Accelerated AI Algorithms for Data-Driven Discovery

National Science Foundation

On-Device Training Under 256KB Memory

Song Han
MIT
Associate Professor

songhan.mit.edu
tinyml.mit.edu
@SongHan_MIT

Wei-Chen Wang
MIT
Postdoctoral Associate

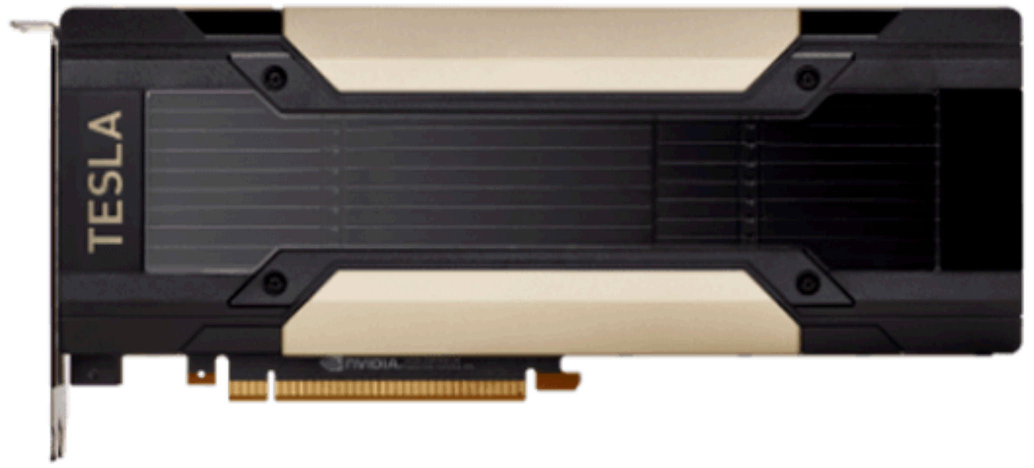
weichenwang.me
@RaymondWang0

Acknowledgment to NSF Grant OAC-2117997

https://mcunet.mit.edu

MCUNet for TinyML Inference

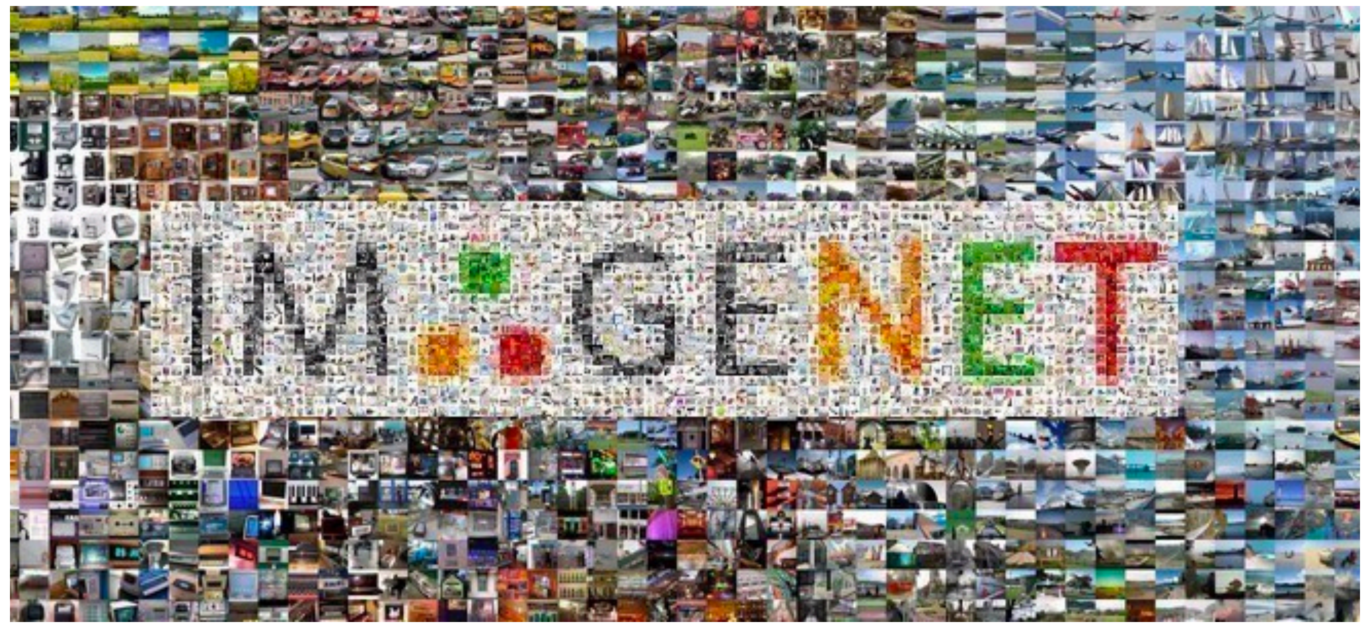
Addressing memory bottleneck issues



	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	32GB	4GB	320kB

Toy applications

MCUNet



Real-life applications

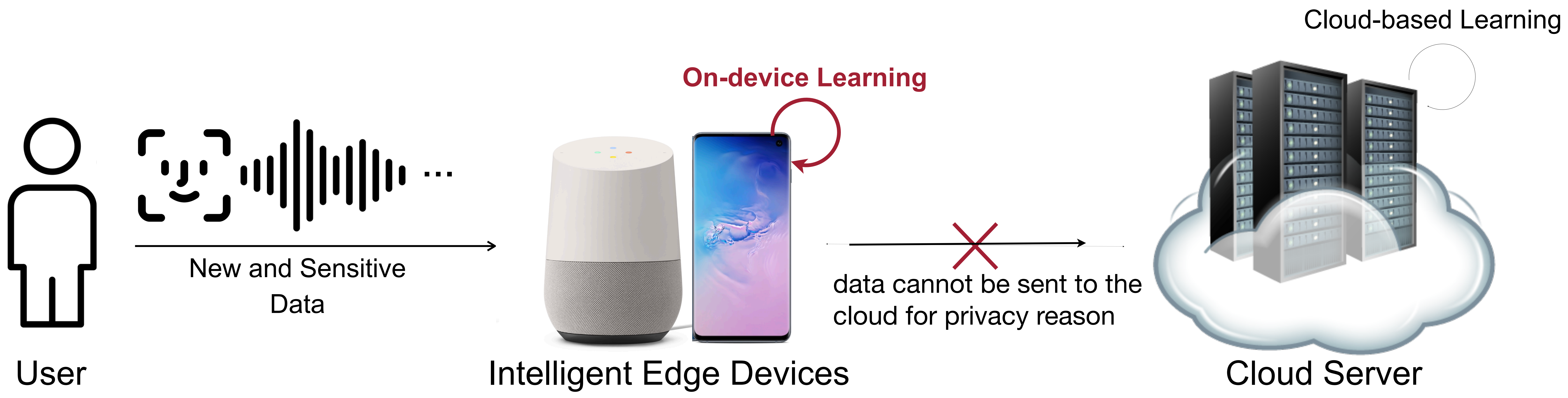
MCUNet: Tiny Deep Learning on IoT Devices (Lin et al., 2020)

Acknowledgment to NSF Grant OAC-2117997

Can We Learn on the Edge?

AI systems need to continually adapt to new data collected from the sensors
Not only inference, but also training

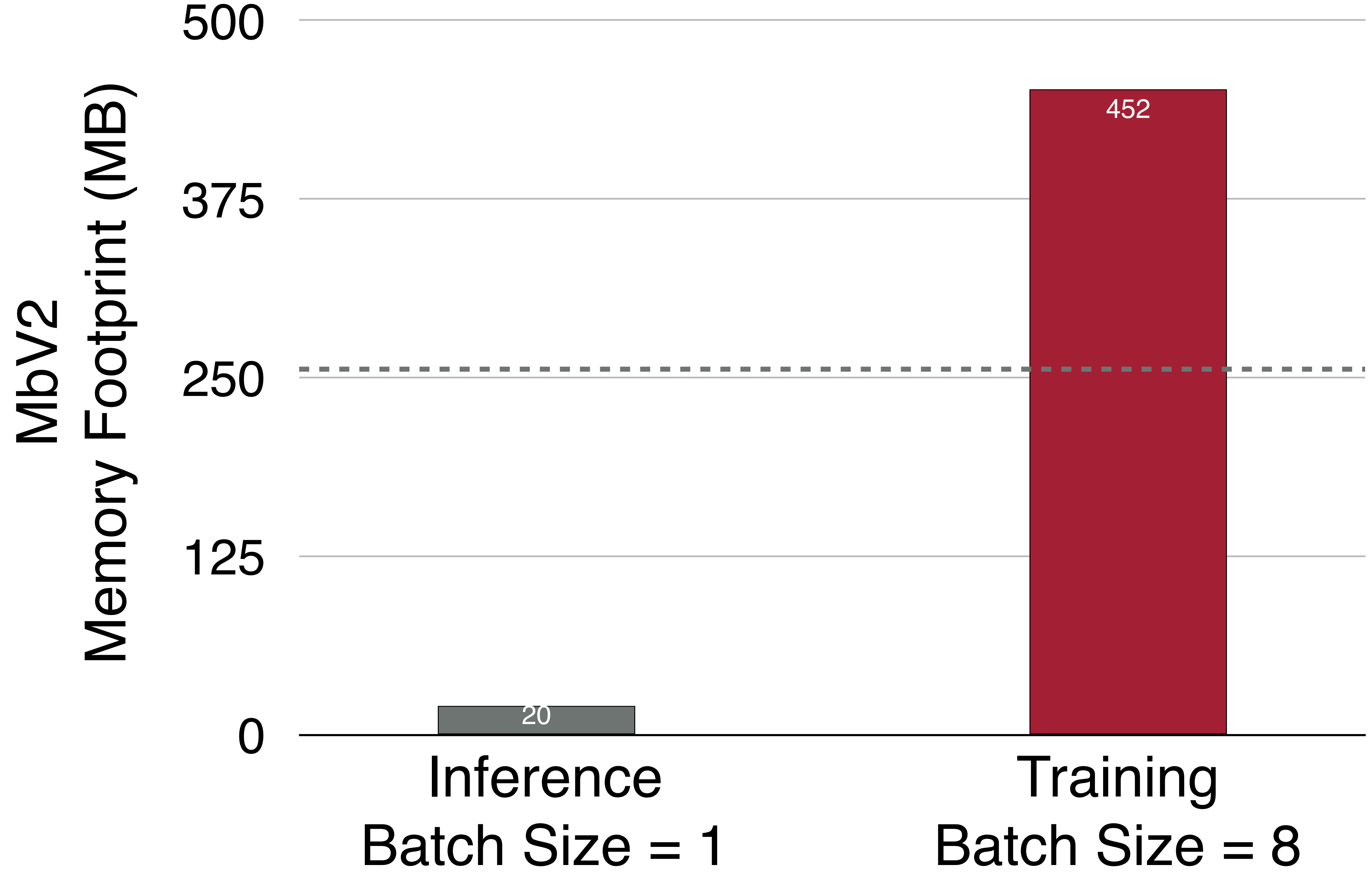
- On-device learning: **better privacy, lower cost, customization, life-long learning**
- Training is more **expensive** than inference, hard to fit edge hardware (limited memory)

