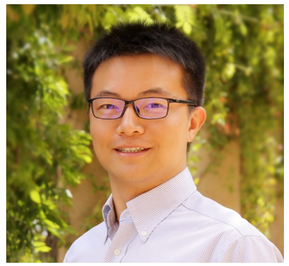
MIT HAN LAB

Hardware, AI, and Neural-nets
open source, co-design
http://github.com/mit-han-lab

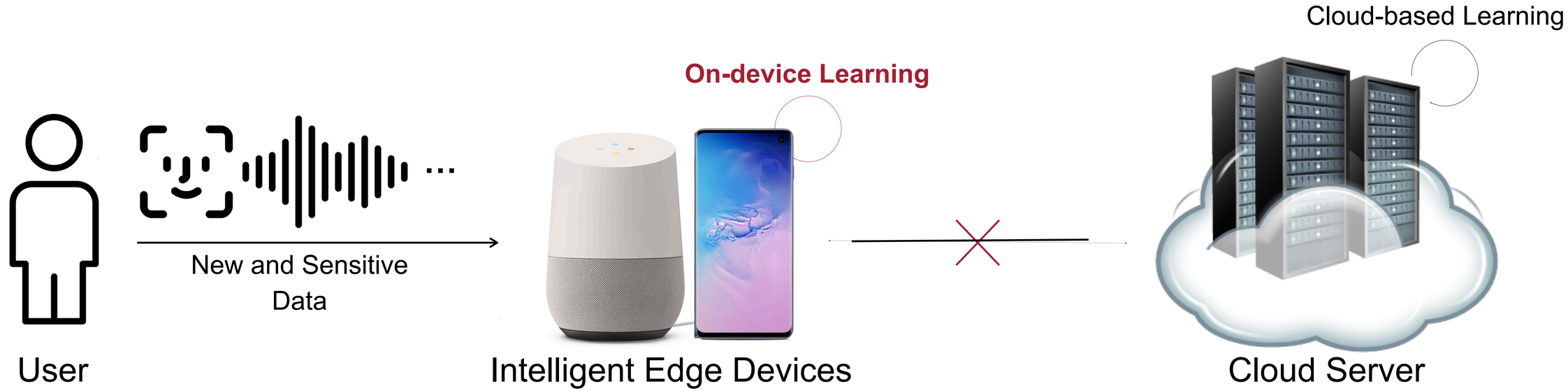
On-Device Training Under 256KB Memory



Song Han
MIT, OmniML
songhan.mit.edu
mcunet.mit.edu

Can we Learn on the Edge?

AI systems need to continually adapt to new data collected from the sensors
Not only inference, but also run back-propagation on edge devices

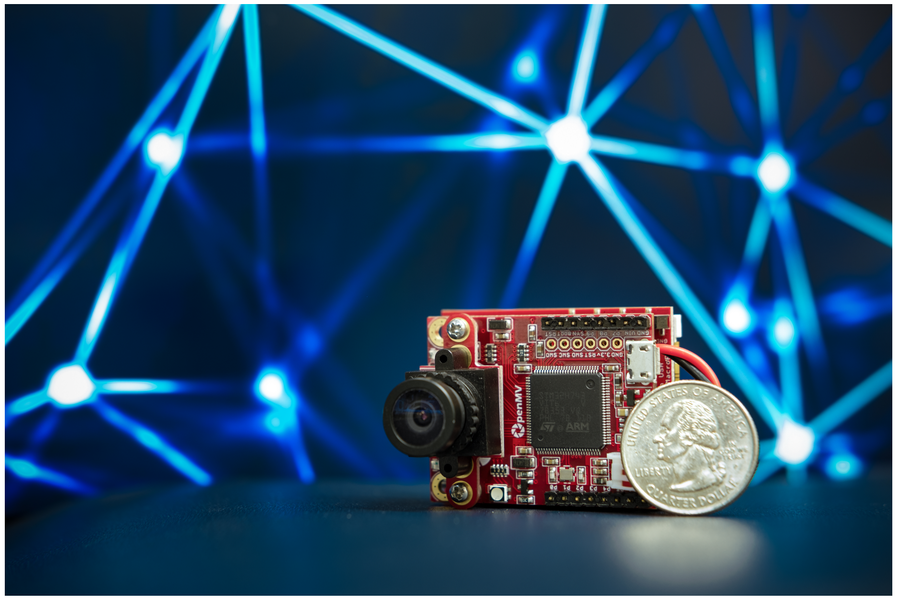


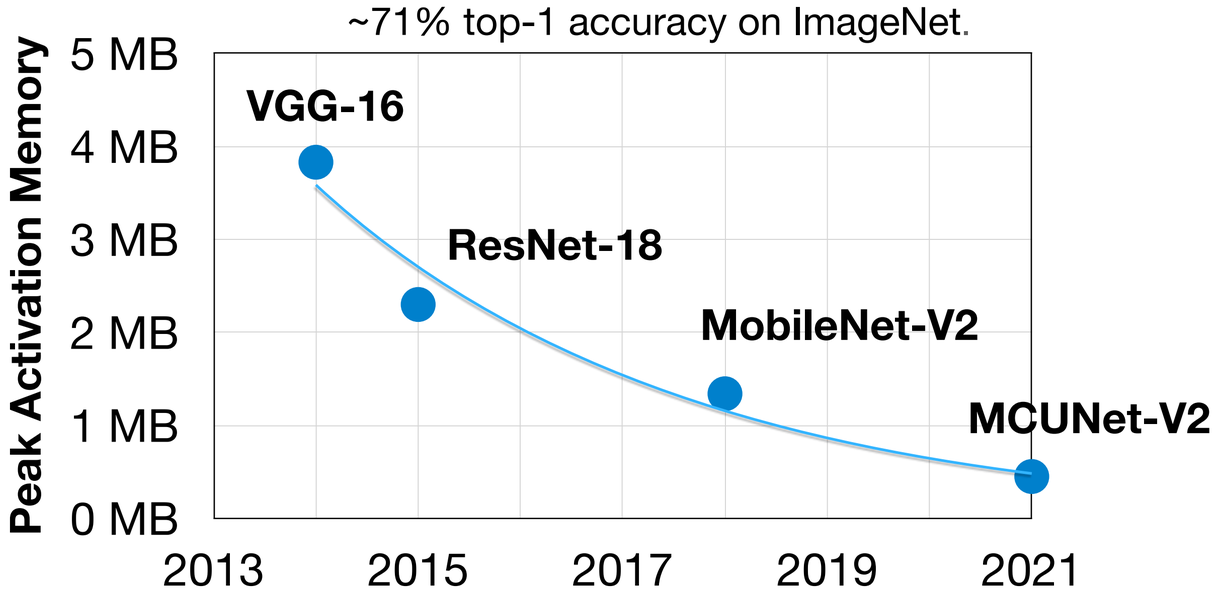
- On-device learning: **better privacy, lower cost, customization, life-long learning**
- Training is more **expensive** than inference, hard to fit edge hardware (limited memory)

Background work: MCUNet: Bring AI to IoT Devices

Unlock ultra low-power AIoT Applications

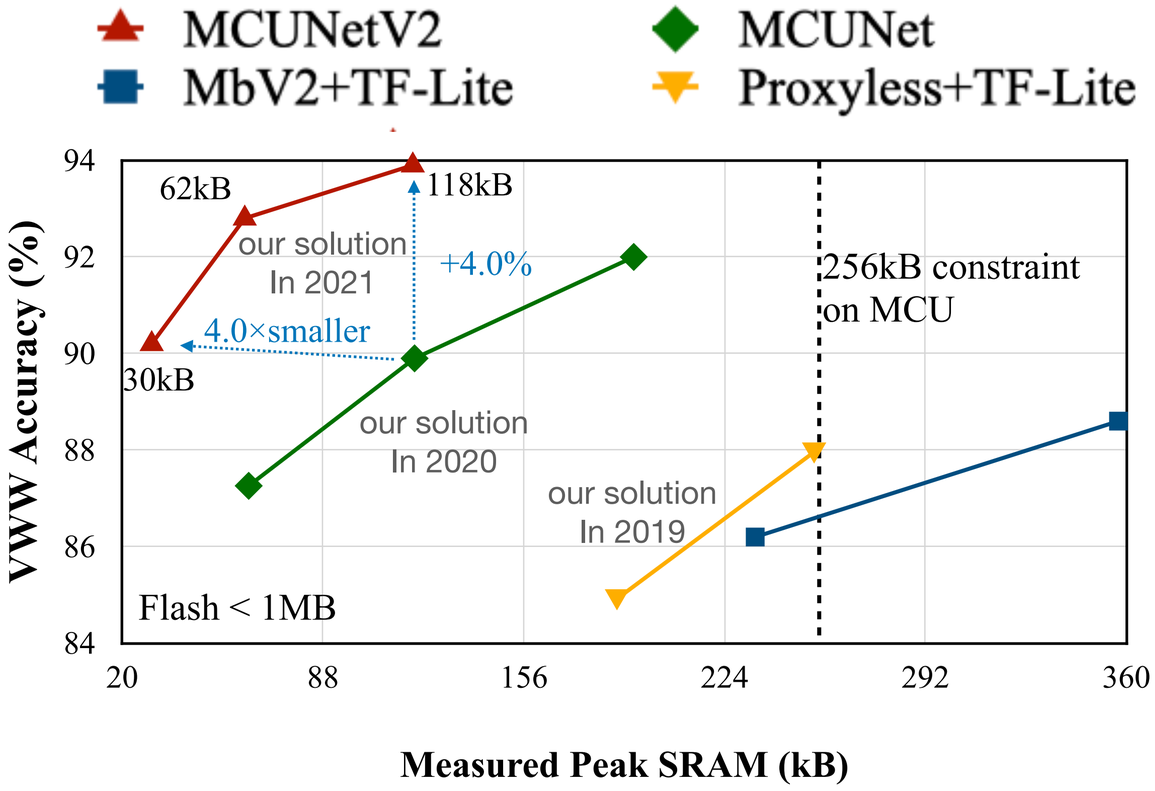
- TinyML: design light-weighted neural networks and deploy on cheap edge devices that has low power, computing, and memory.
- Low-cost ($1-2), low-power, small, everywhere in our lives.
- AI on MCU is **hard**: No DRAM. No OS. Extreme memory constraint.
- Existing work optimize for **#parameters**, but **#activation** is the real bottleneck.
- MCUNet: first to achieve >70% ImageNet top1 accuracy on a microcontroller.
- **Cloud AI**: ResNet; **Mobile AI**: MobileNet; **Tiny AI**: MCUNet.

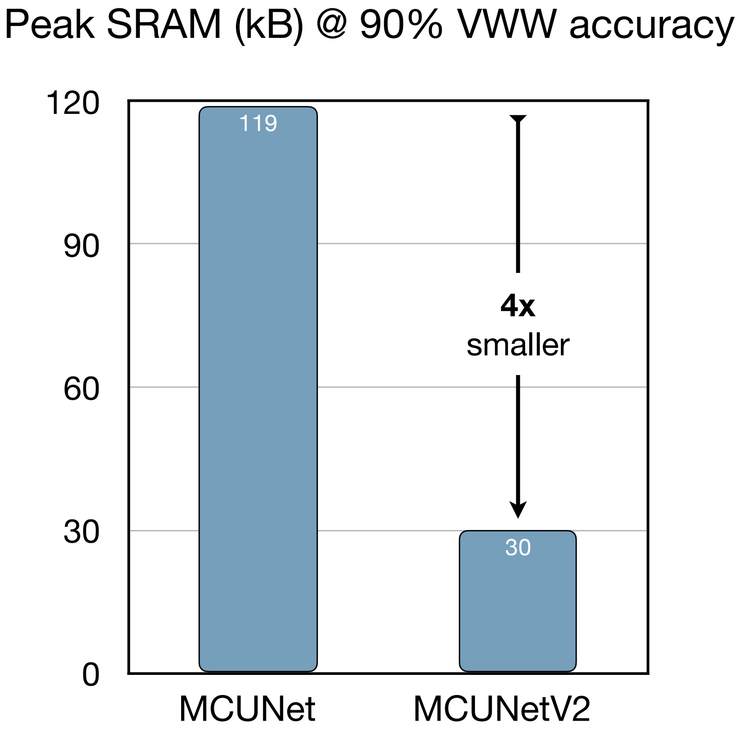




Background work: MCUNet-v2: Patch-Based Inference

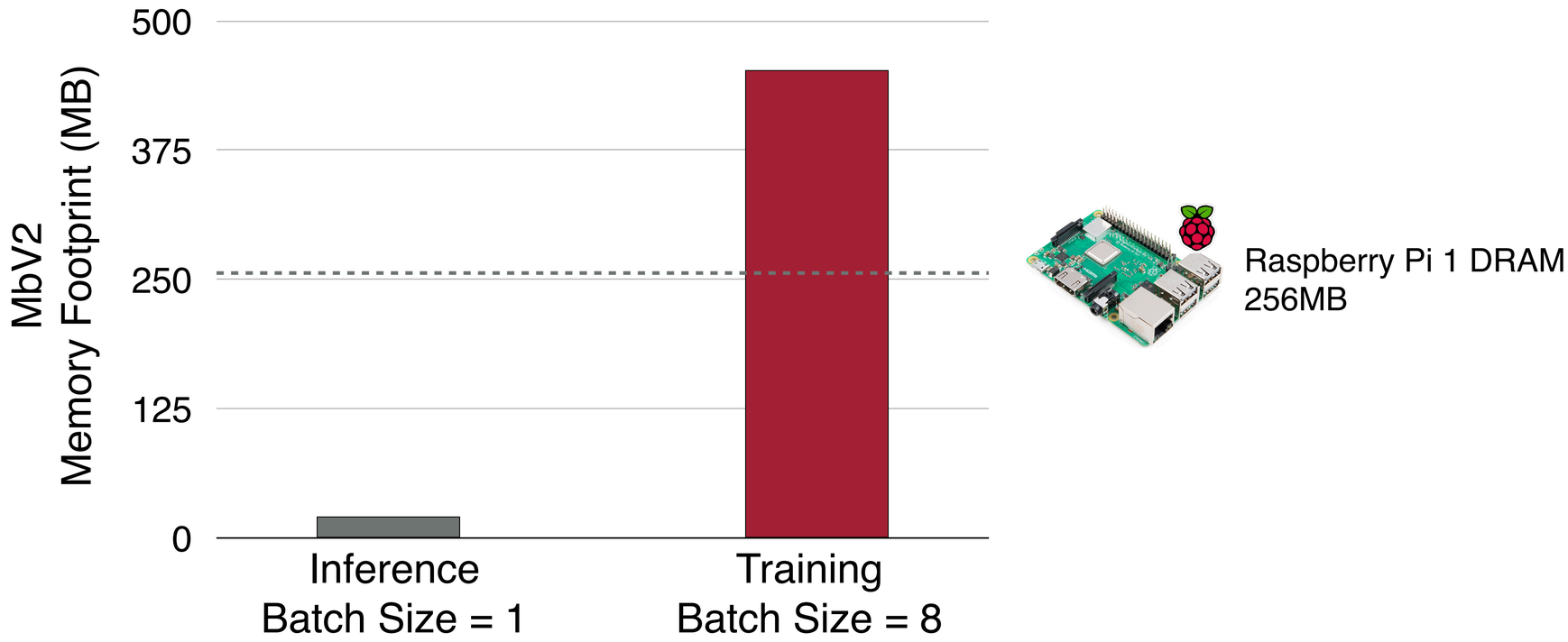
Detect person using only 30KB of memory!





