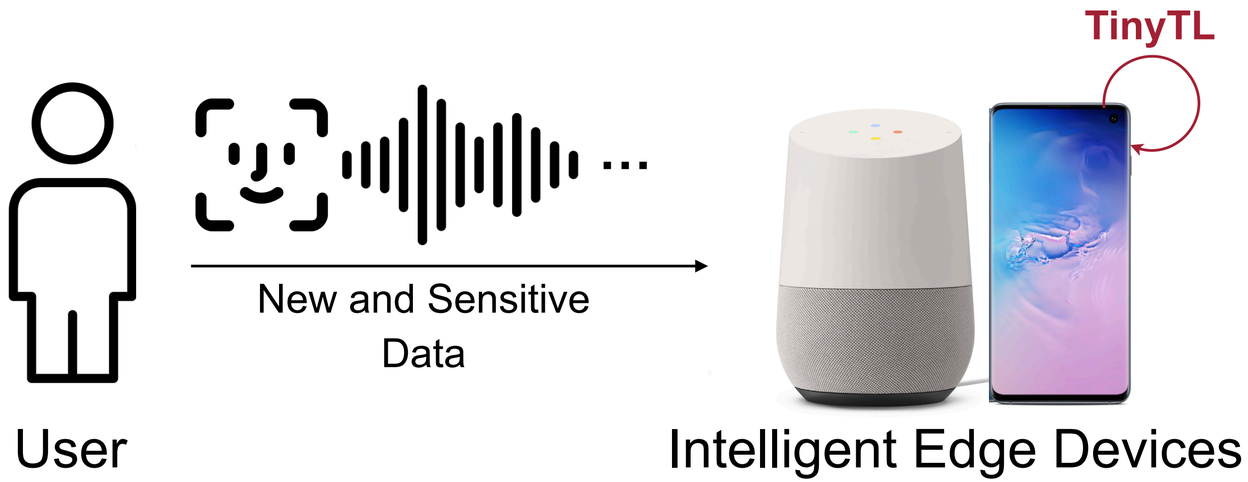
- [1] Chatfield, Ken, et al. "Return of the devil in the details: Delving deep into convolutional nets." *BMVC 2014*.
- [2] Mudrakarta, Pramod Kaushik, et al. "K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning." *ICLR 2019.*
- [3] Kornblith, Simon, Jonathon Shlens, and Quoc V.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

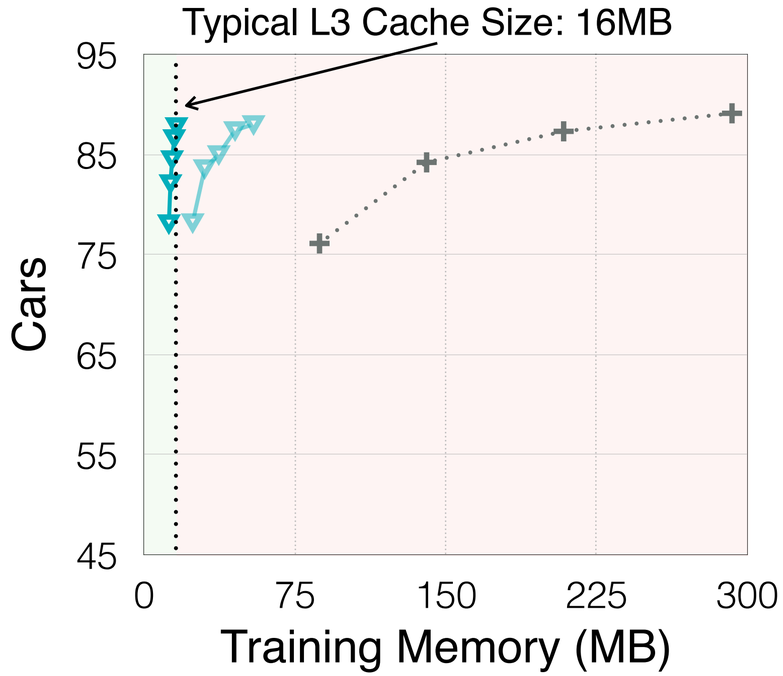
TinyTL: Fit the Training Process Into Cache



- TinyTL supports training with batch size 1.
- It further reduces the training memory cost to 16MB (typical L3 cache size), making it easier to train on the cache, which is much more energy-efficient than training on DRAM.

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





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