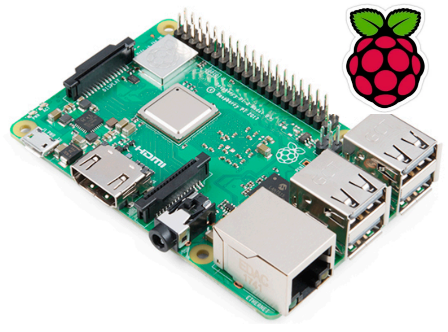


Acknowledgment to NSF Grant OAC-2117997

Training Memory is the Key Bottleneck

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





Raspberry Pi 1 DRAM
256MB

MCU: 256KB SRAM

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

On-Device Training Under 256KB Memory

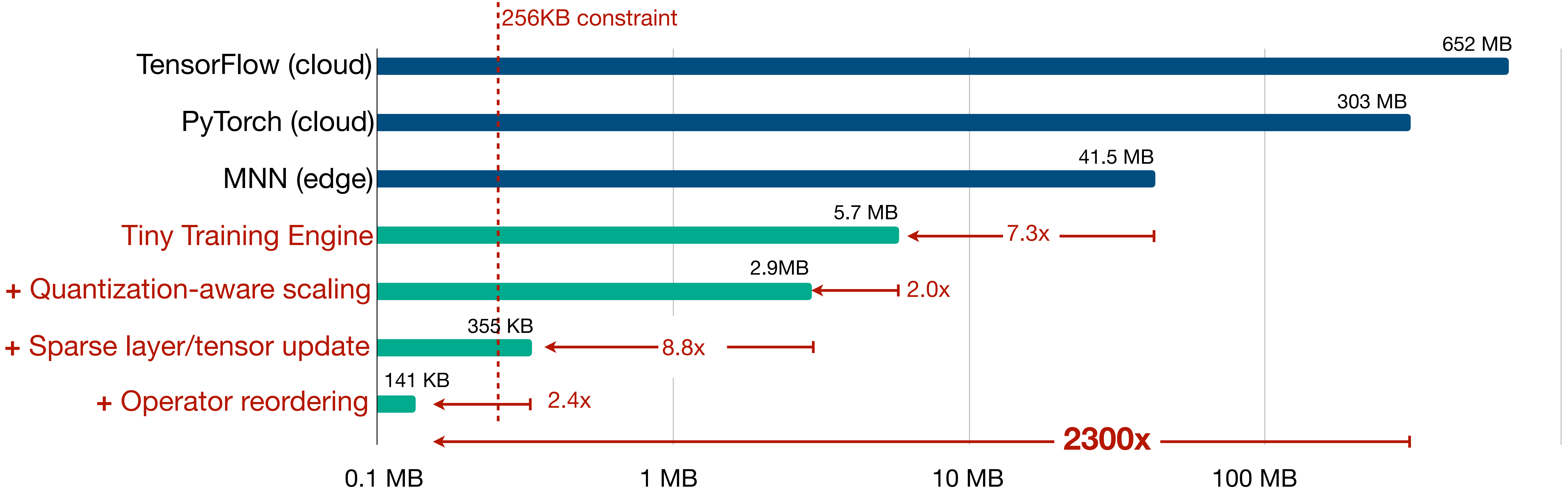
- **Training** is more expensive than **inference** due to back-propagation, making it hard to fit IoT devices (e.g., MCU only has 256KB SRAM).

On-Device Training Under 256KB Memory [Lin *et al.*, NeurIPS 2022]

Acknowledgment to NSF Grant OAC-2117997

On-Device Training Under 256KB Memory

- **Training** is more expensive than **inference** due to back-propagation, making it hard to fit IoT devices (e.g., MCU only has 256KB SRAM).

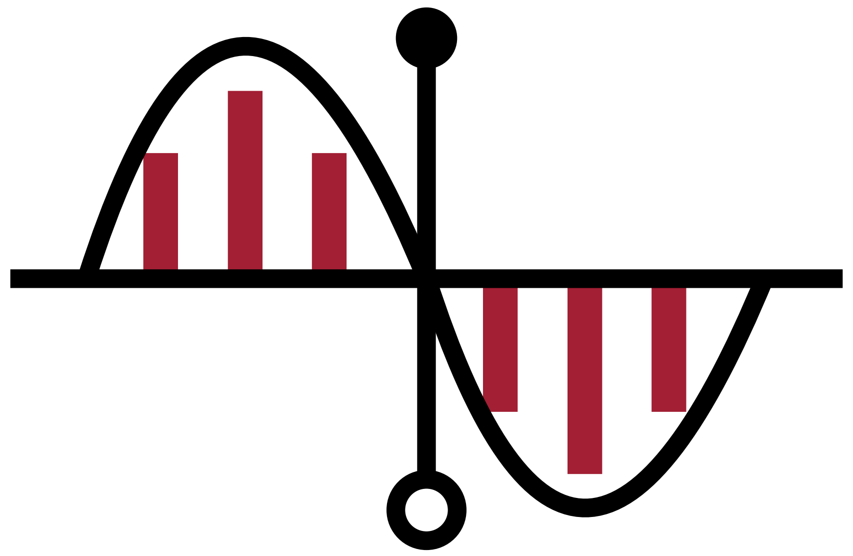


On-Device Training Under 256KB Memory [Lin *et al.*, NeurIPS 2022]

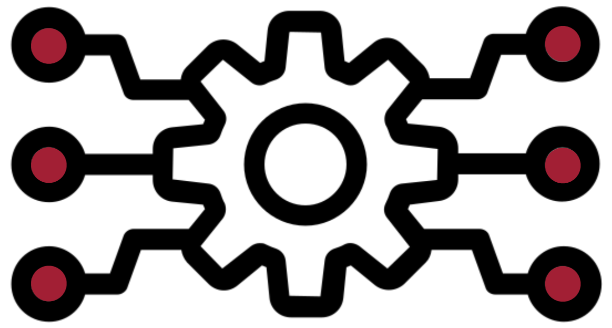
Acknowledgment to NSF Grant OAC-2117997

On-Device Training Under 256KB Memory

1. Sparse layer/tensor update



2. Quantization-aware scaling



3. Tiny Training Engine

Acknowledgment to NSF Grant OAC-2117997

On-Device Training Under 256KB Memory

1. Sparse layer/tensor update

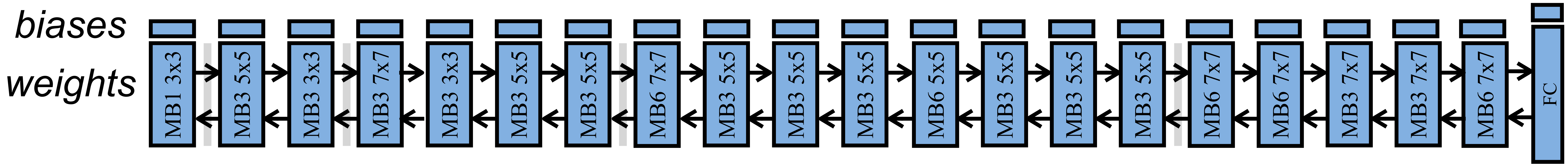
2. Quantization-aware scaling

3. Tiny Training Engine

Acknowledgment to NSF Grant OAC-2117997

1. Sparse Layer/Tensor Update

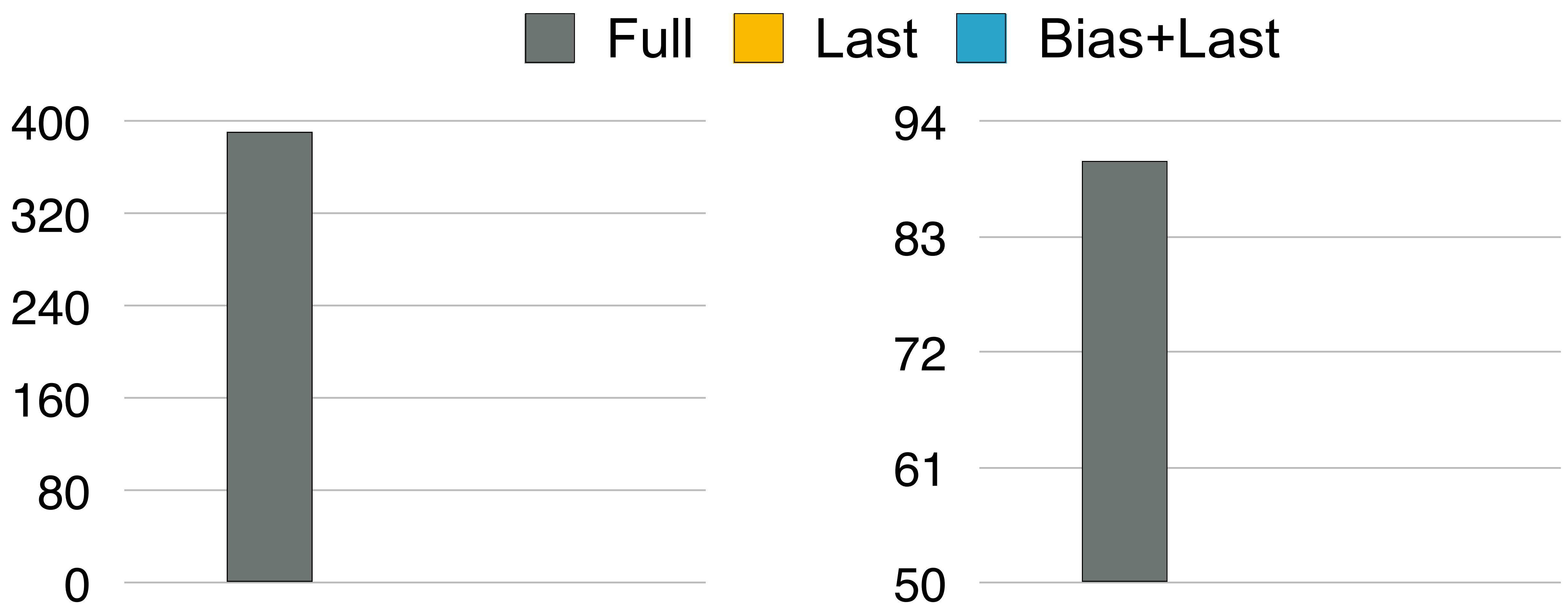
Full update



Model: ProxylessNAS-Mobile

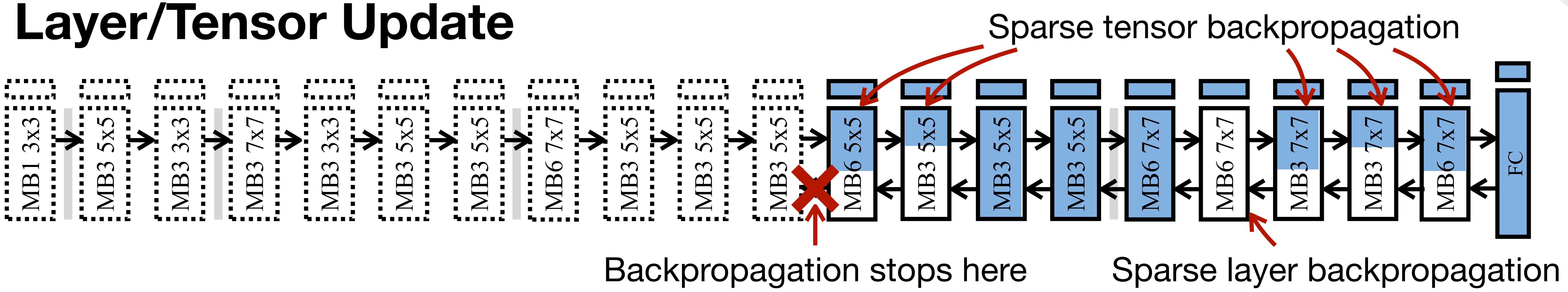
Updating the whole model is **too expensive**:

- Need to save all intermediate activation (quite large)
- Need to store the updated weights in SRAM (Flash is read-only)