MCUNet V2, NeurIPS'21

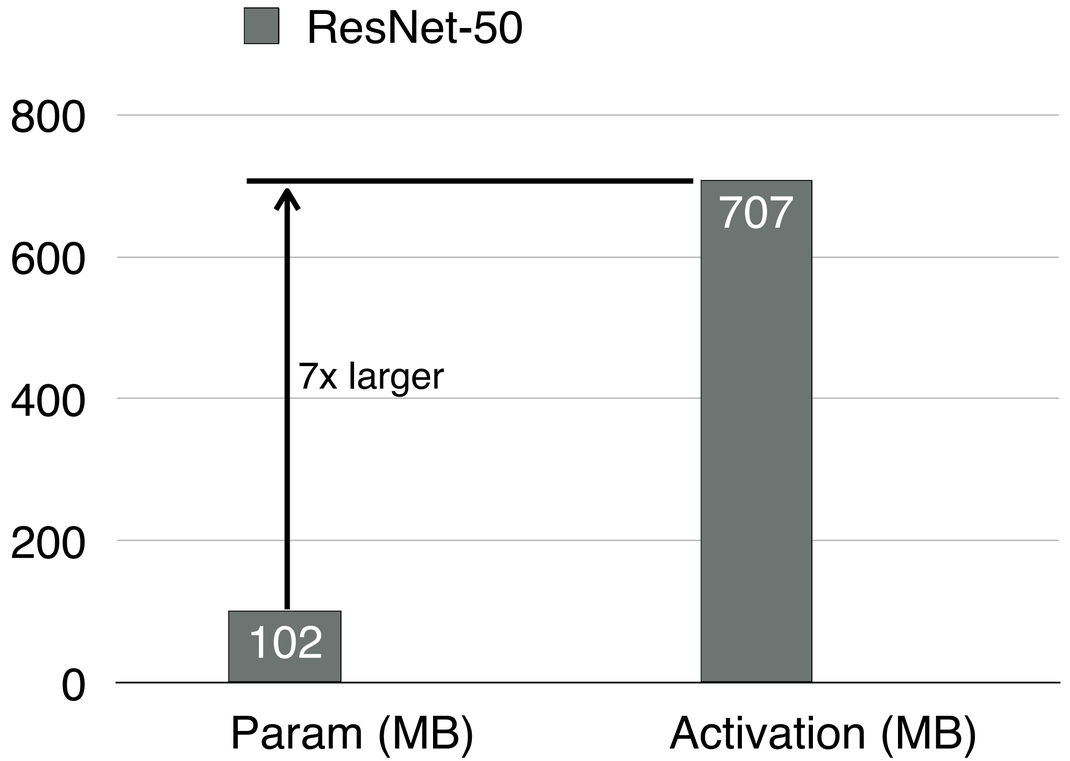
Training Memory is much Larger than Inference



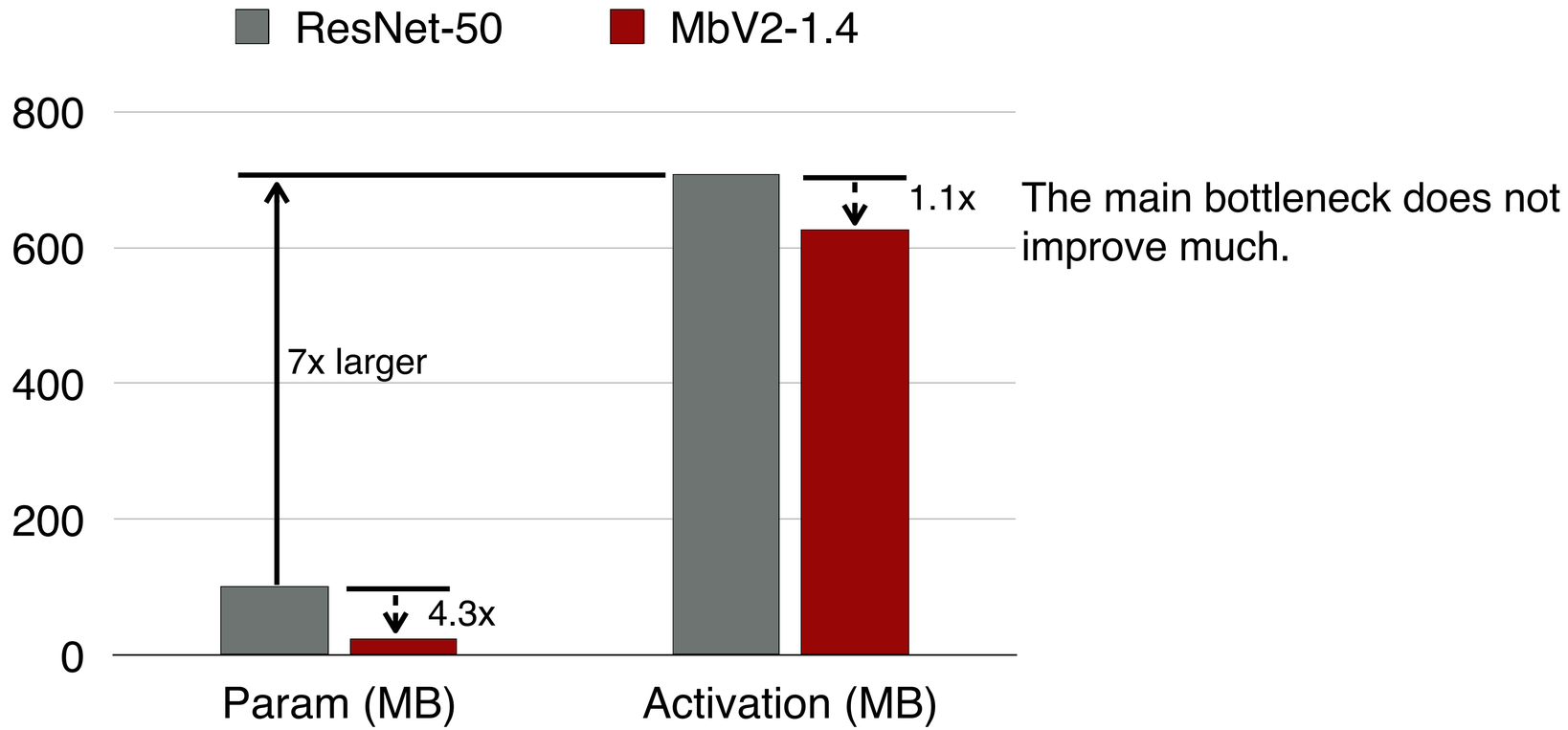
- Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.

- 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 #Trainable Parameters

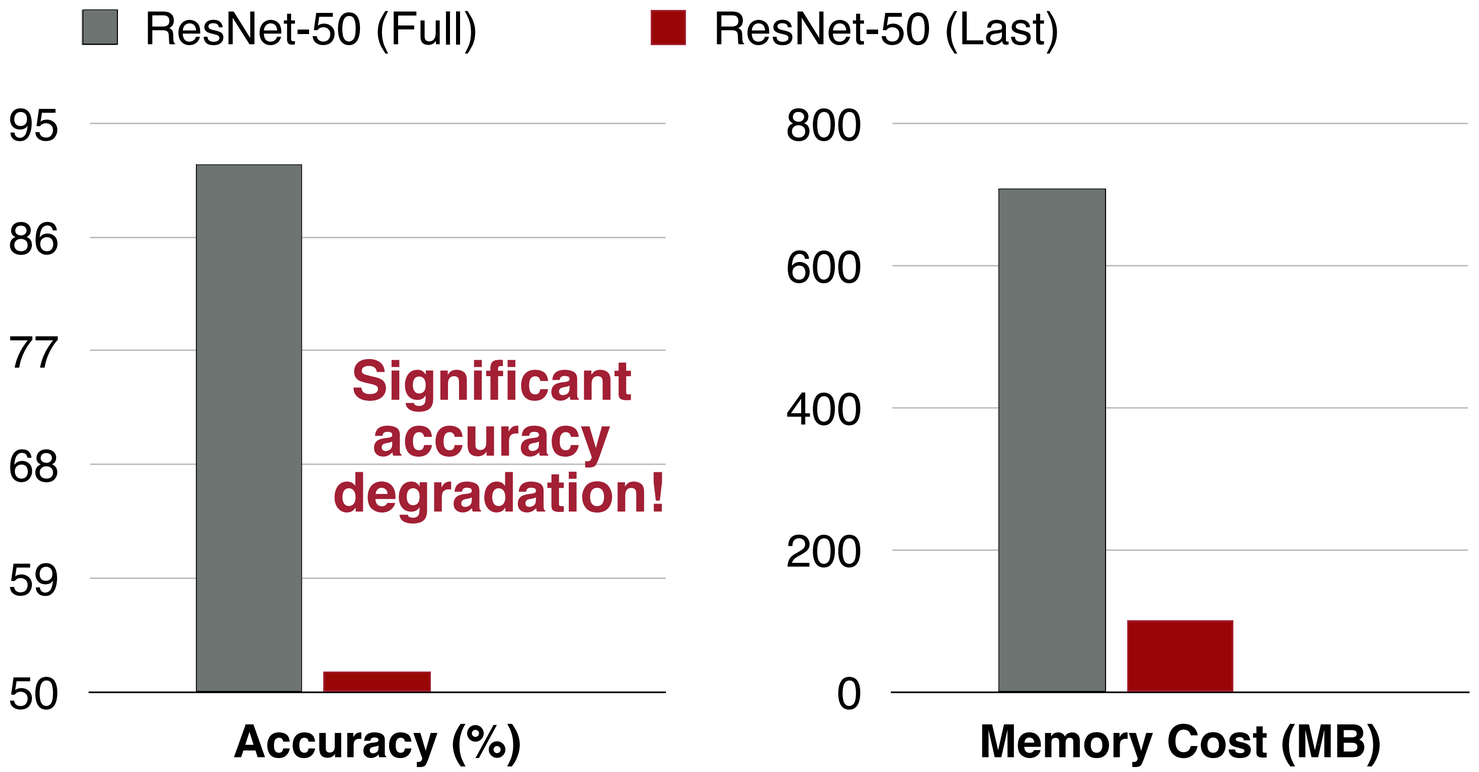


#Activation is the Memory Bottleneck, not #Trainable Parameters



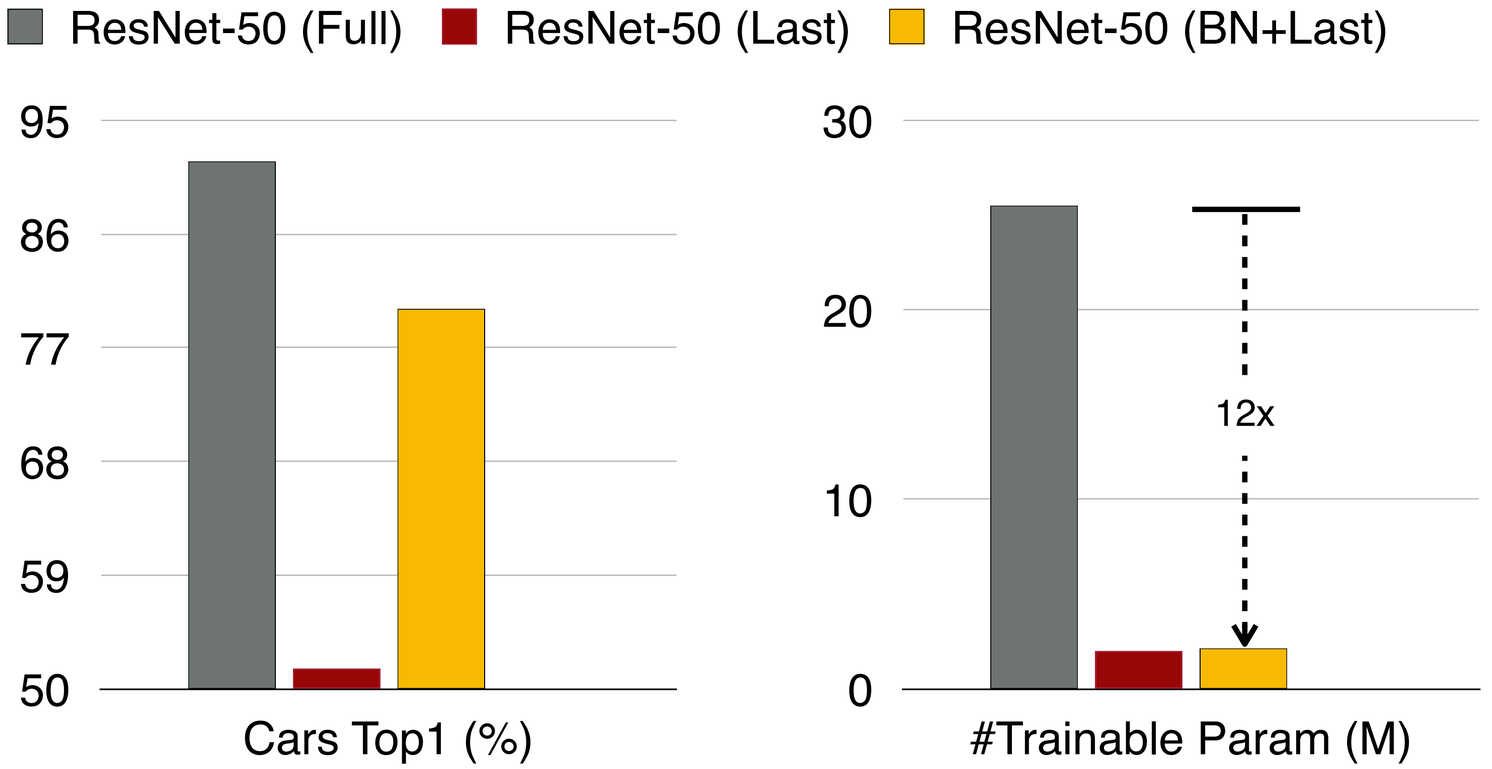
- Previous methods focus on reducing the number of parameters or FLOPs, while the main bottleneck does not improve much.

What about just finetune the last layer?



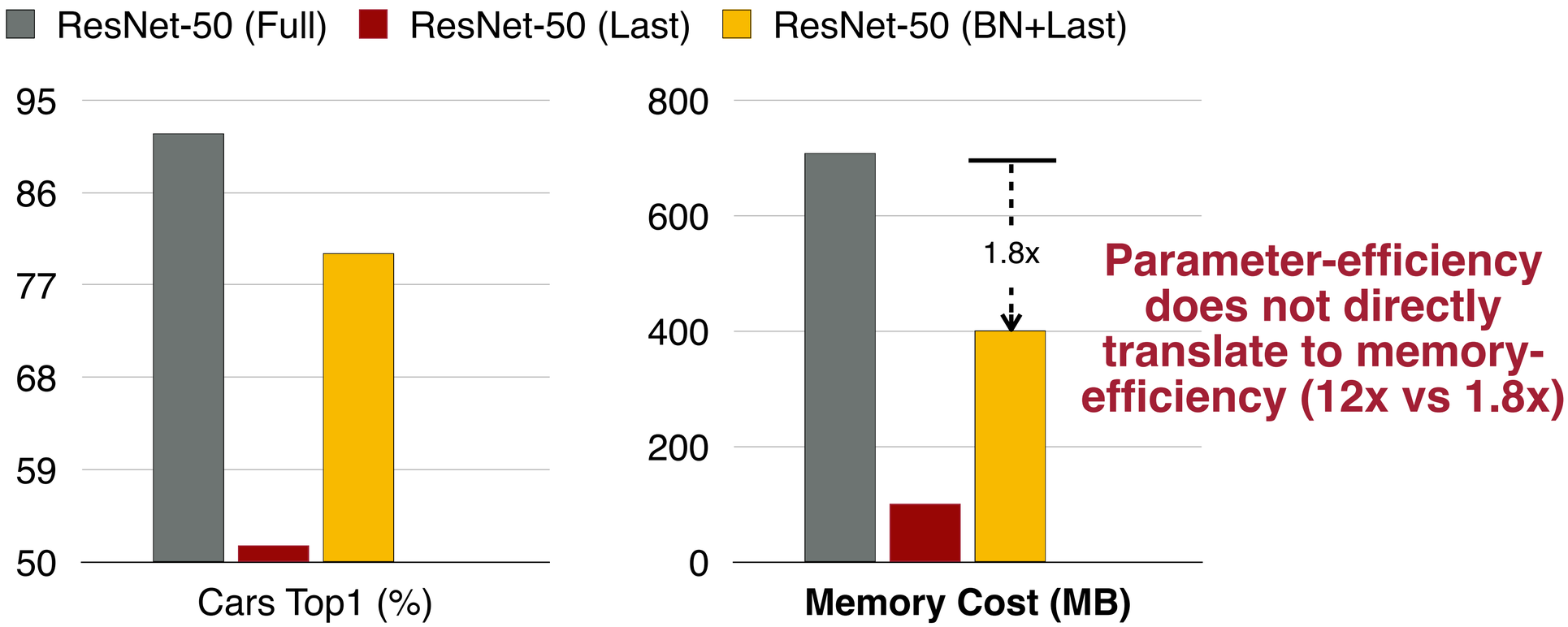
- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.

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 layer. Parameter-efficient.

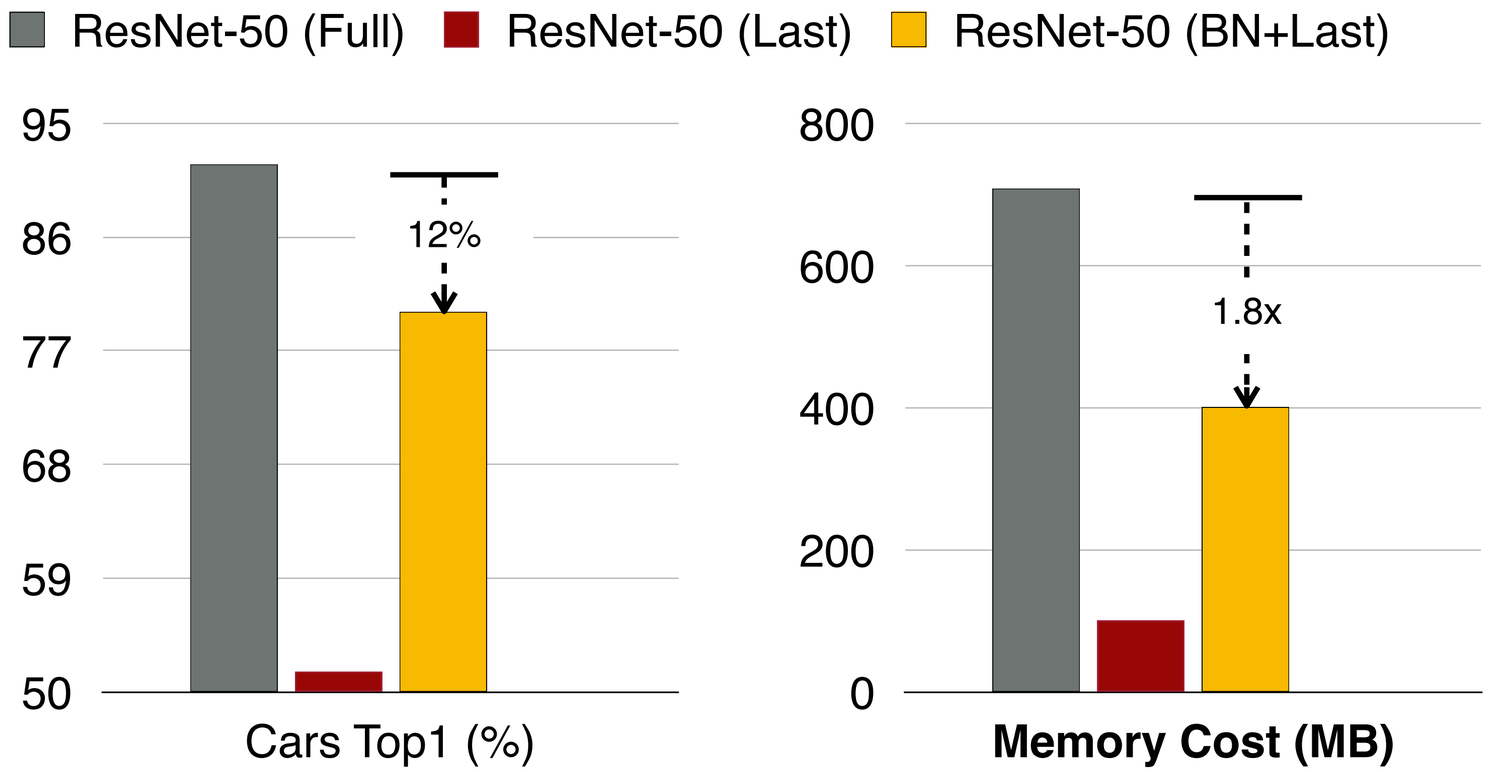
Related Work: Parameter-Efficient Transfer Learning



Parameter-efficiency does not directly translate to memory-efficiency (12x vs 1.8x)

- 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 layer. Parameter-efficient, **but the memory saving is limited.**

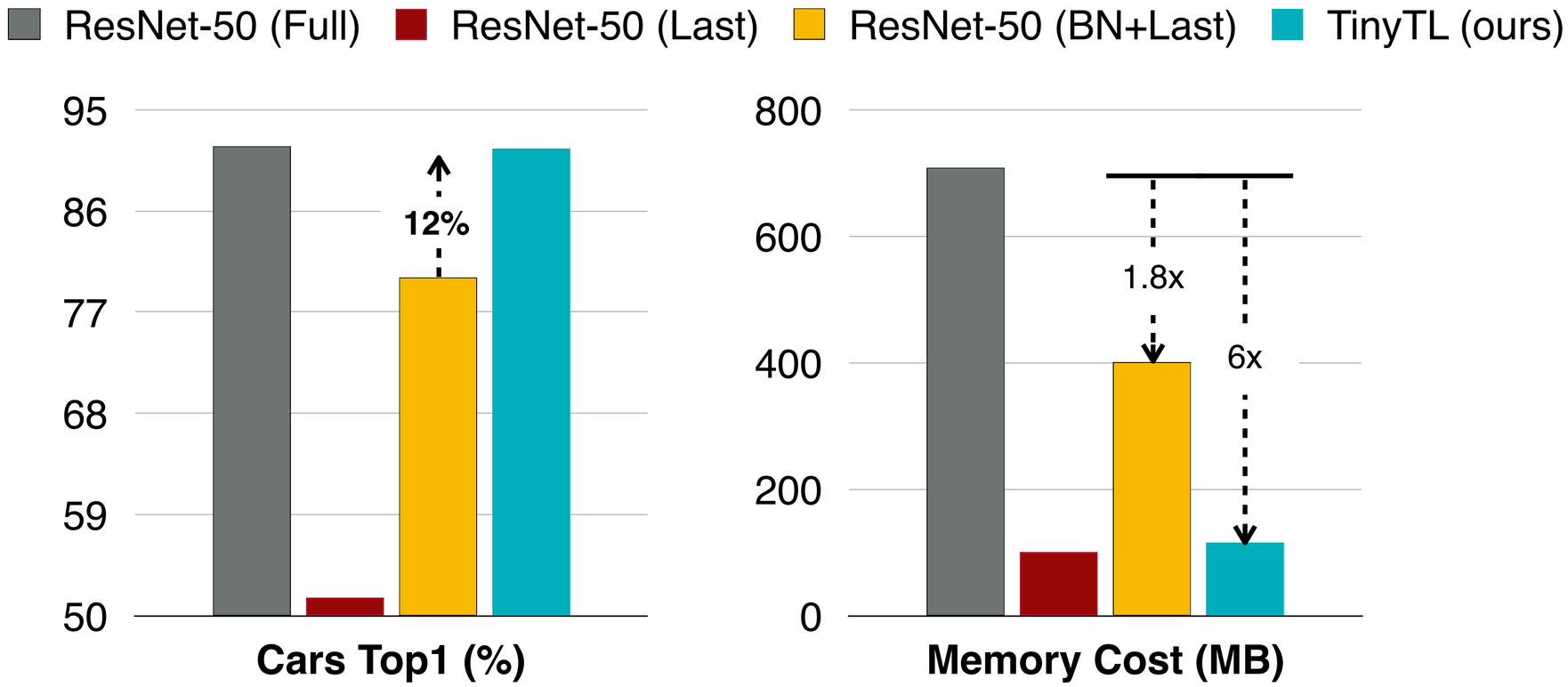
Parameter-Efficiency does not Directly Translate to Memory-Efficiency



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 layer. Parameter-efficient, **but the memory saving is limited. Significant accuracy loss.**

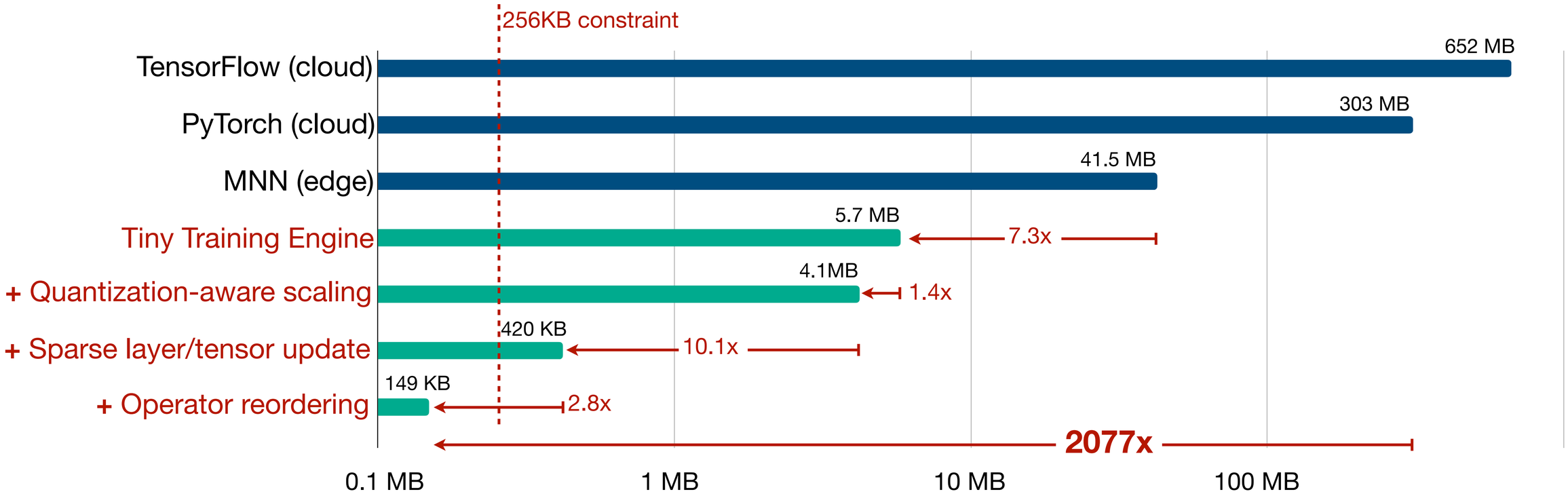
TinyTL: Memory-Efficient Transfer Learning



- 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 layer. Parameter-efficient, **but the memory saving is limited. Significant accuracy loss.**
- TinyTL: fine-tune bias only + lite residual learning: high accuracy, large memory saving