1. Sparse Layer/Tensor Update

Sparse Layer/Tensor Update



Model: ProxylessNAS-Mobile

Updating the sparse tensors/layers

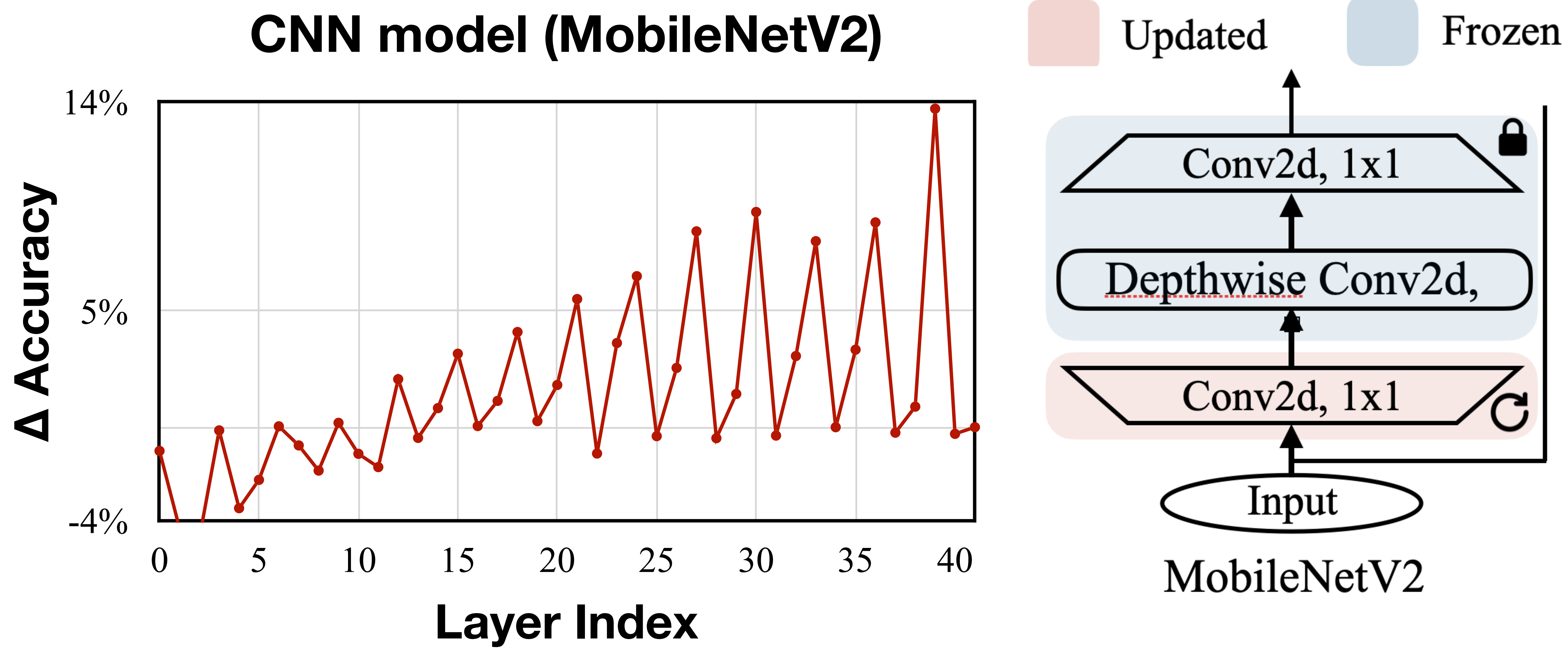
- Some layers are more important than others
- No need to backpropagate to the early layers
- Only need to store a subset of the activations

 Acknowledgment to NSF Grant OAC-2117997

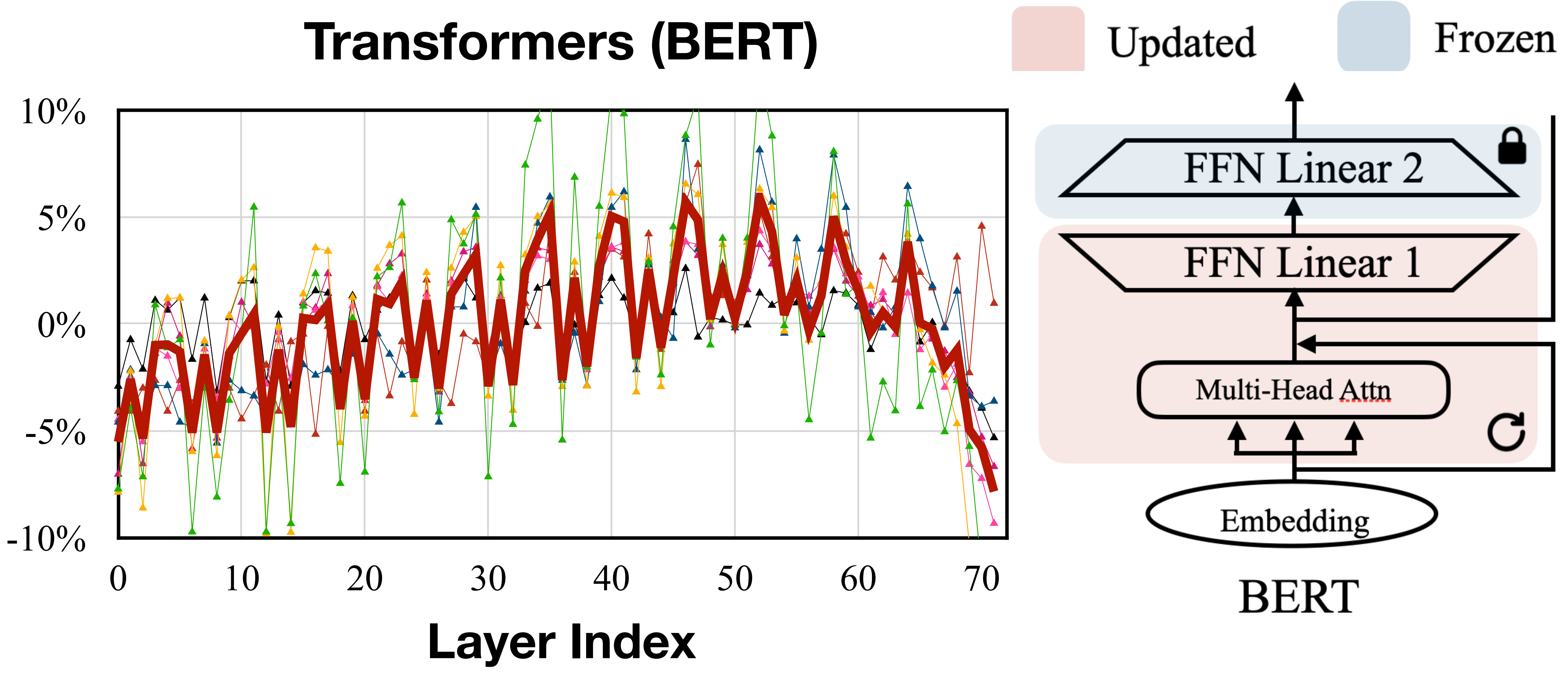
Find Layers to Update by Contribution Analysis

Some layers are more helpful than others

- Fine-tune each layer on a downstream dataset to measure accuracy improvement
- Generalize well to other datasets



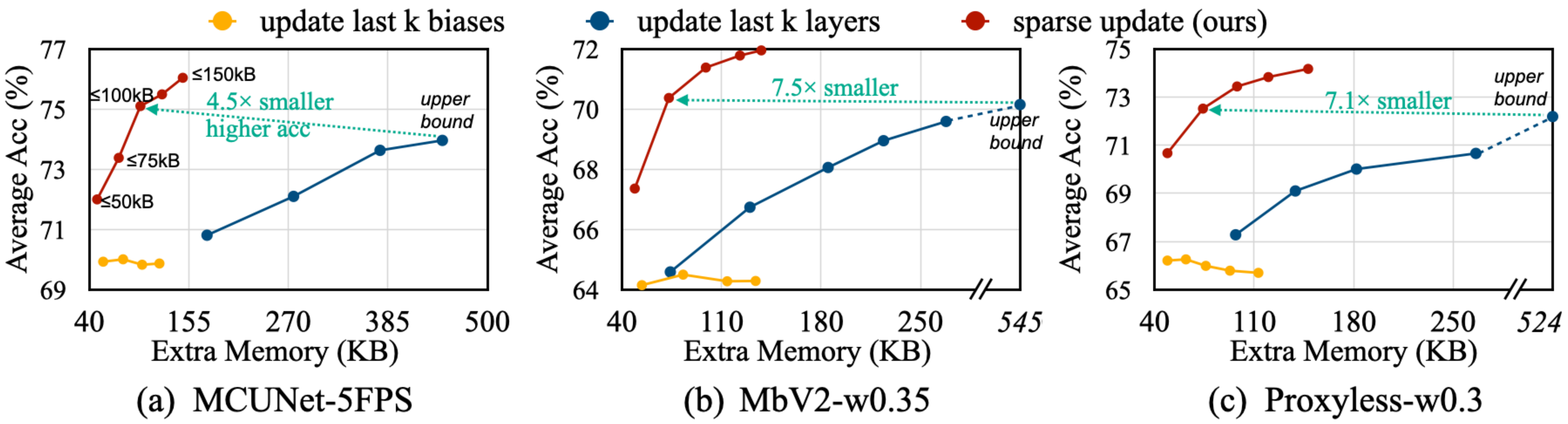
- **Later layers** are more important
- The **first point-wise conv** in each block contributes more



- **Middle layers** are more important
- **Attention and first FFN layers** contribute more

1. Sparse Layer/Tensor Update

Our method finds better trade-off

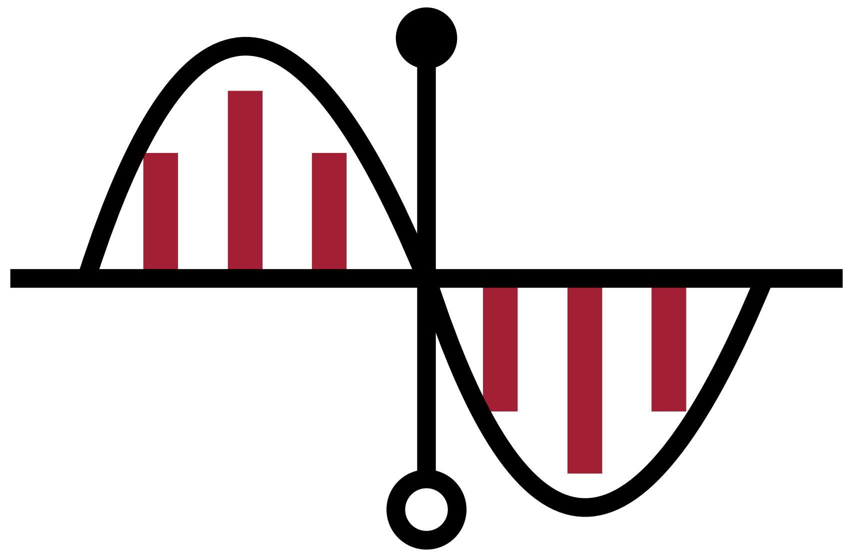


Sparse update can achieve higher transfer learning accuracy using **4.5-7.5x** smaller extra memory.

Acknowledgment to NSF Grant OAC-2117997

On-Device Training Under 256KB Memory

1. Sparse layer/tensor update



2. Quantization-aware scaling

3. Tiny Training Engine

Acknowledgment to NSF Grant OAC-2117997

2. Address Optimization Difficulty of Quantized Graphs

Quantized Training: lower memory and latency

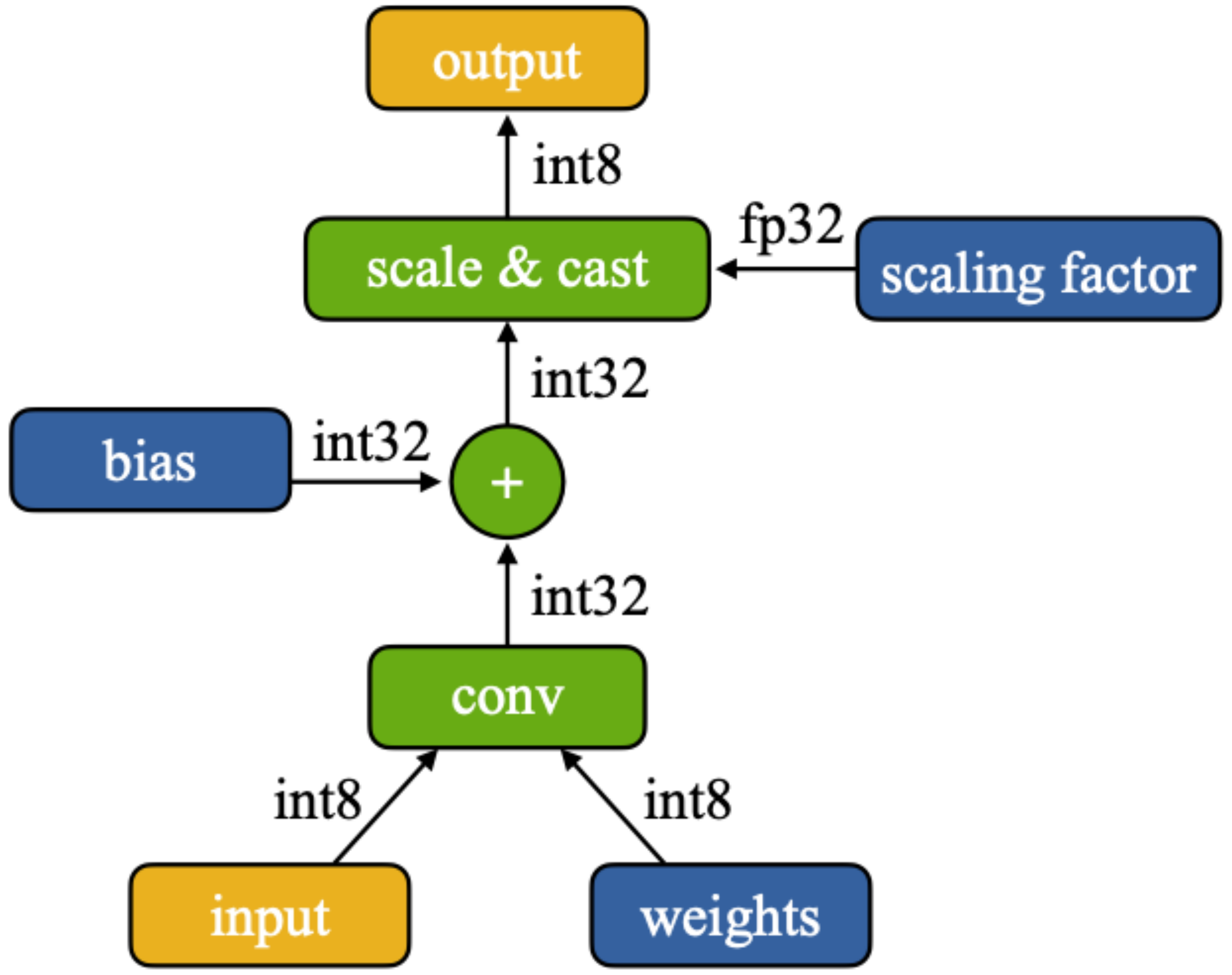
Full-precision training (32-bit)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Quantized training (2-bit)

1	-2	0	-1
-1	-1	-2	1
-2	1	-1	-2
1	-1	0	0

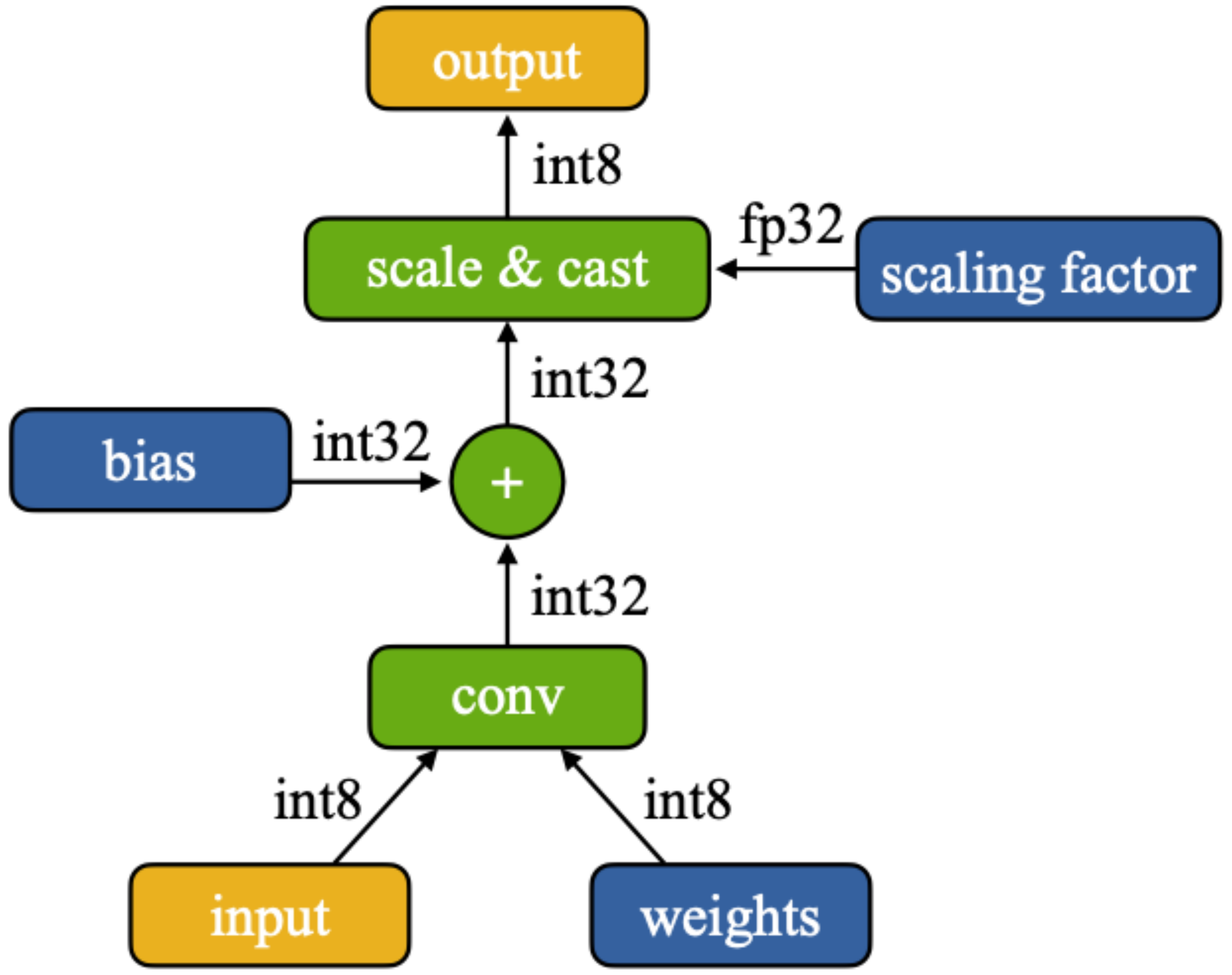
More efficient, but **difficult** to update



Real quantized graphs (integer tensors)

2. Address Optimization Difficulty of Quantized Graphs

But optimization is hard to quantization

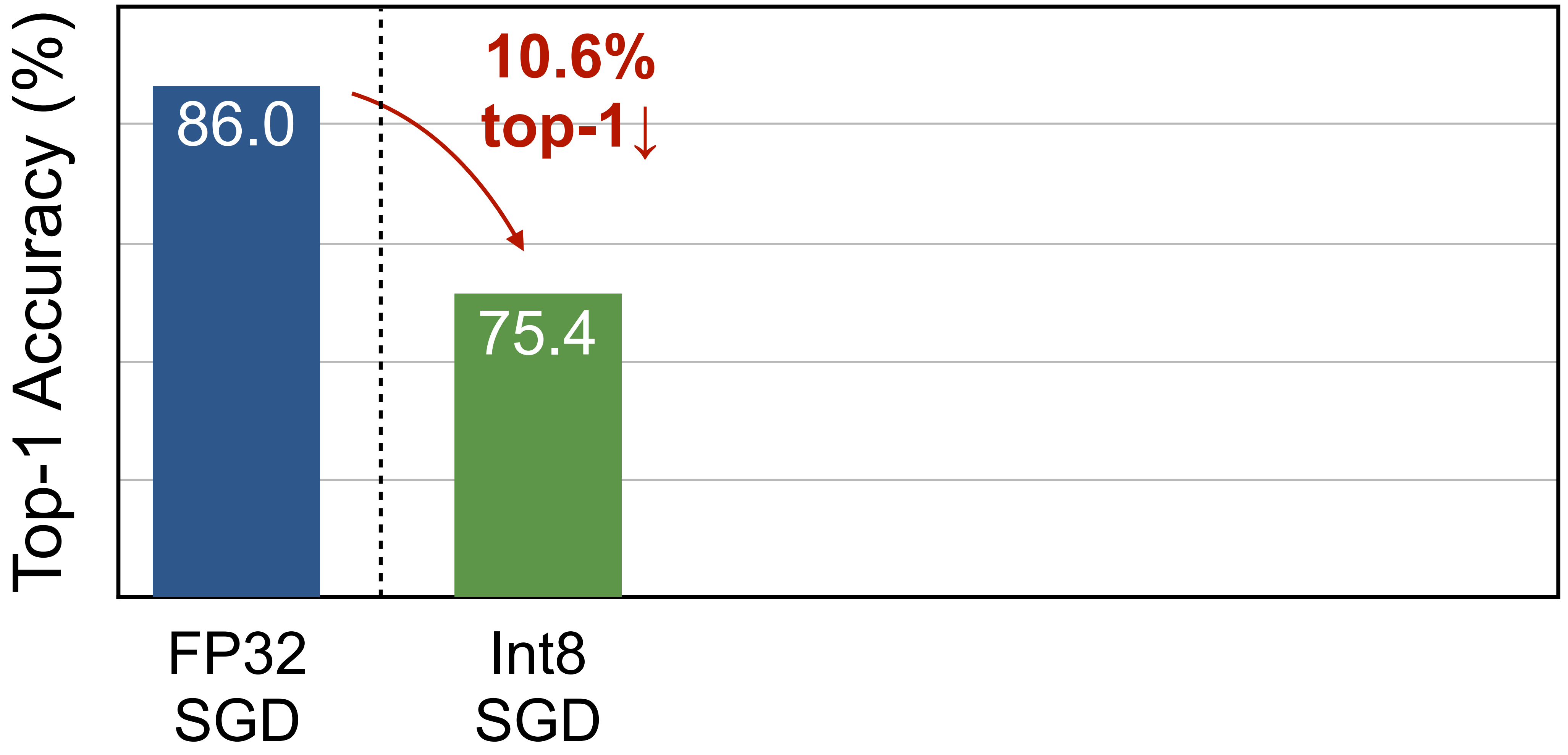


(a) Real Quantization

Making training difficult:

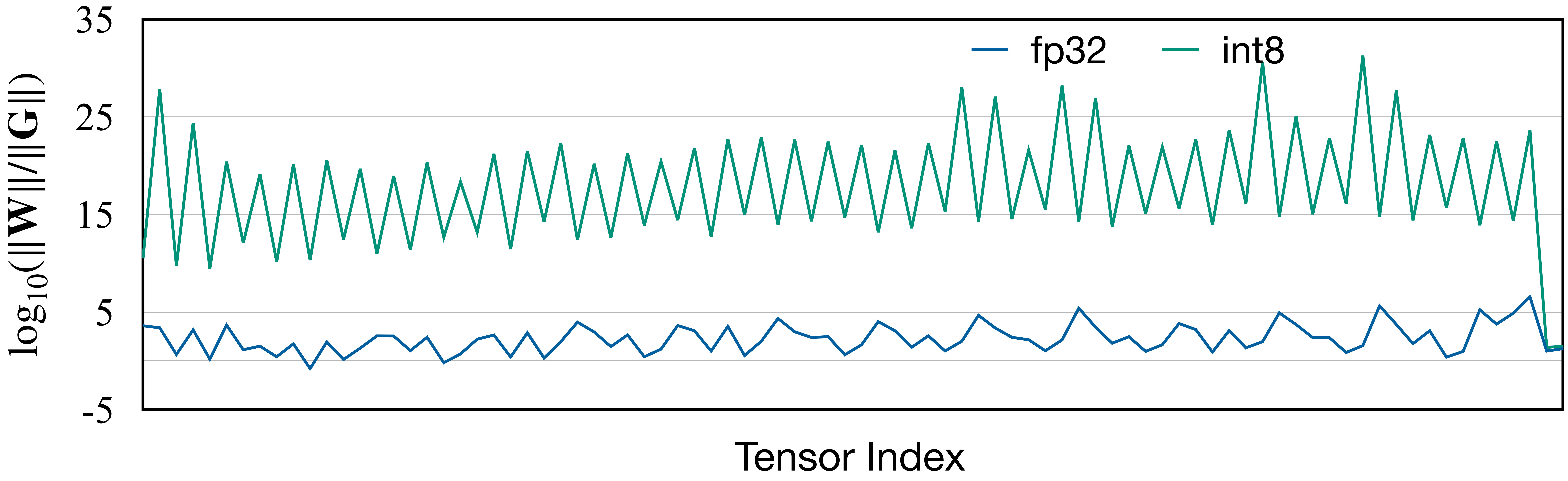
- Mixed precisions: int8/int32/fp32…
- Lack BatchNorm

Performance Comparison (average on 10 datasets)