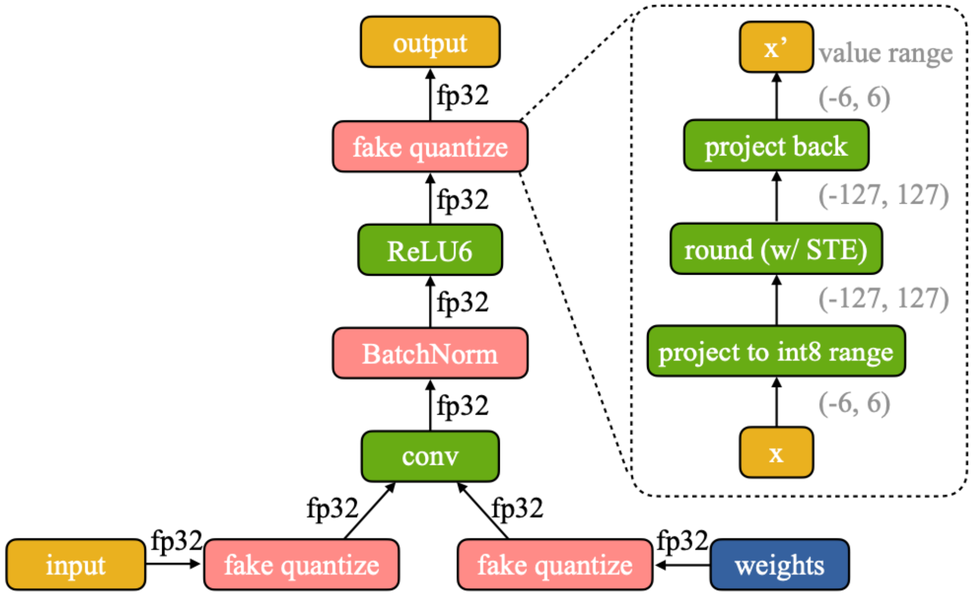
On-Device Training under 256KB Memory

- Reducing memory usage by >1000x

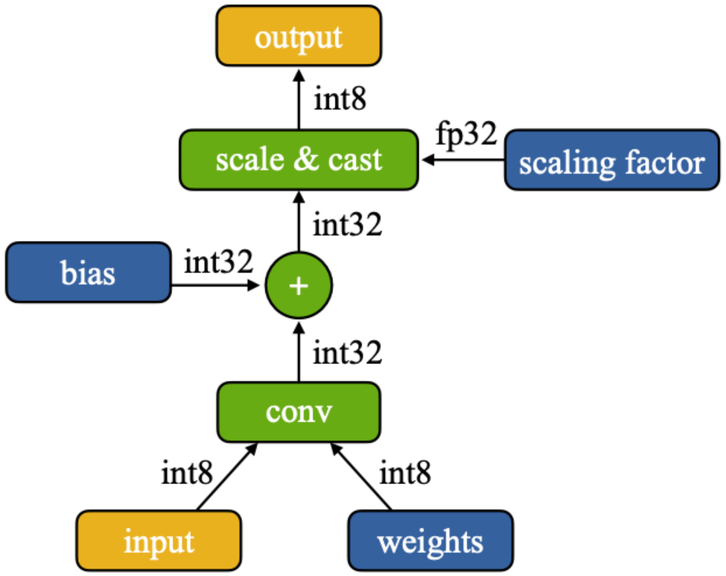


1. Address Optimization Difficulty of Quantized Graphs

- **Fake** quantized graph vs. **Real** quantized graph



(a) Fake Quantization
(quantization aware training)

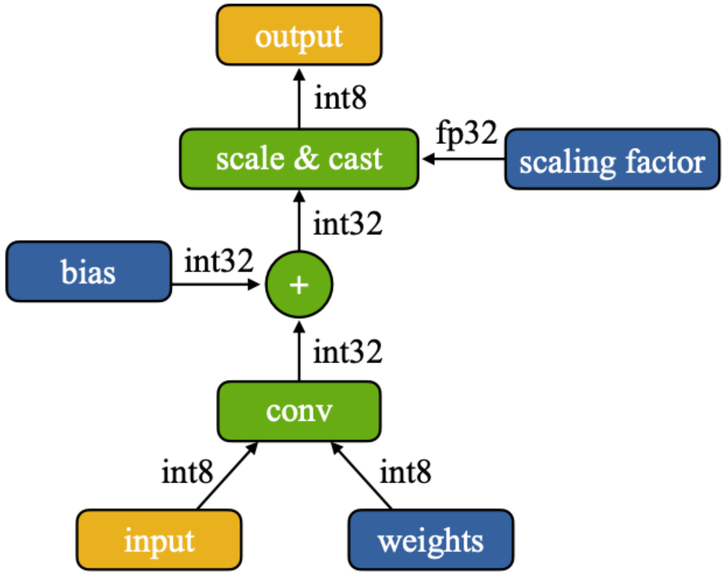


(b) Real Quantization
(on-device training)

	Fake	Real
Weight	FP32	INT8
Activation	FP32	INT8
Batch Norm	Yes	No

1. Address Optimization Difficulty of Quantized Graphs

- **Real** quantized graphs vs. **fake** quantized graphs

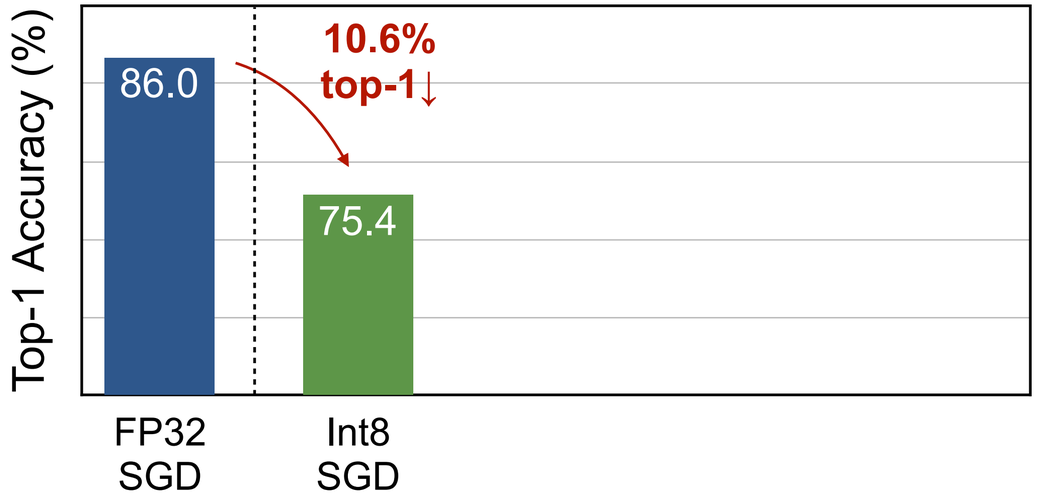


(a) Real Quantization.

Making training difficult:

- Mixed precisions: int8/int32/fp32…
- Lack BatchNorm

Performance Comparison (average on 10 datasets)

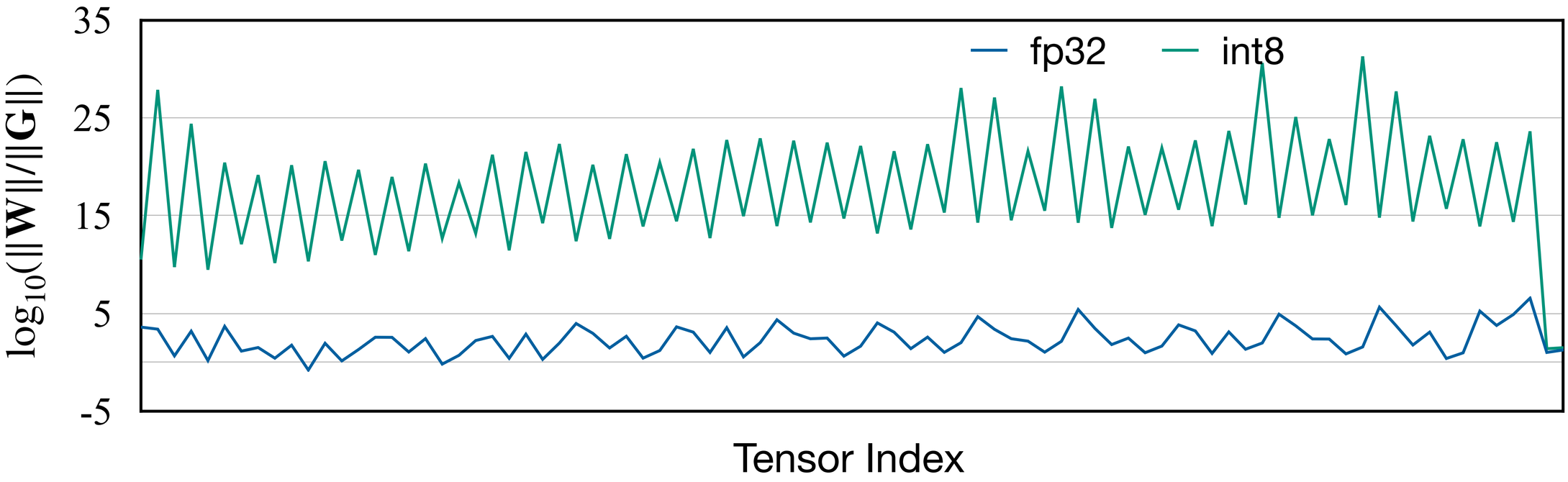


1. Address Optimization Difficulty of Quantized Graphs

- Why is the training convergence worse?

1. Address Optimization Difficulty of Quantized Graphs

- Why is the training convergence worse?
- The scale of weight and gradients does not match in ***real quantized training!***



QAS: Quantization-Aware Scaling

QAS addresses the optimization difficulty of quantized graphs

Quantization overview

$$\bar{\mathbf{y}}_{\texttt{int8}} = \texttt{cast2int8}[s_{\texttt{fp32}} \cdot (\bar{\mathbf{W}}_{\texttt{int8}} \bar{\mathbf{x}}_{\texttt{int8}} + \bar{\mathbf{b}}_{\texttt{int32}})],$$

Per Channel scaling

$$\mathbf{W} = s_{\mathbf{W}} \cdot (\mathbf{W}/s_{\mathbf{W}}) \overset{\text{quantize}}{\approx} s_{\mathbf{W}} \cdot \bar{\mathbf{W}}, \quad \mathbf{G}_{\bar{\mathbf{W}}} \approx s_{\mathbf{W}} \cdot \mathbf{G}_{\mathbf{W}},$$

Weight and gradient ratios are off by Sw

$$\|\bar{\mathbf{W}}\|/\|\mathbf{G}_{\bar{\mathbf{W}}}\| \approx \|\mathbf{W}/s_{\mathbf{W}}\|/\|s_{\mathbf{W}} \cdot \mathbf{G}_{\mathbf{W}}\| = \boxed{s_{\mathbf{W}}^{-2}} \cdot \|\mathbf{W}\|/\|\mathbf{G}\|.$$

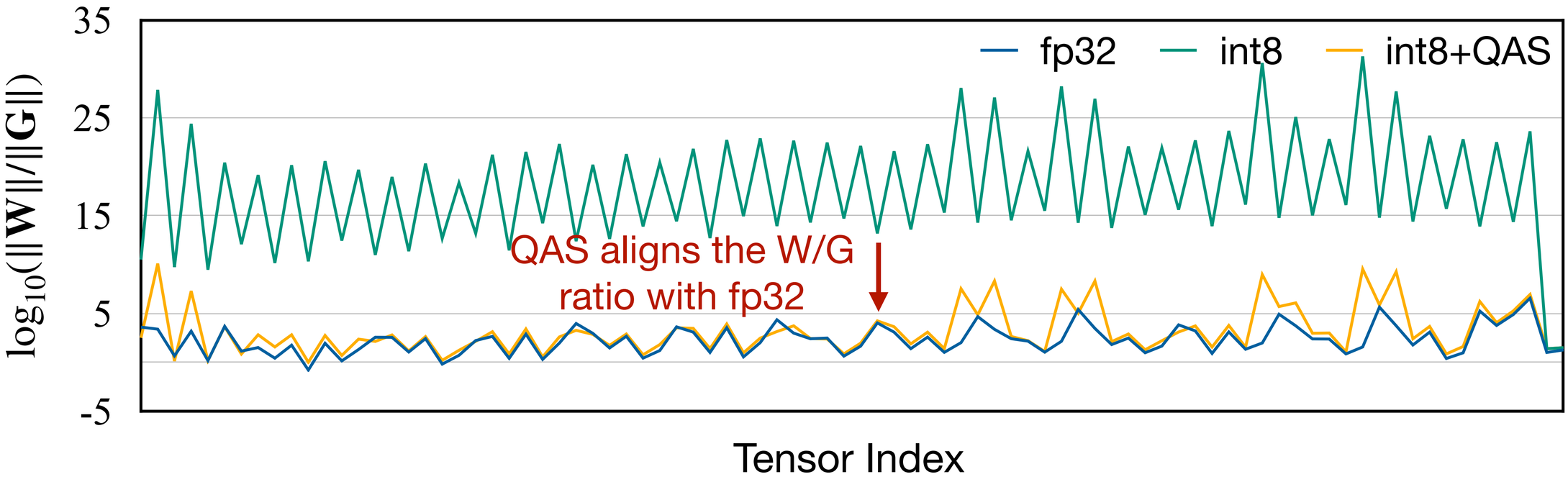
Thus, re-scale the gradients

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

QAS: Quantization-Aware Scaling

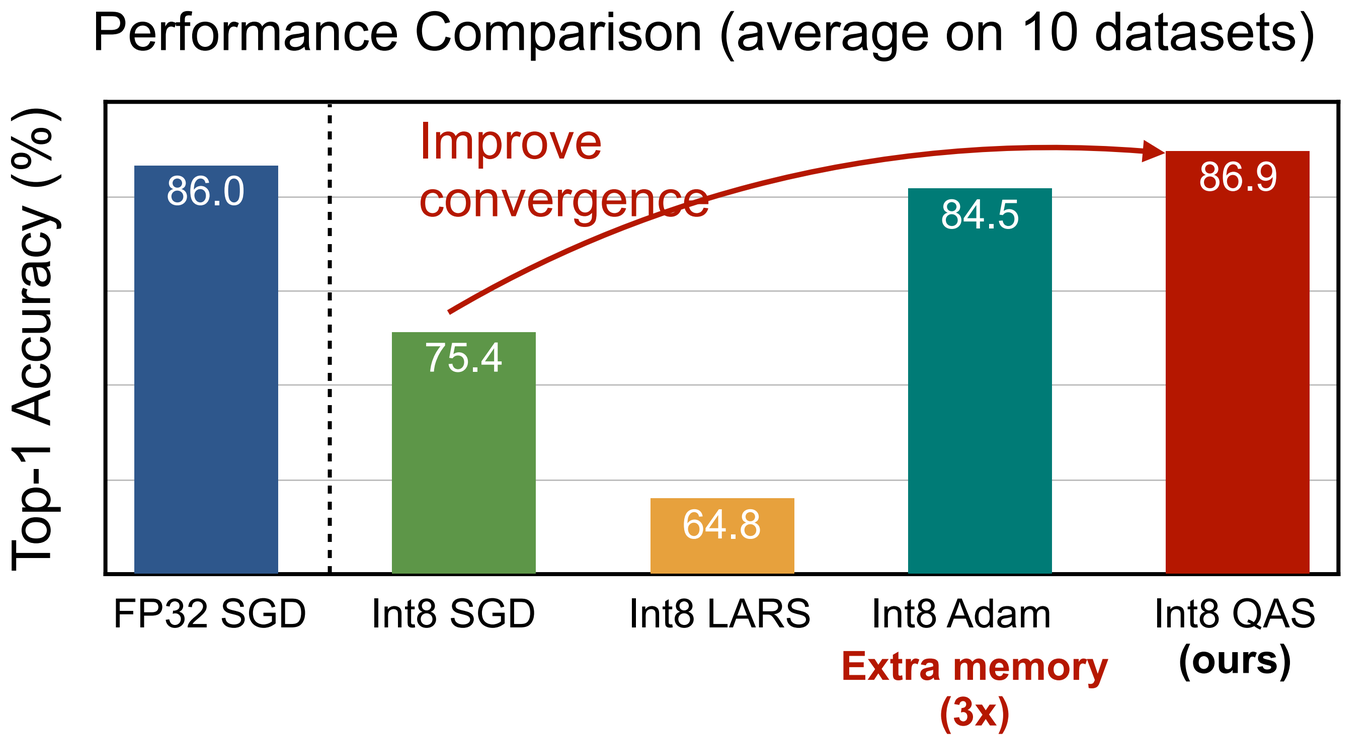
QAS addresses the optimization difficulty of quantized graphs

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

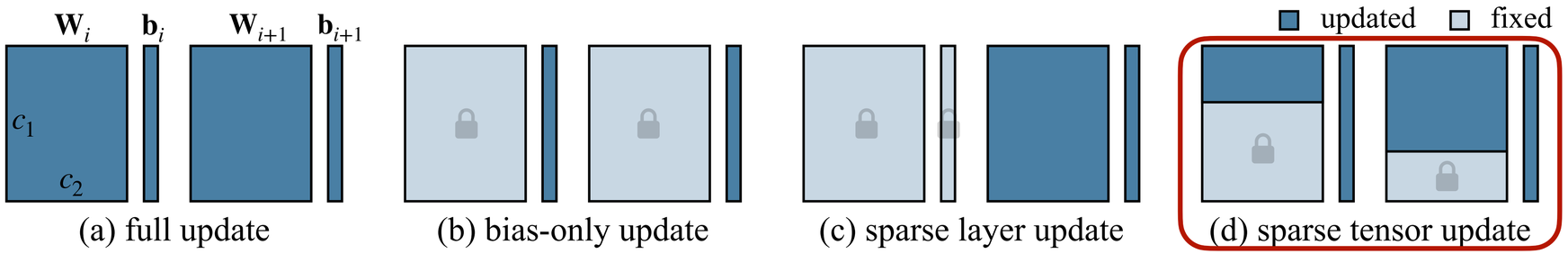


QAS: Quantization-Aware Scaling

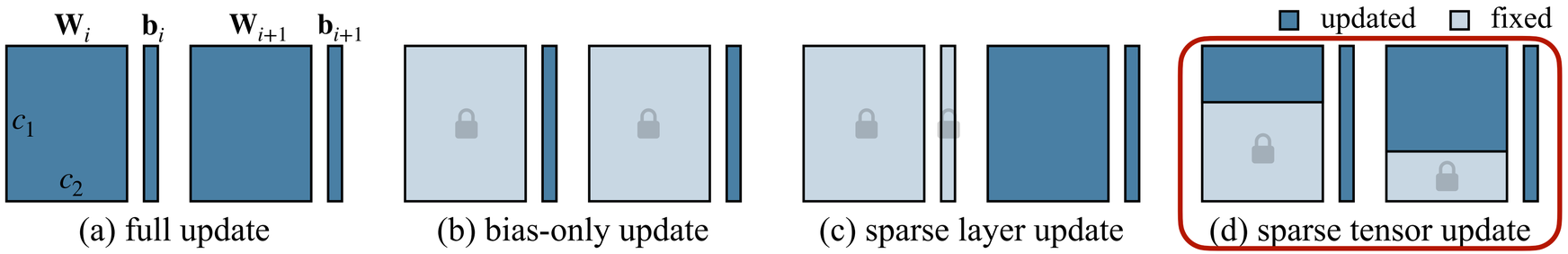
QAS addresses the optimization difficulty of quantized graphs



2. Sparse Layer/Tensor Update



2. Sparse Layer/Tensor Update



Dense Backward

$$\frac{dy}{dw} : \underset{\text{(H, N)}}{\boxed{\text{G.T}}} \quad \underset{\text{(N, M)}}{\boxed{\text{X}}} = \underset{\text{(H, M)}}{\boxed{\text{(dW).T}}}$$

Activation to store: (H, M)

Weight in SRAM: (M, H)

Reduce by 4x

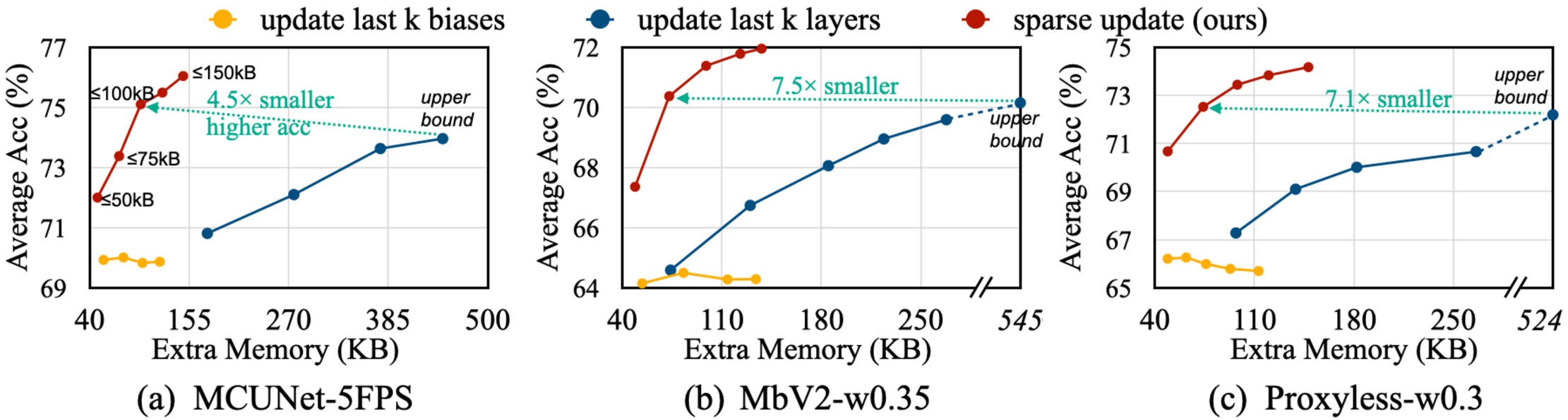
Sparse Tensor Backward

$$\frac{dy}{dw} : \underset{\text{(H, N)}}{\boxed{\text{G.T}}} \quad \underset{\text{(N, M)}}{\boxed{\text{X}}} = \underset{\text{(H, M)}}{\boxed{\text{(dw).T}}}$$

Activation to store: (H, 0.25*M)

Weight in SRAM: (0.25*M, N)

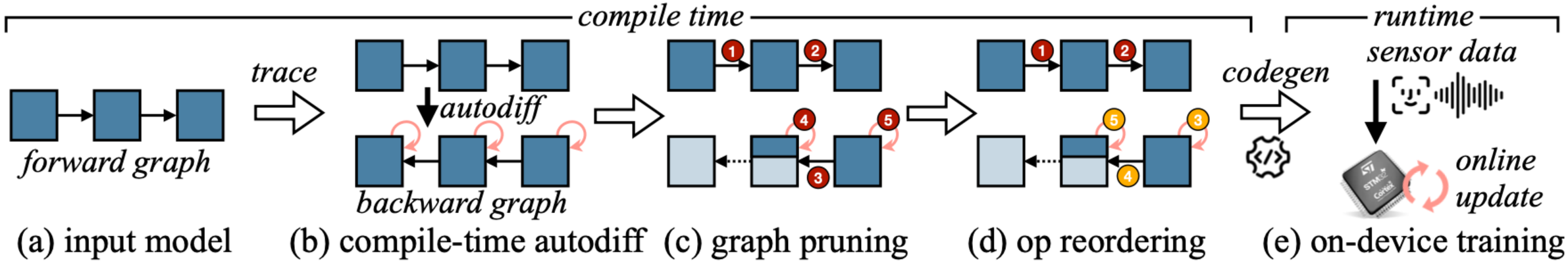
Sparse Update: Lower Memory, Higher Accuracy



(a) MCUNet-5FPS (b) MbV2-w0.35 (c) Proxyless-w0.3

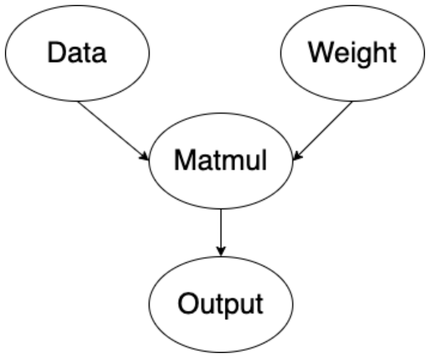
Sparse update can achieve higher transfer learning accuracy using **4.5-7.5x** smaller extra memory.

3. Tiny Training Engine (TTE)



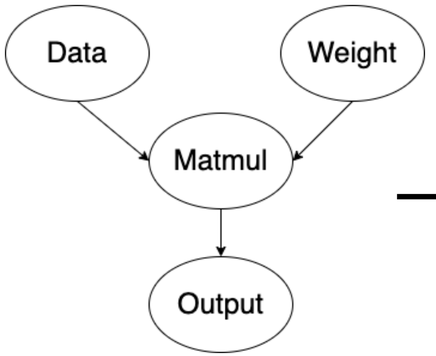
Previous DL Training

1. Computation Graph (forward)



Previous DL Training

1. Computation Graph (forward)

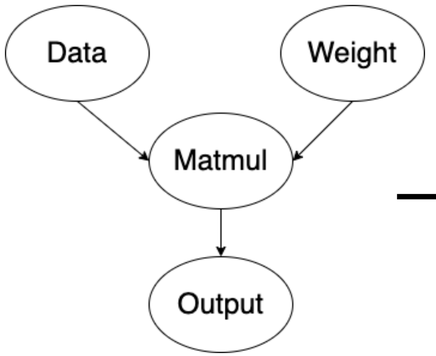


2. Autograd Engine

$$f(x) \rightarrow f'(x)$$

Previous DL Training

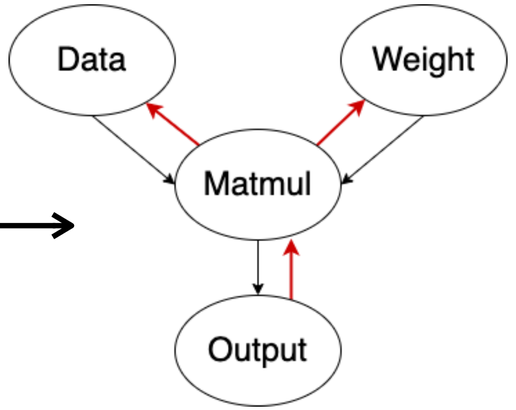
1. Computation Graph (forward)



2. Autograd Engine

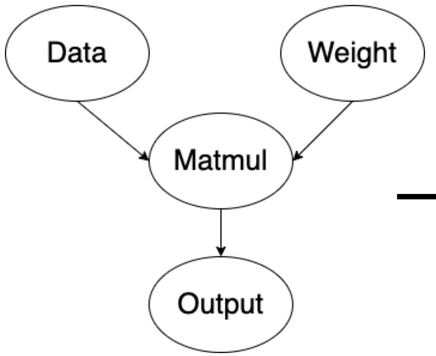
$$f(x) \rightarrow f'(x)$$

3. Computation Graph (backward)



Previous DL Training

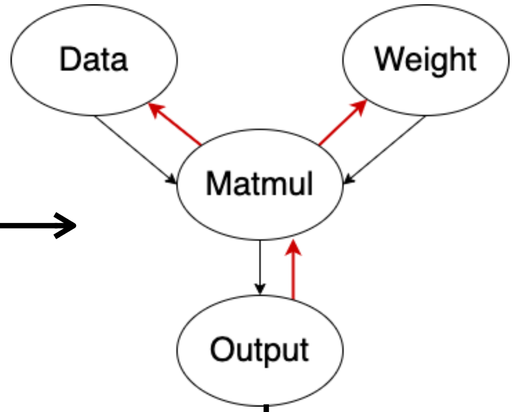
1. Computation Graph (forward)



2. Autograd Engine

$$f(x) \rightarrow f'(x)$$

3. Computation Graph (backward)



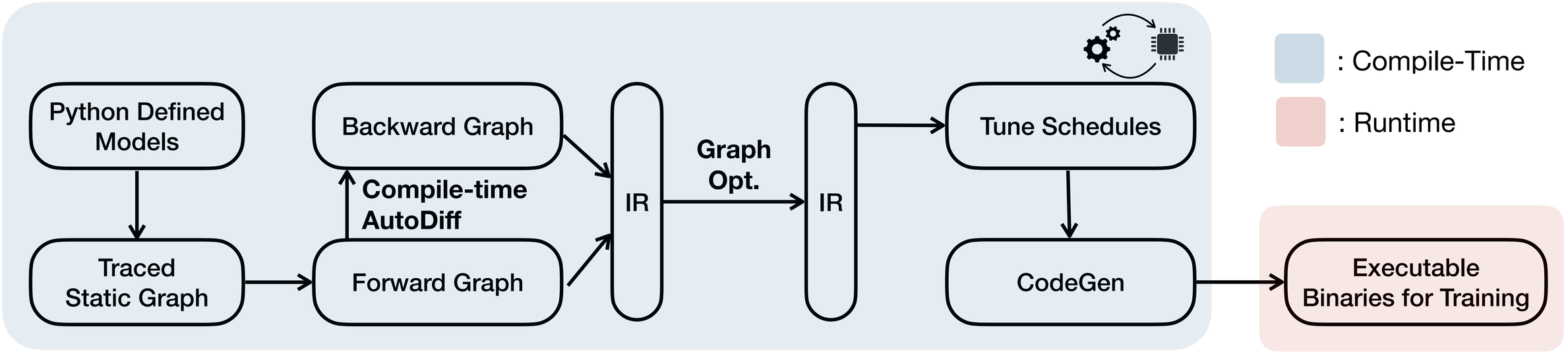
4. Execution Engine

Detailed execution schedules.