2. Address Optimization Difficulty of Quantized Graphs

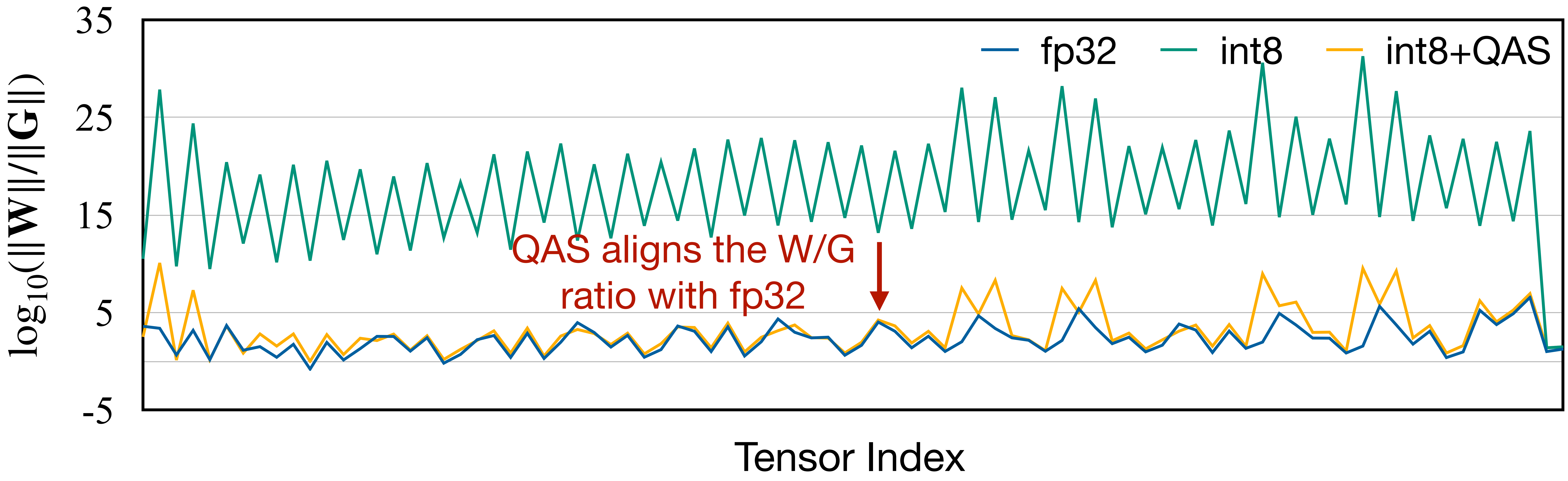
- Why is the training convergence worse?
- The scale of weight and gradients does not match in ***real quantized training!***



2. QAS: Quantization-Aware Scaling

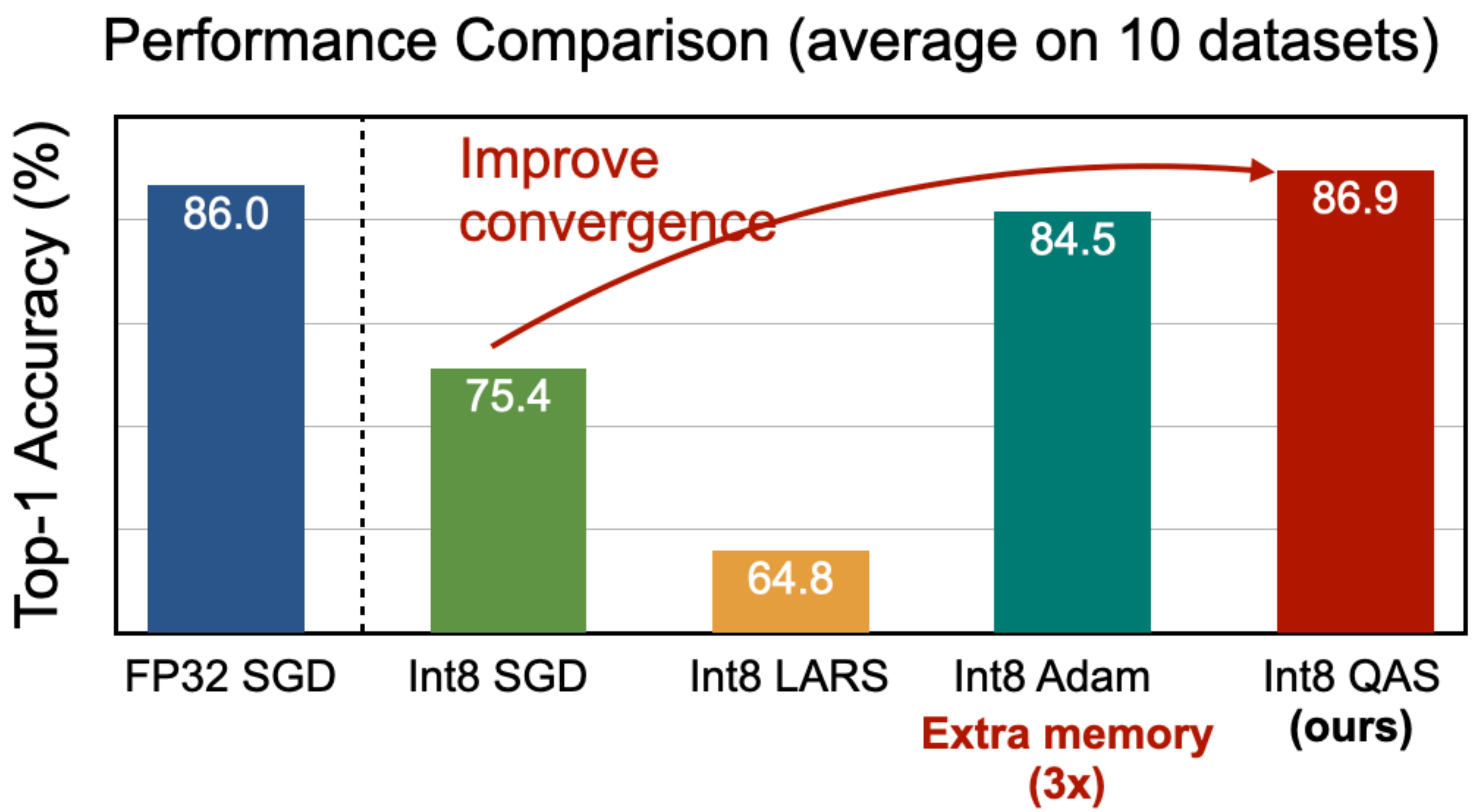
QAS addresses the optimization difficulty of quantized graphs

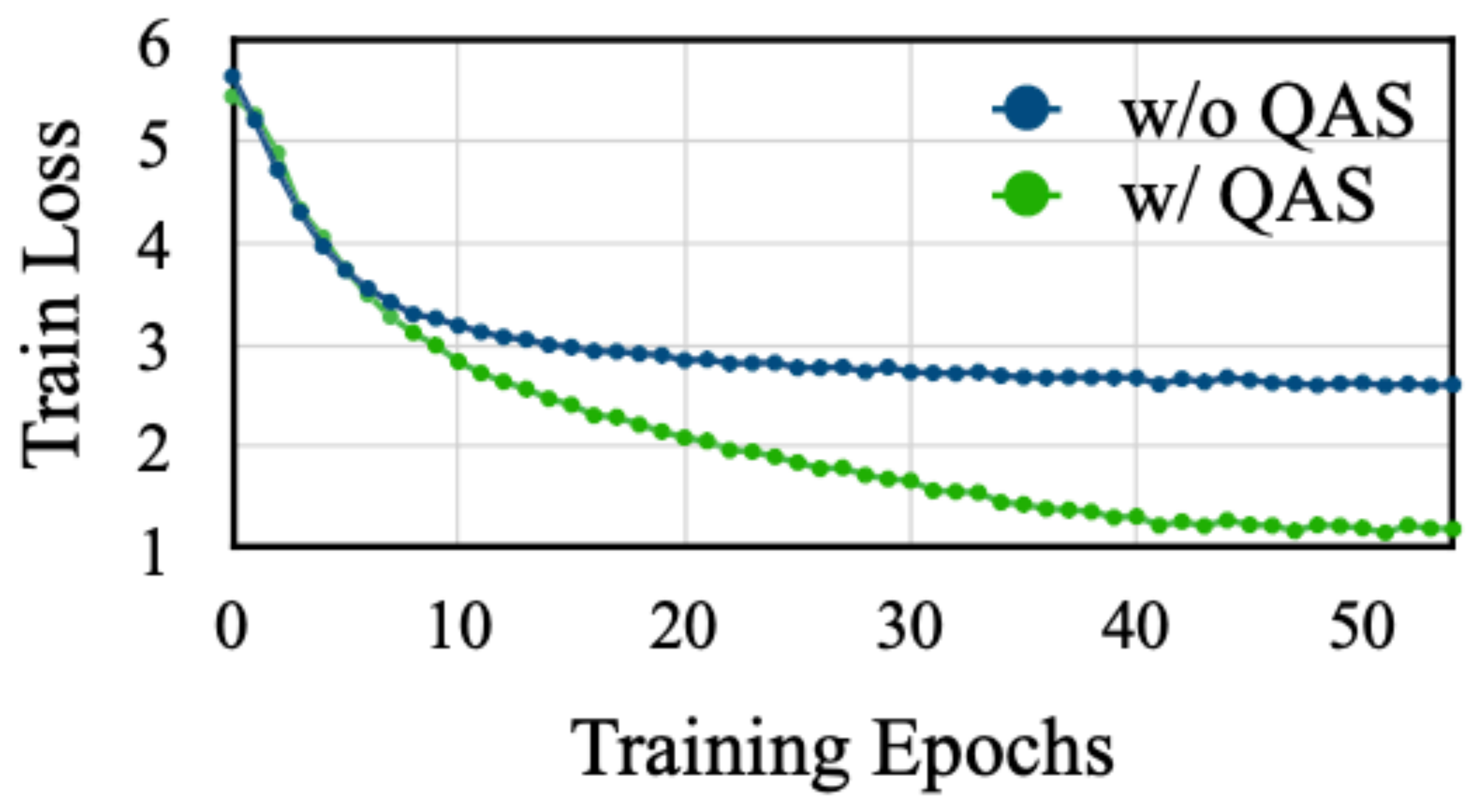
$$\tilde{\mathbf{G}}_{\bar{\mathbf{W}}} = \mathbf{G}_{\bar{\mathbf{W}}} \cdot s_{\mathbf{W}}^{-2}, \quad \tilde{\mathbf{G}}_{\bar{\mathbf{b}}} = \mathbf{G}_{\bar{\mathbf{b}}} \cdot s_{\mathbf{W}}^{-2} \cdot s_{\mathbf{x}}^{-2} = \mathbf{G}_{\bar{\mathbf{b}}} \cdot s^{-2}$$

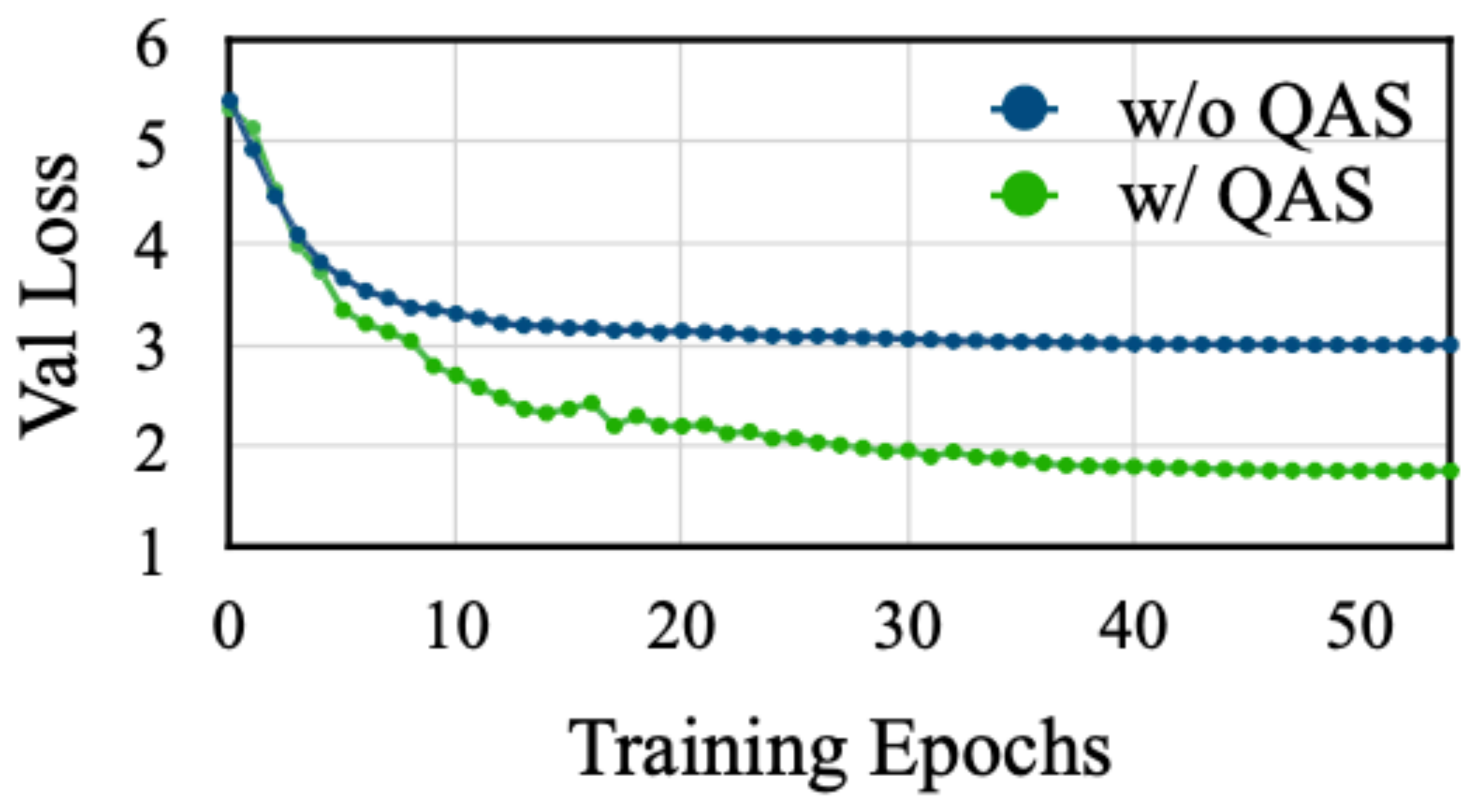


2. QAS: Quantization-Aware Scaling

QAS addresses the optimization difficulty of quantized graphs





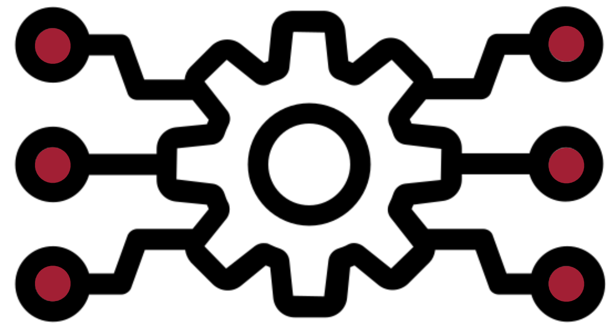


 Acknowledgment to NSF Grant OAC-2117997

On-Device Training Under 256KB Memory

1. Sparse layer/tensor update

2. Quantization-aware scaling



3. Tiny Training Engine

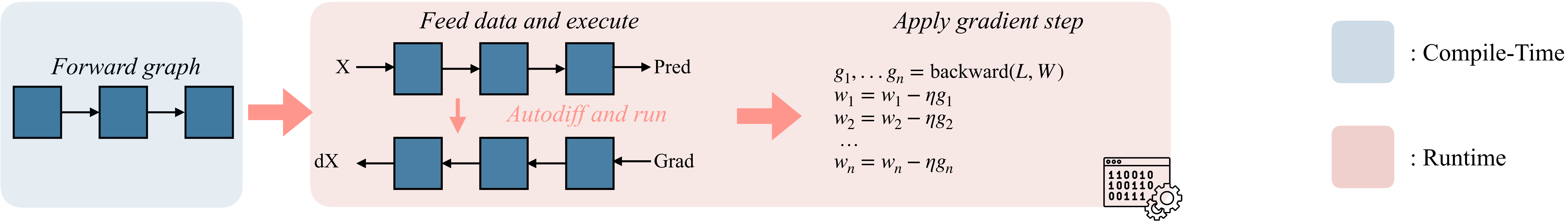
Acknowledgment to NSF Grant OAC-2117997

3. Tiny Training Engine (TTE)

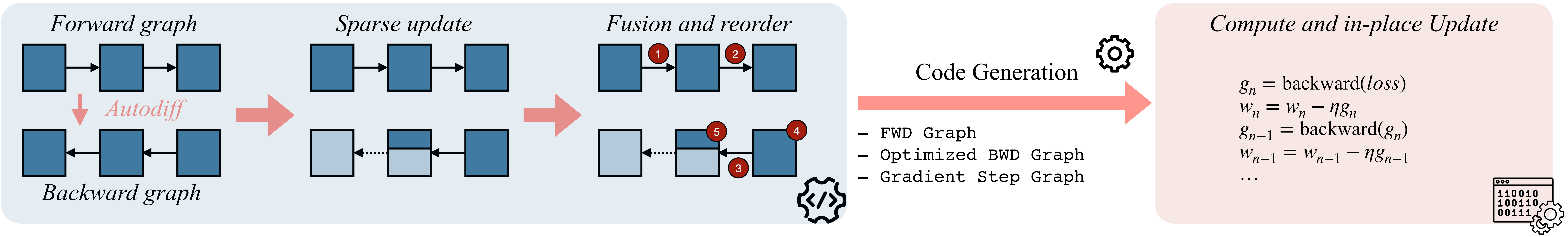
Existing frameworks cannot fit

- **Runtime** is heavy
  - Heavy dependencies and large binary size (>**100MB** static memory)
  - Auto-diff at runtime; low edge efficiency
- **Memory** is heavy
  - A lot of intermediate (and unused) buffers
  - Has to compute full gradients

3. Tiny Training Engine (TTE)



Conventional training framework performs most tasks at runtime.

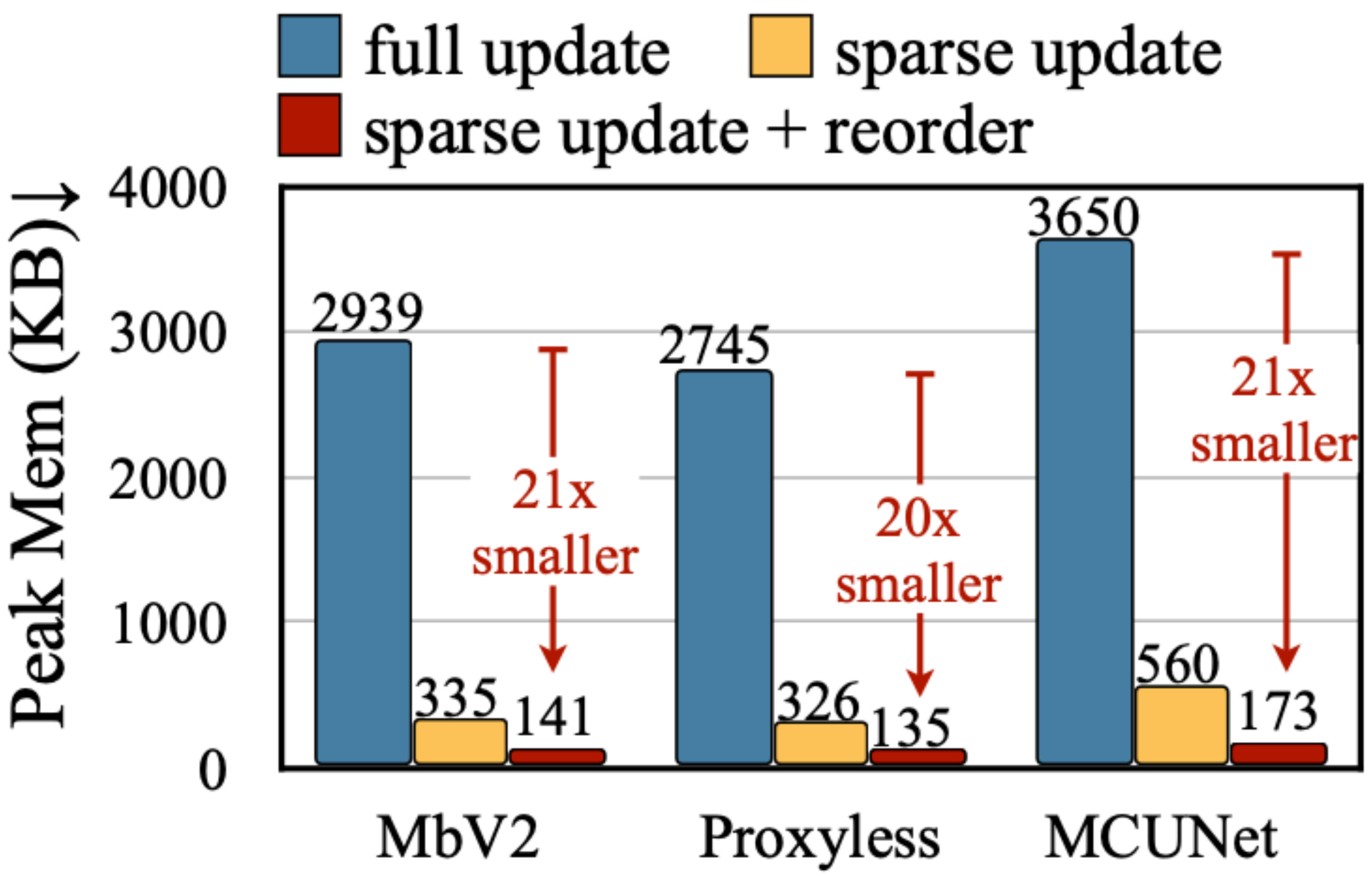


Tiny Training Engine (ours) **separate** the environment of runtime and compile time.