Limitations with Previous Training Infra

- Runtime is heavy
  - Autodiff at runtime
  - Heavy dependencies and large binary size
  - Operators optimized for the cloud, not for edge
- Memory is heavy
  - A lot of intermediate (and unused) buffers
  - Has to compute full gradients

Tiny Training Engine

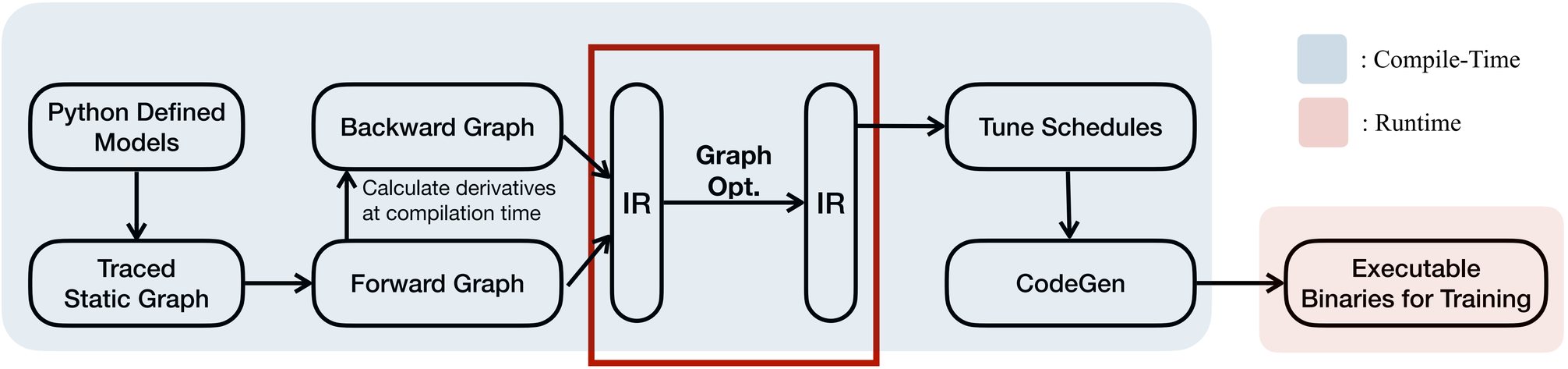


Tiny Training Engine (TTE) **separates** the runtime and compile-time.

TTE offloads most workloads like autodiff / graph optimization / perform tuning **into compile-time.**

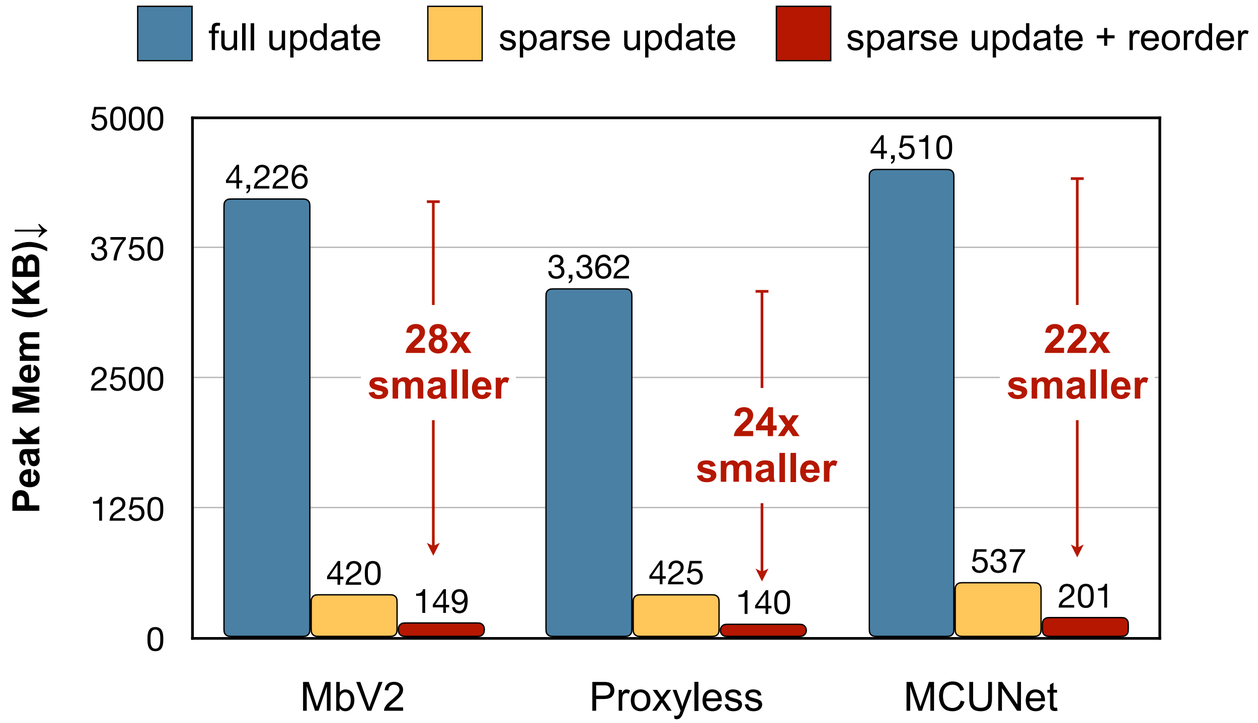
Thus, the overhead of runtime is **minimized**.

Tiny Training Engine Workflow



- Graph-level optimizations:
  - Sparse layer / sparse tensor update
  - Operator reordering and in-place update
  - Constant folding
  - Dead-code elimination

Sparse Layer / Sparse Tensor Update

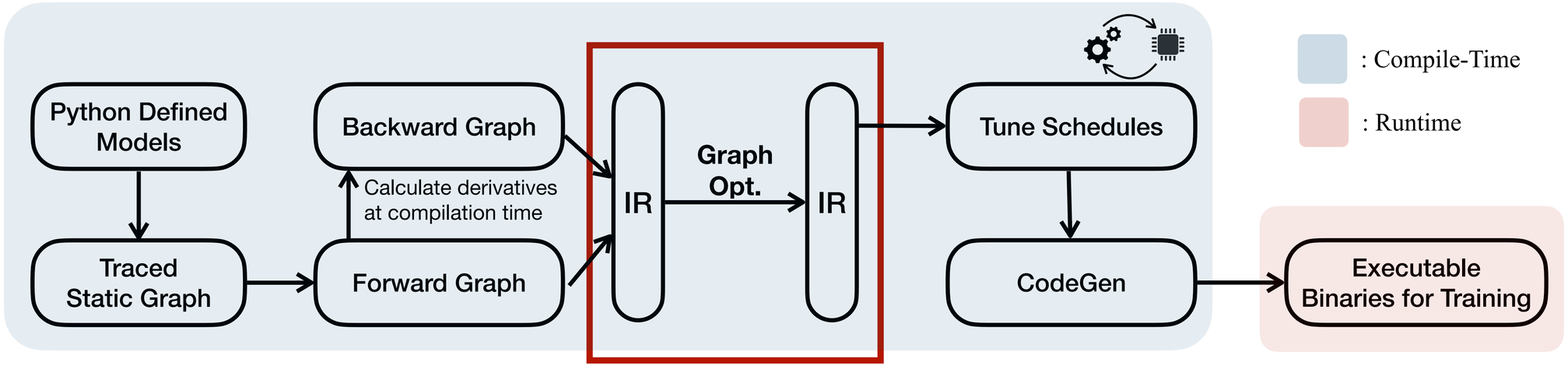


Tiny Training Engine supports backward graph pruning and sparse update at IR-level.

After pruning, un-used weights and sub-tensors are pruned from DAG => 8-10x memory saving

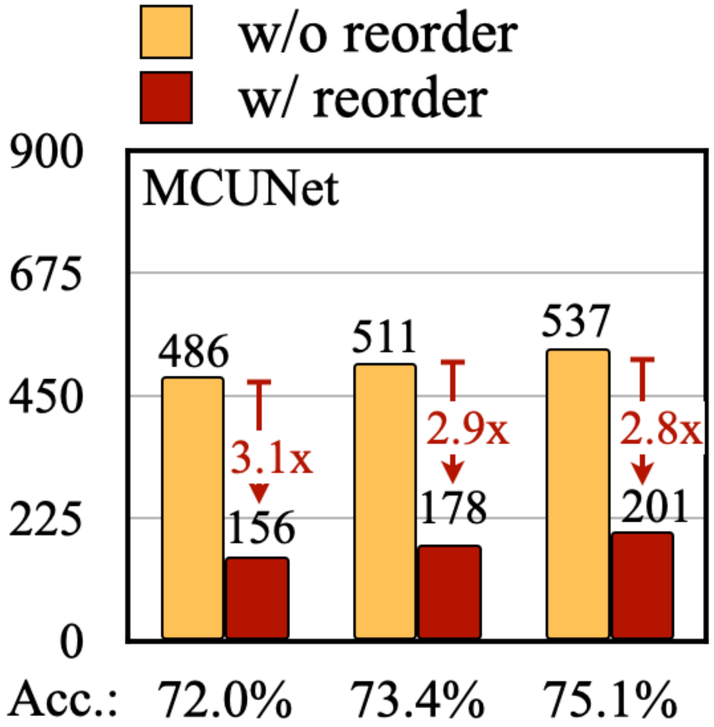
Combined with operator reorder => 22-28x memory saving

Tiny Training Engine



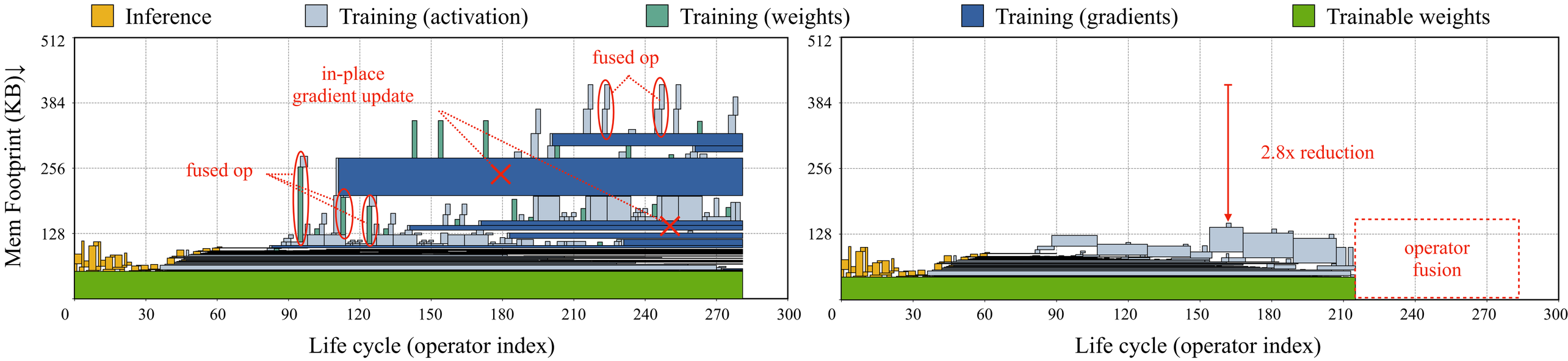
- Graph-level optimizations:
  - Sparse layer / sparse tensor update
  - Operator reordering and in-place update
  - Constant folding
  - Dead-code elimination

Operator Reordering and Inplace Update



By reordering, the gradient update can be immediately applied.Gradients buffer can be released earlier before before back-propagating to earlier layers, leading to **2.7x ~ 3.1x** peak memory reduction.