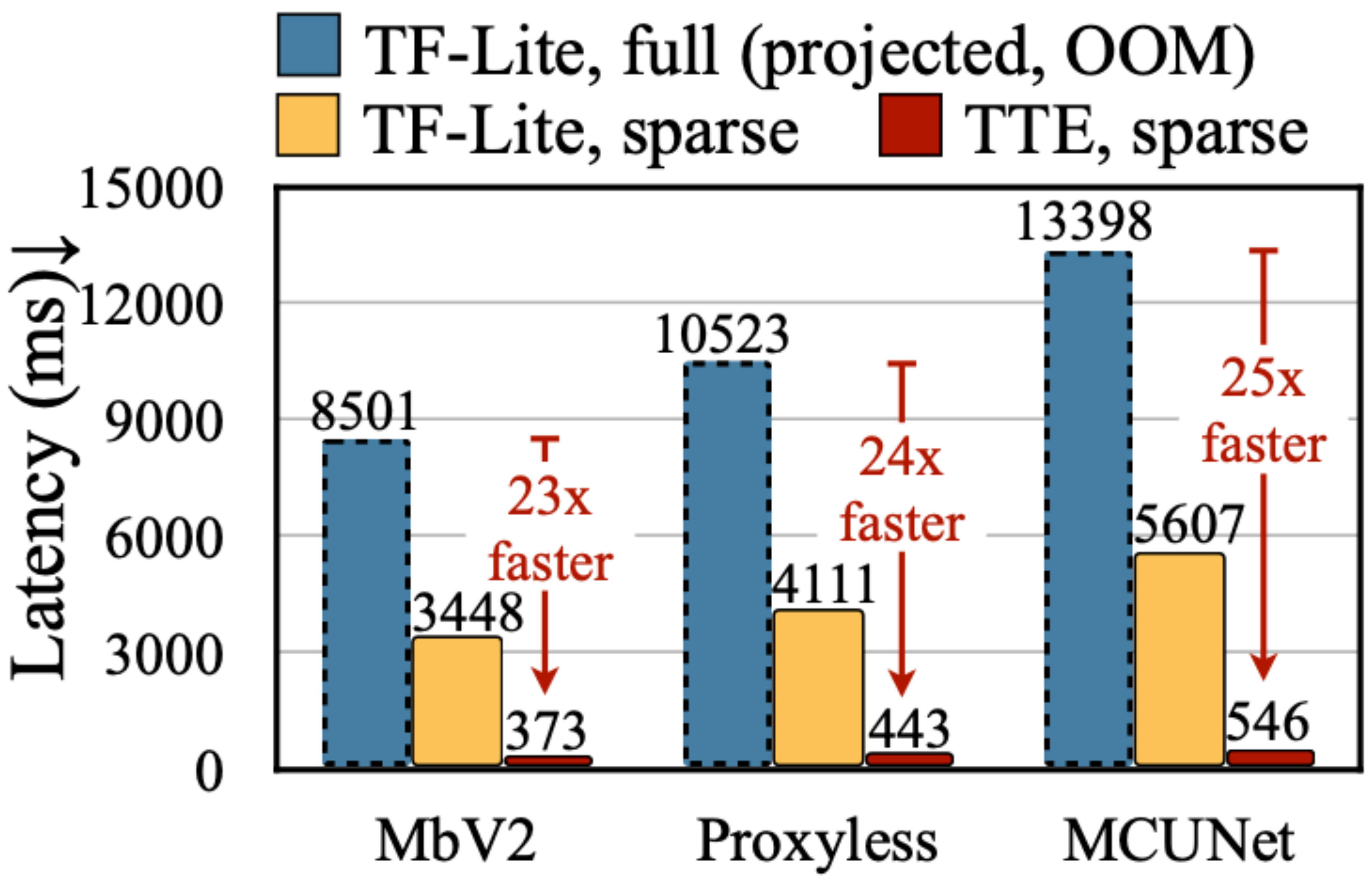
Acknowledgment to NSF Grant OAC-2117997

3. Tiny Training Engine (TTE)

Smaller memory usage, faster training speed



(a) Peak memory *vs.* models



(c) Training latency *vs.* models

20x smaller memory

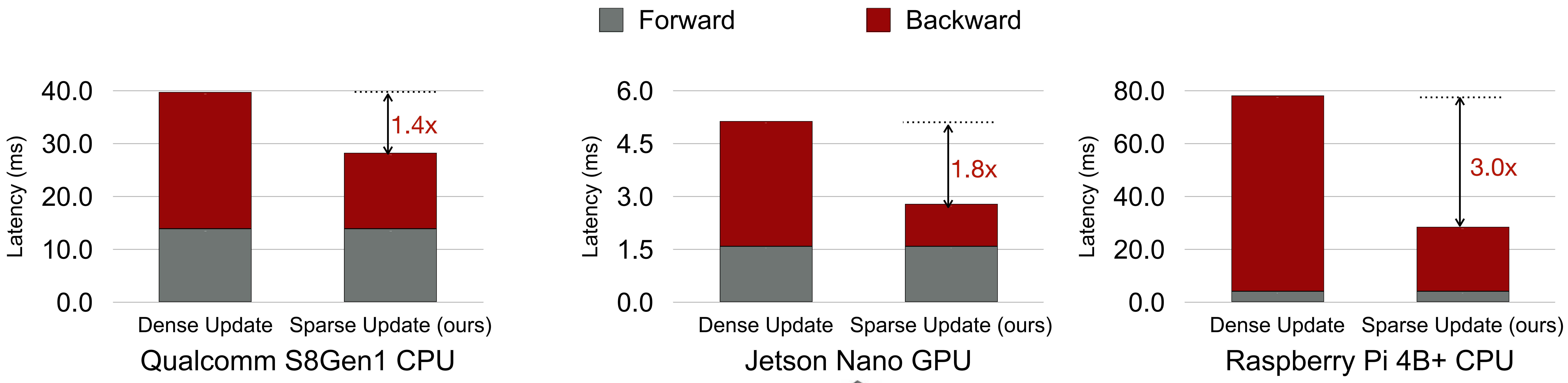
23x faster speed

Acknowledgment to NSF Grant OAC-2117997

3. Tiny Training Engine (TTE)

Accelerated AI Algorithms for Data-Driven Discovery

Scalable to diverse edge hardware platforms



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

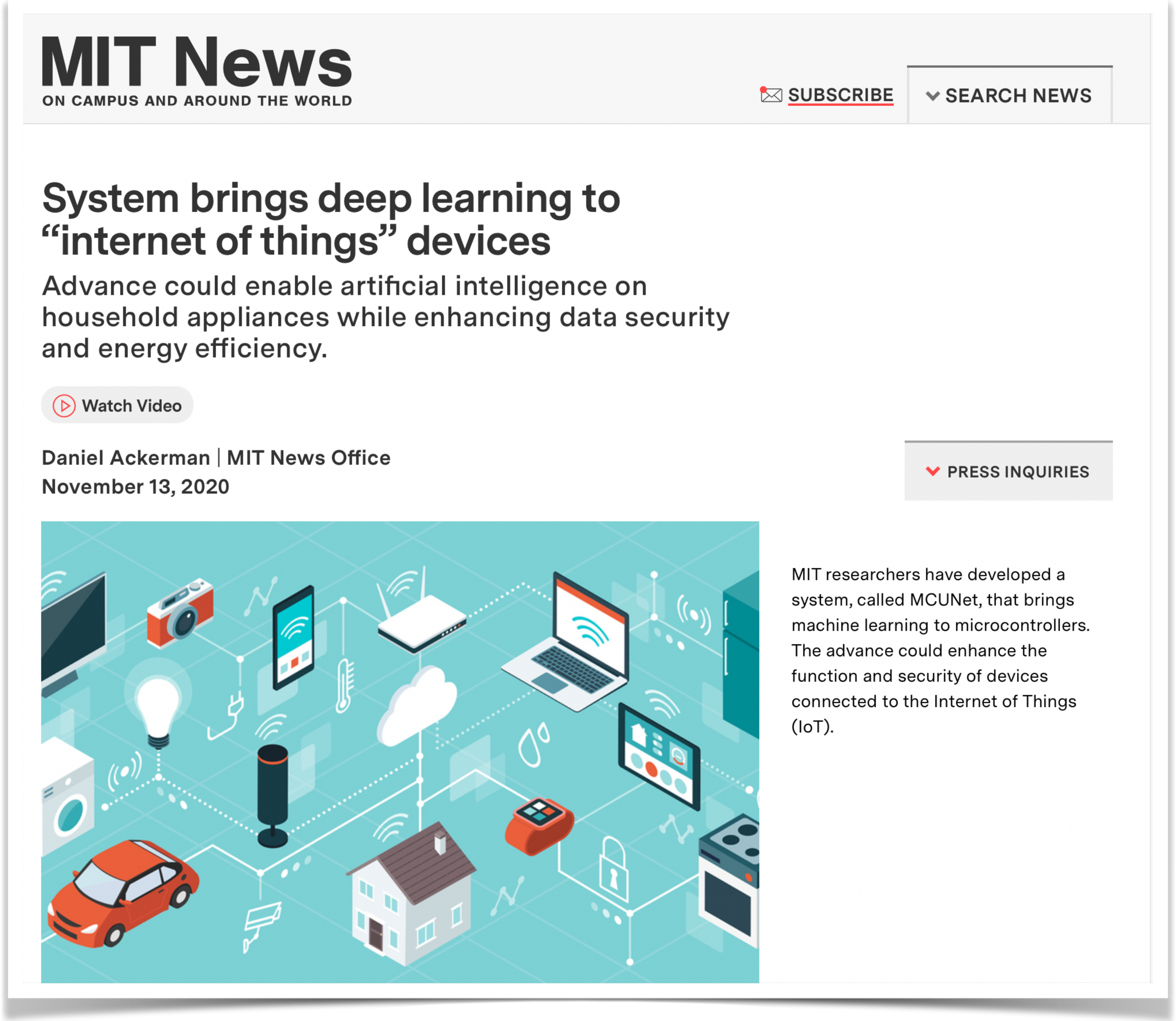
The benchmark model is MobilenetV2-035 with input resolution 128x128.

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

Acknowledgment to NSF Grant OAC-2117997

Coverage

Media Report



MIT News
ON CAMPUS AND AROUND THE WORLD
SUBSCRIBE | SEARCH NEWS

System brings deep learning to "internet of things" devices

Advance could enable artificial intelligence on household appliances while enhancing data security and energy efficiency.

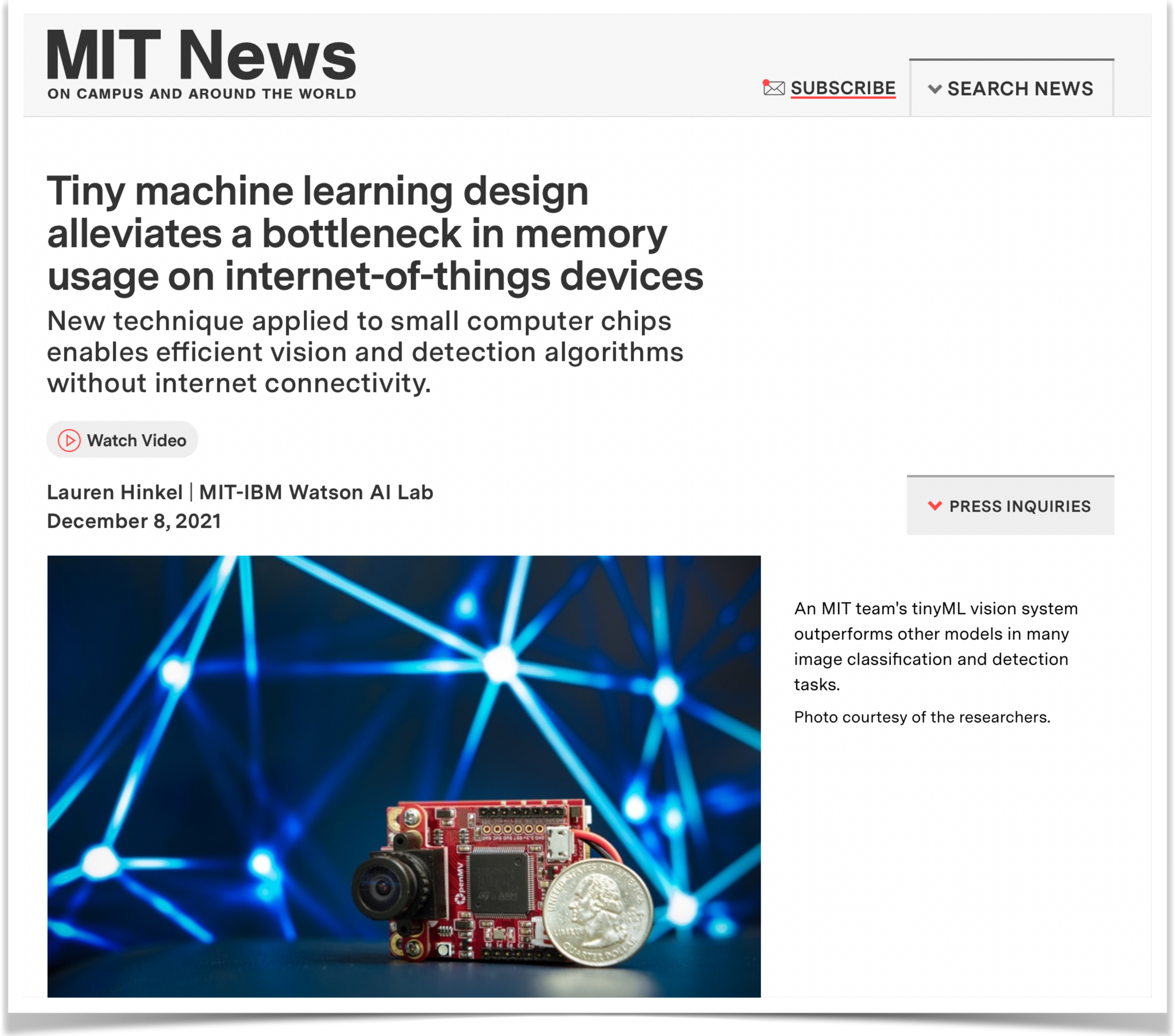
Watch Video

Daniel Ackerman | MIT News Office
November 13, 2020

PRESS INQUIRIES

MIT researchers have developed a system, called MCUNet, that brings machine learning to microcontrollers. The advance could enhance the function and security of devices connected to the Internet of Things (IoT).

(Homepage highlight)



MIT News
ON CAMPUS AND AROUND THE WORLD
SUBSCRIBE | SEARCH NEWS

Tiny machine learning design alleviates a bottleneck in memory usage on internet-of-things devices

New technique applied to small computer chips enables efficient vision and detection algorithms without internet connectivity.