Life Cycle Analysis

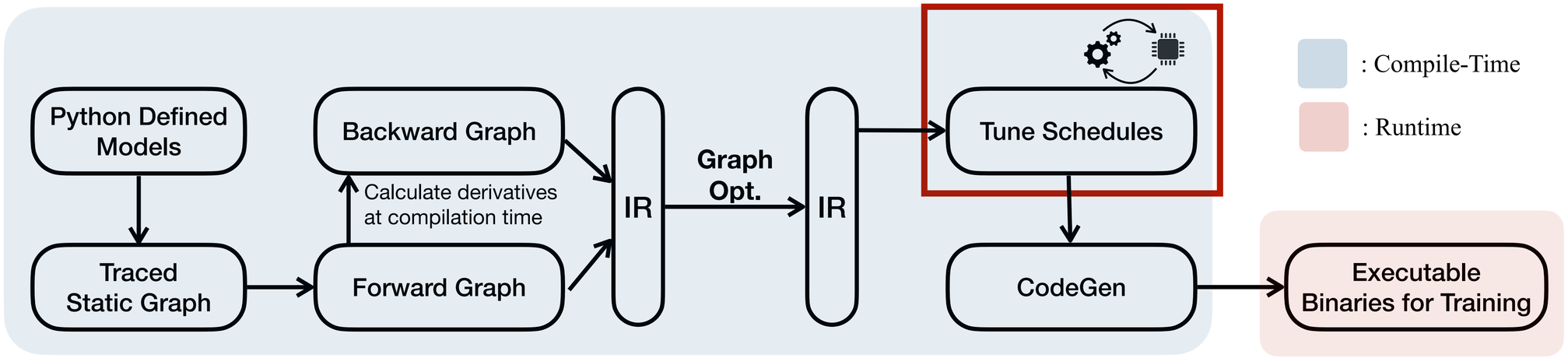


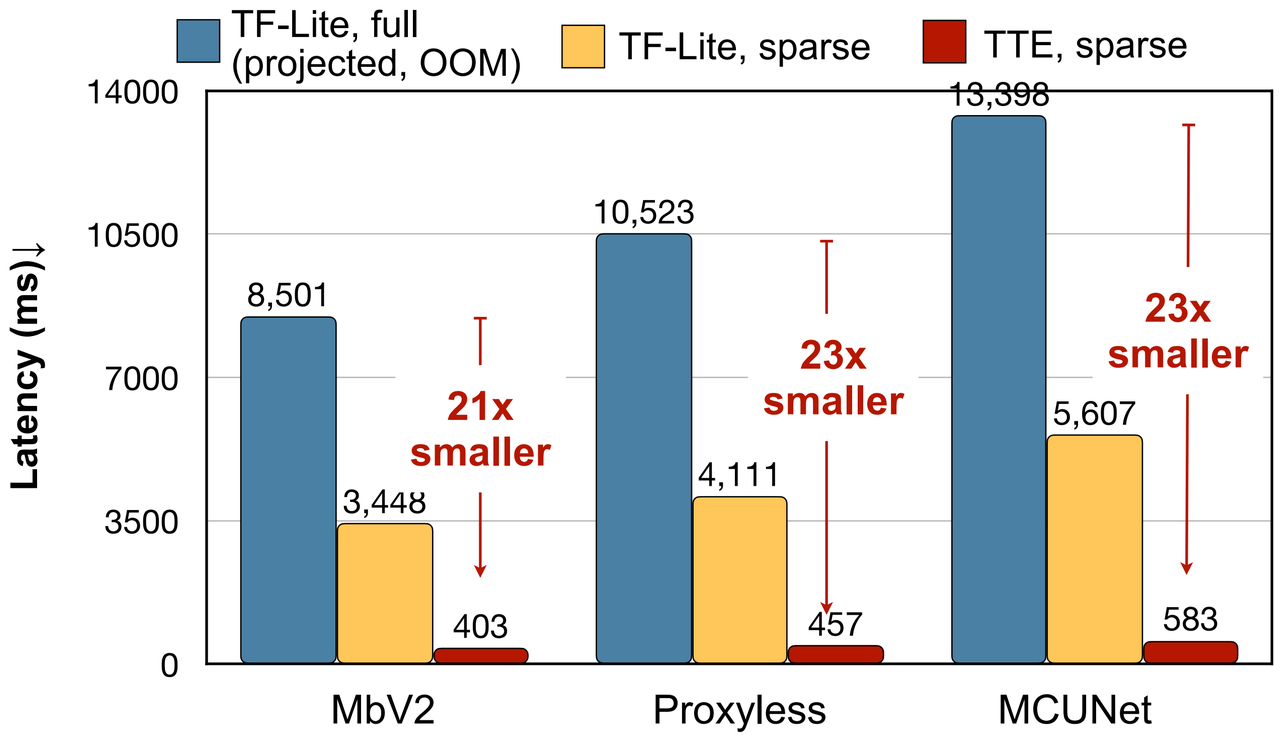
(a) Vanilla backward graph

(b) Optimized backward graph

Operator life-cycle analysis shows memory footprint can be greatly reduced by operator re-ordering.

Tiny Training Engine



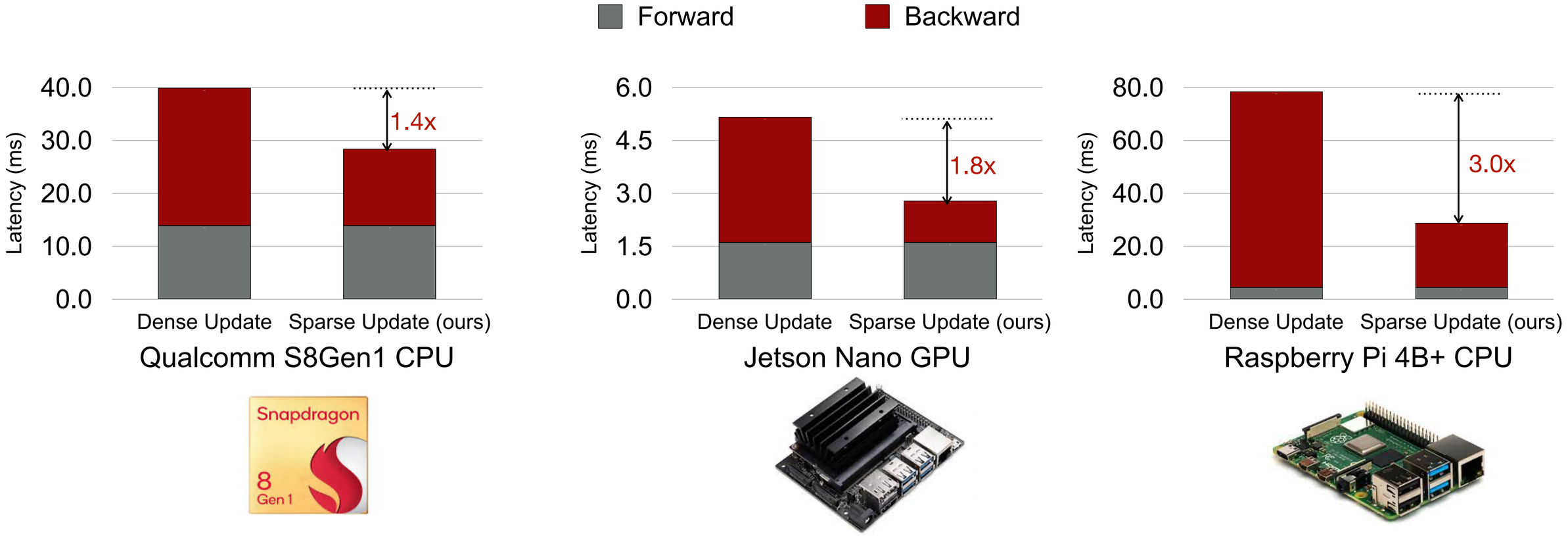


Our optimized operators demonstrate **21x ~ 23x** speedup over TensorFlow-Lite.

2. On-device training

Prediction:
green: correct
red: incorrect

Tiny Training Engine on Diverse Hardware Platforms



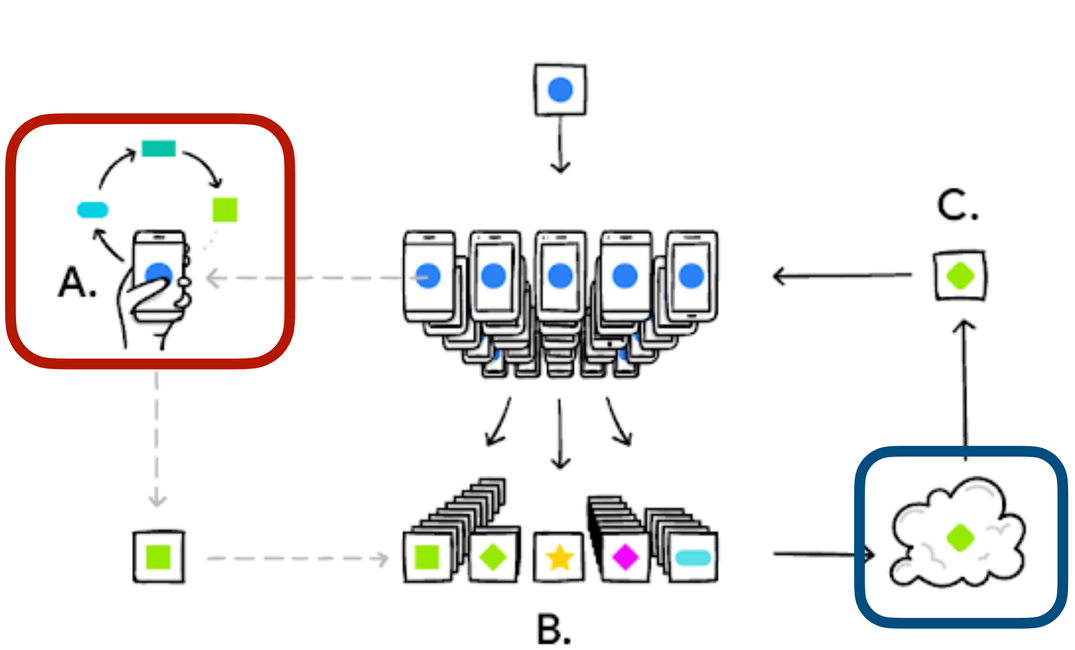
The measured timed includes the **complete forward + backward**.

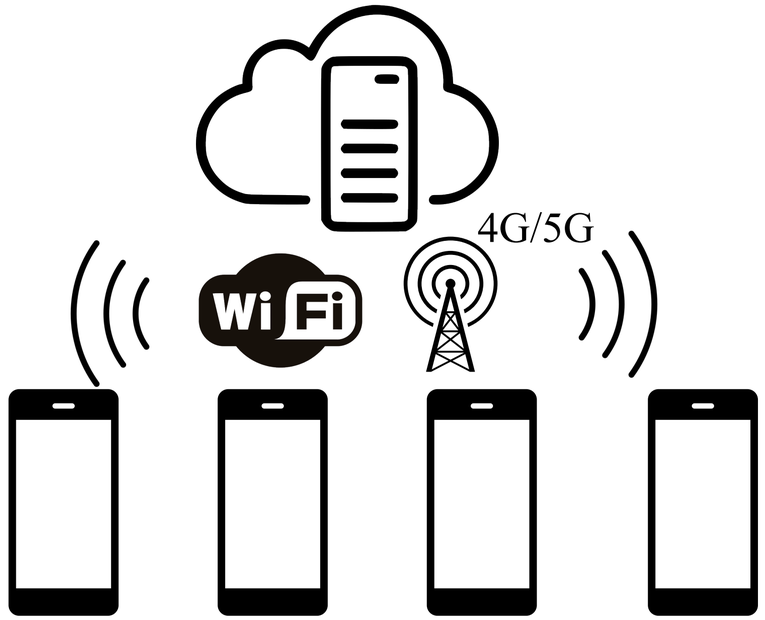
The benchmark model is MobilenetV2-035 with input resolution 128x128.

Our engine **supports various platforms** and our sparse update shows consistent speedup **1.4 to 3.0x**.

Federated On-Device Learning

From single device to multiple devices



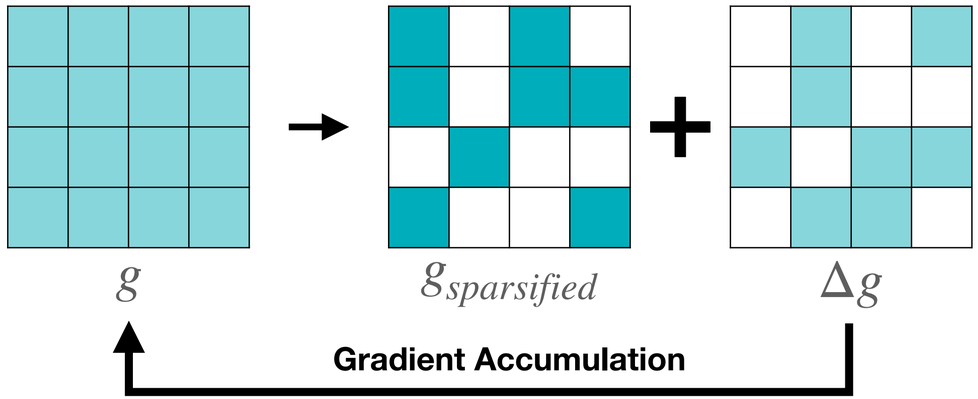


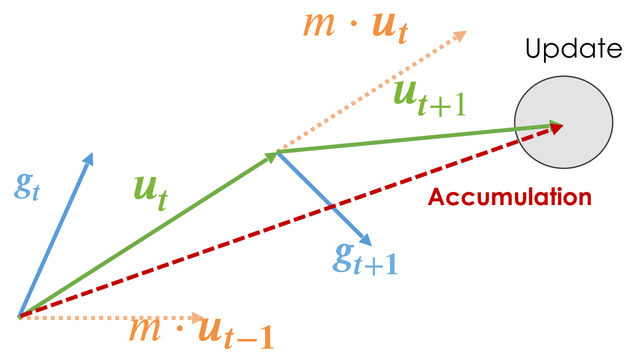
Only **gradients are sharing** across, the **user data** never leaves local device.

Connected through WiFi or Cellular network
Bandwidth up to **1Gb/s**, Latency **~200ms**.

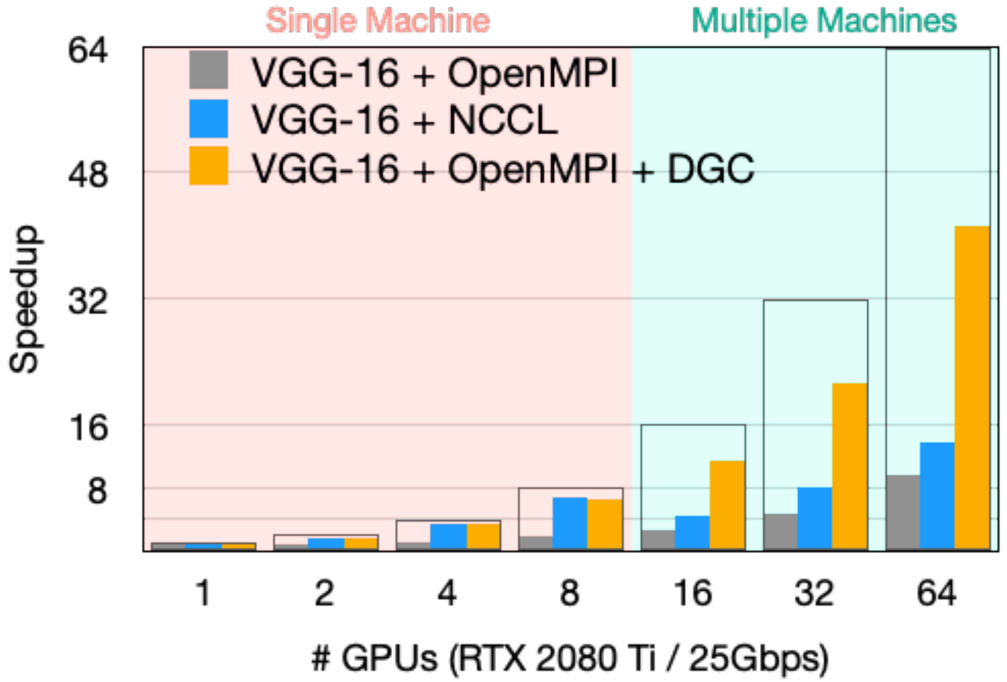
Federated learning suffers from limited communication bandwidth and long latency for mobile devices.

Deep Gradient Compression: Reduce Bandwidth





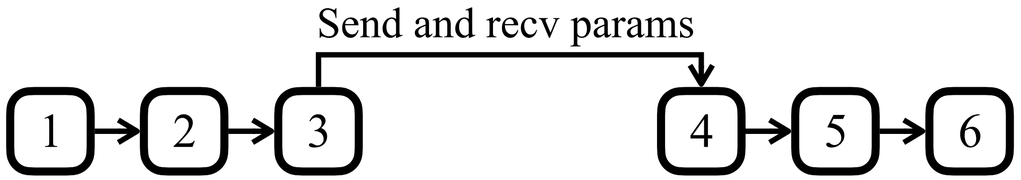
Momentum Correction



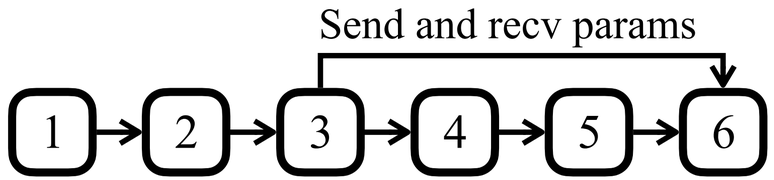
- Reduce the bandwidth by Deep Gradient Compression, which can reduce the gradients by 500x without losing accuracy.

Deep Gradient Compression, ICLR'18

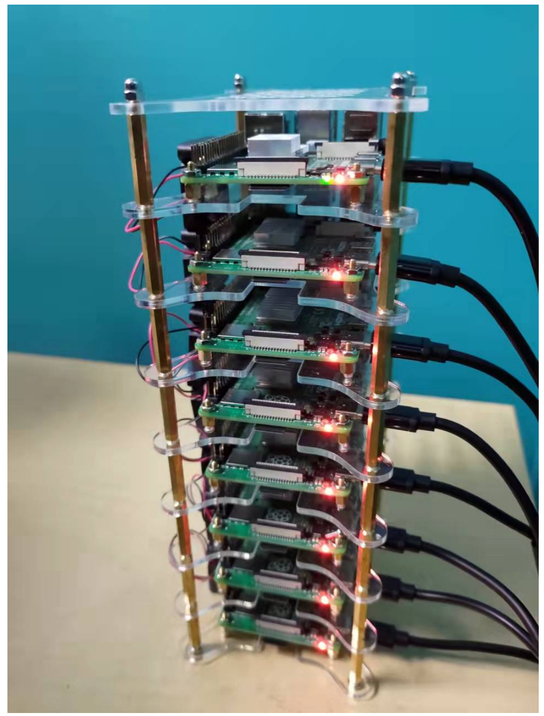
Delayed Gradient Averaging: Tolerate Latency

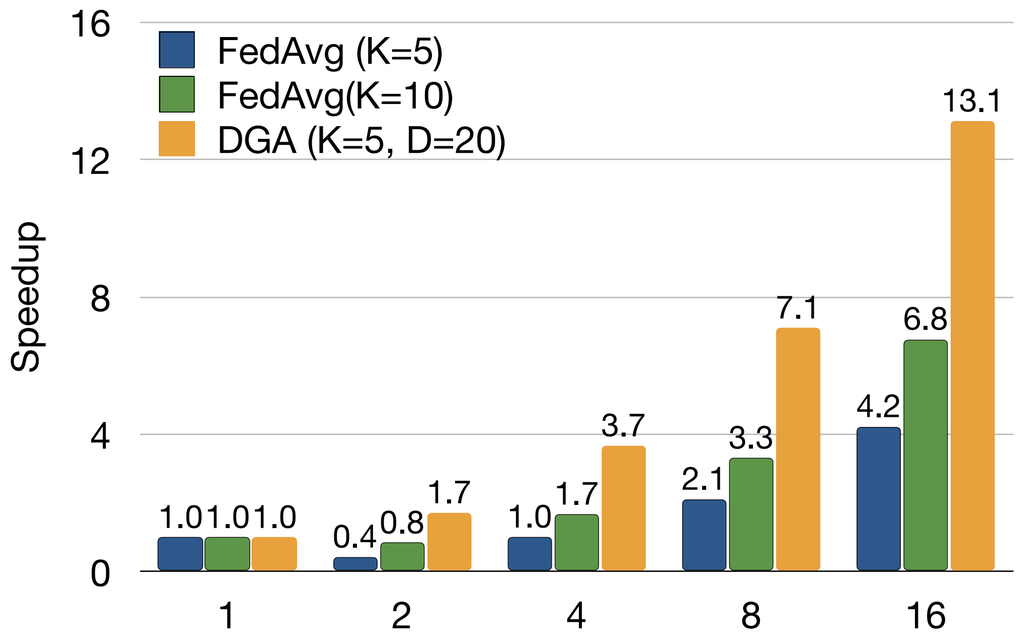


W/o delay: all the local machines are blocked to wait for the synchronization to finish



With delay: Worker keep performing local updates while the parameters are in transmission.

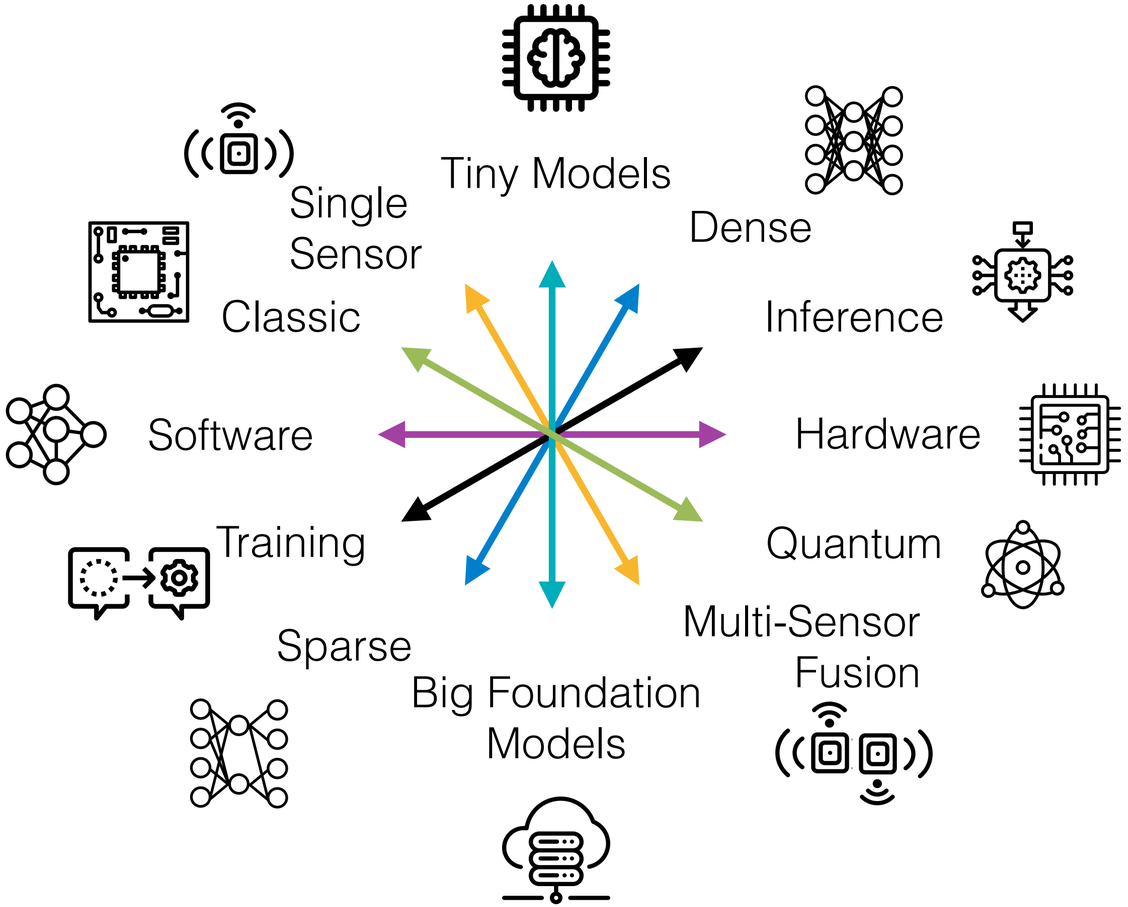




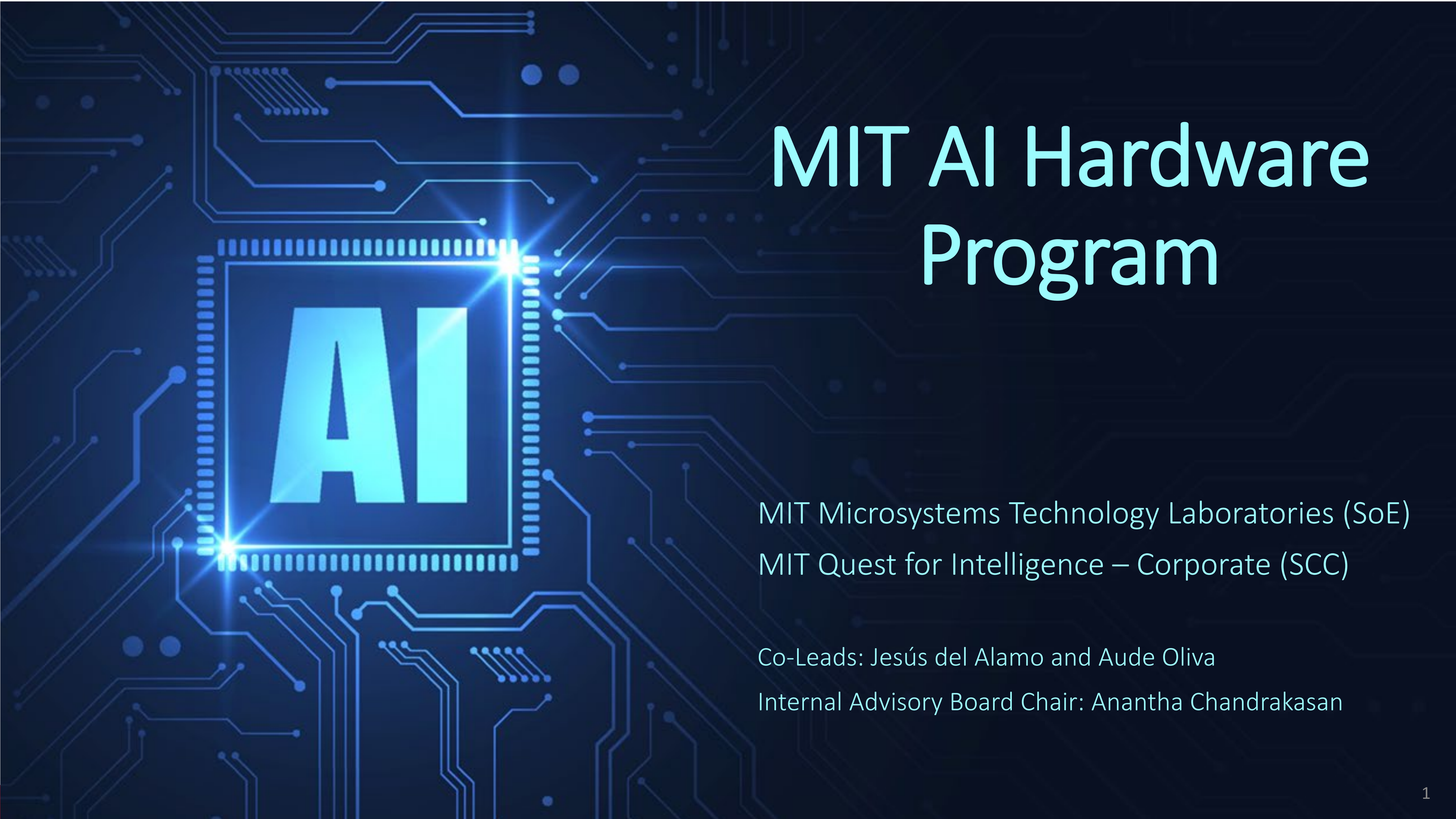
Delayed Gradient Averaging, NeurIPS'21

TinyML and Efficient Deep Learning

https://hanlab.mit.edu/



1. Learning both Weights and Connections for Efficient Neural Network, NeurIPS'15
2. Deep Compression, ICLR'16
3. AMC, ECCV'18
4. ProxylessNAS, ICLR'19
5. Once For All, ICLR'20
6. HAT, ACL'20
7. Anycost GAN, CVPR'21
8. SPVNAS, ECCV'21
9. Lite Pose, CVPR'22
10. NAAS, DAC'21
11. QuantumNAS, HPCA'22
12. QuantumNAT, DAC'22
13. QOC, DAC'22
14. MCUNet, NeurIPS'20
15. MCUNet-V2, NeurIPS'21
16. TinyTL, NeurIPS'20
17. MCUNet-V3, Arxiv'22
18. DGC, ICLR'18
19. DGA, NeurIPS'21
20. PVCNN, NeurIPS'19
21. Fast-LiDARNet, ICRA'21
22. BEVFusion, Arxiv'22
23. TSM, ICCV'19
24. GAN Compression, CVPR'20
25. SpAtten, HPCA'21
26. SpArch, HPCA'20
27. PointAcc, Micro'20
28. TorchSparse, SysML'22



MIT AI Hardware Program

MIT Microsystems Technology Laboratories (SoE)

MIT Quest for Intelligence – Corporate (SCC)

Co-Leads: Jesús del Alamo and Aude Oliva

Internal Advisory Board Chair: Anantha Chandrakasan

TinyML and Efficient AI

MIT HAN LAB

Hardware, AI and Neural-nets

github.com/mit-han-lab

youtube.com/c/MITHANLab

songhan.mit.edu
tinyml.mit.edu

Sponsors:

IBM, Qualcomm, intel, NVIDIA, XILINX, NSF, amazon, Google, facebook, SONY, SAMSUNG, HYUNDAI, Ford

Media:

MIT Technology Review, IEEE SPECTRUM, WIRED, engadget, MIT News, VentureBeat