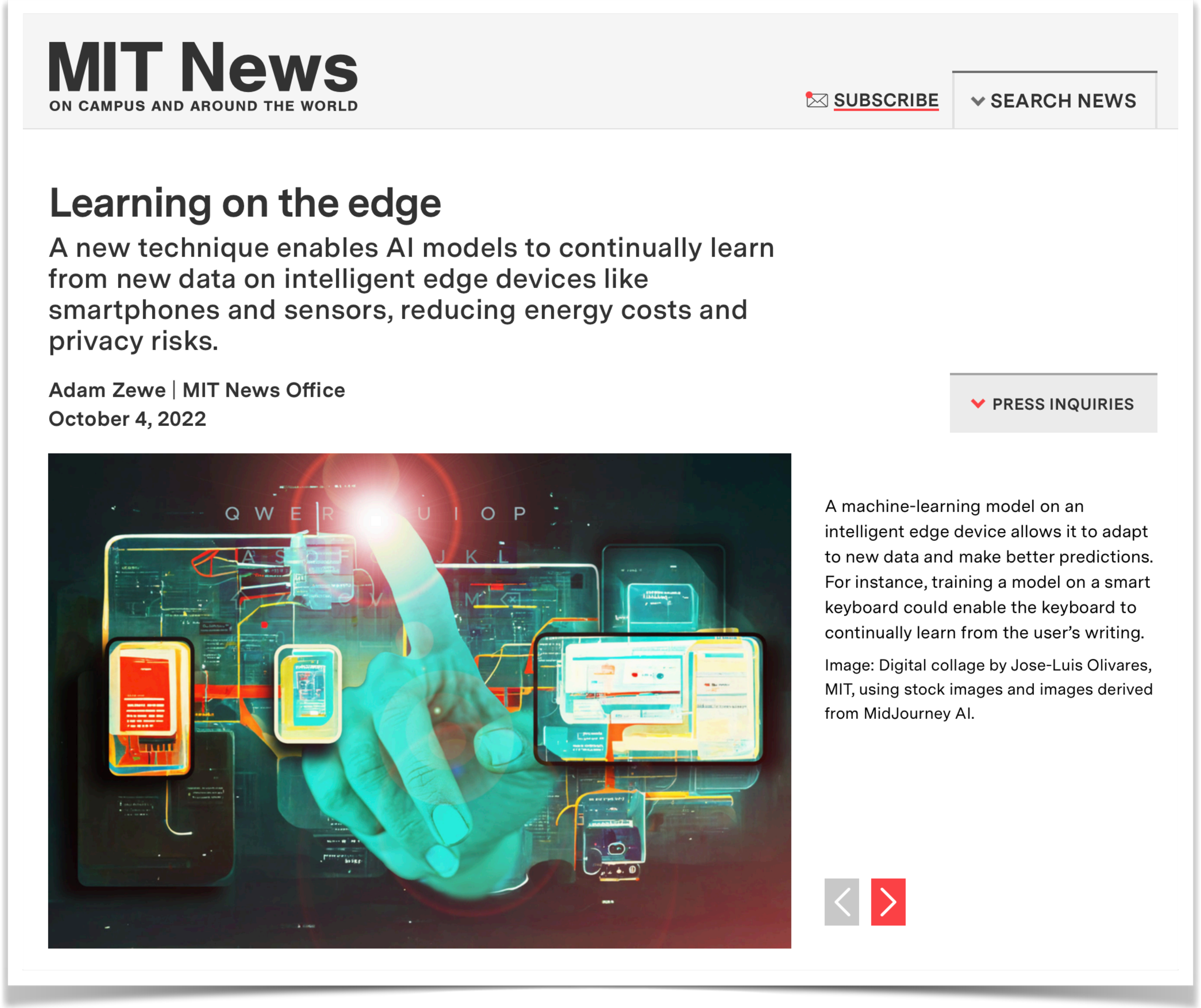
Watch Video

Lauren Hinkel | MIT-IBM Watson AI Lab
December 8, 2021

PRESS INQUIRIES

An MIT team's tinyML vision system outperforms other models in many image classification and detection tasks.

Photo courtesy of the researchers.



MIT News
ON CAMPUS AND AROUND THE WORLD
SUBSCRIBE | SEARCH NEWS

Learning on the edge

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.

Adam Zewe | MIT News Office
October 4, 2022

PRESS INQUIRIES

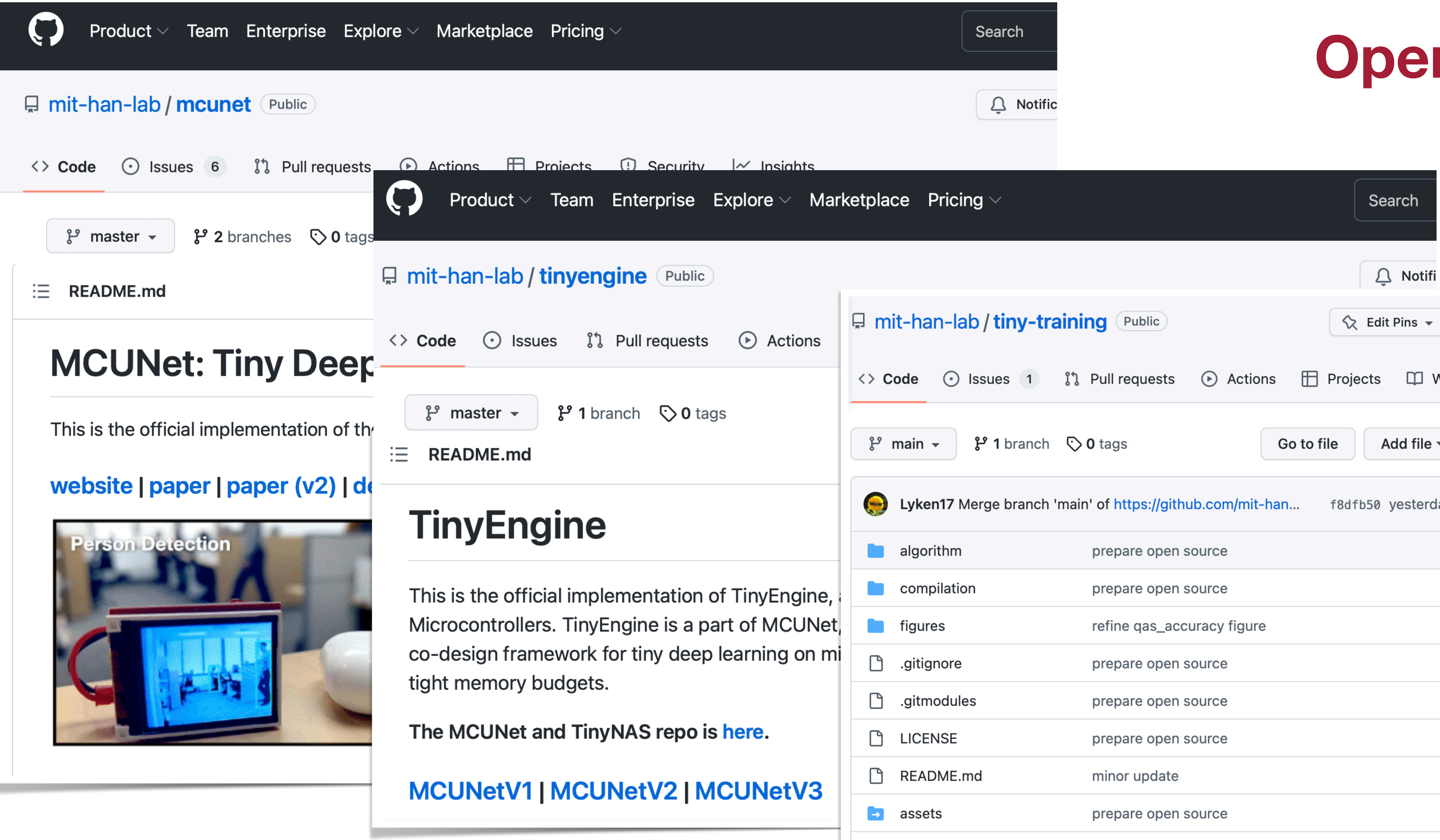
A machine-learning model on an intelligent edge device allows it to adapt to new data and make better predictions. For instance, training a model on a smart keyboard could enable the keyboard to continually learn from the user's writing.

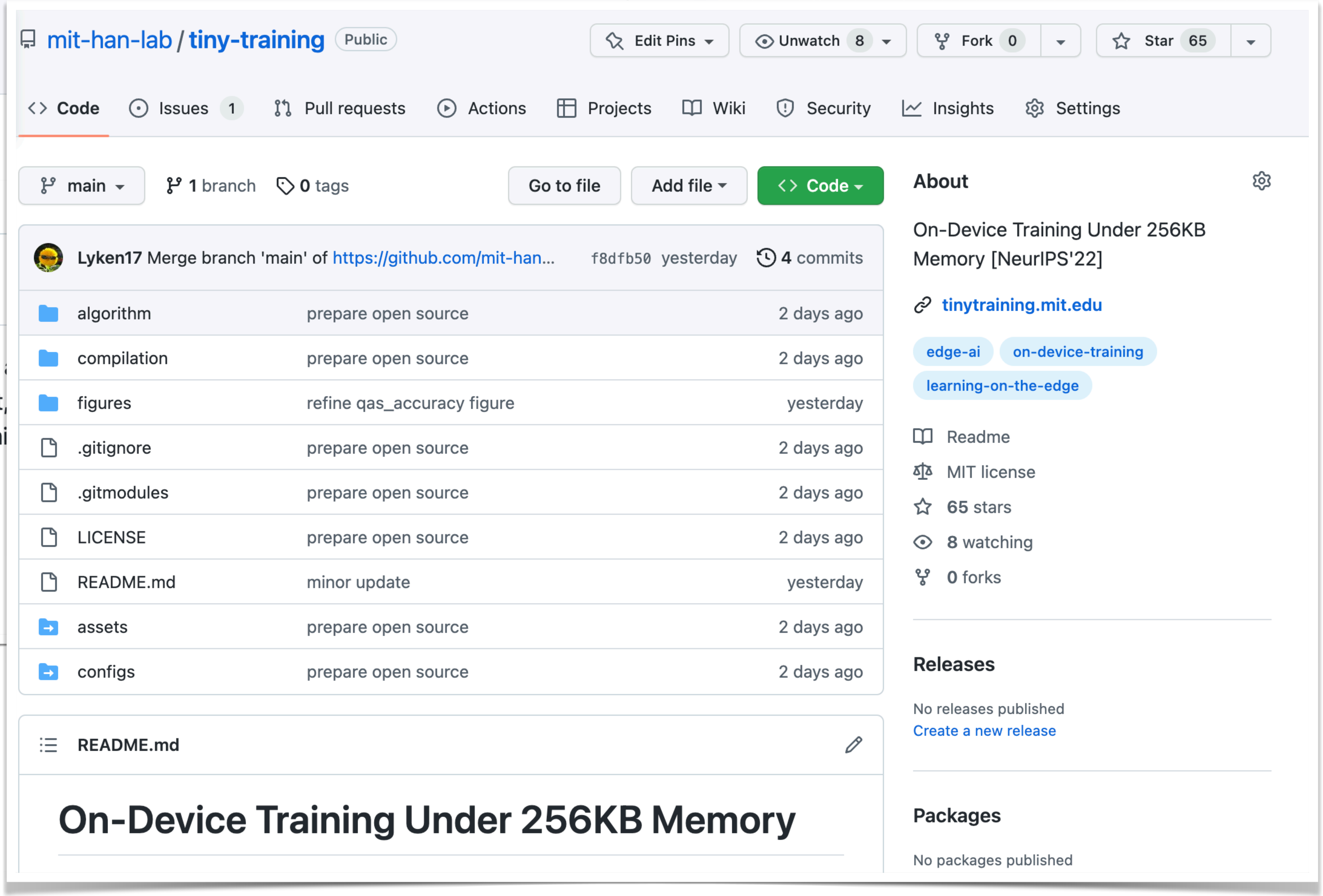
Image: Digital collage by Jose-Luis Olivares, MIT, using stock images and images derived from MidJourney AI.

(Homepage highlight)

MCUNet: Tiny Deep Learning on IoT Devices [Lin *et al.*, NeurIPS 2020]
MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning [Lin *et al.*, NeurIPS 2021]
On-Device Training Under 256KB Memory [Lin *et al.*, NeurIPS 2022]

Open Source





Sign up here to get updates!

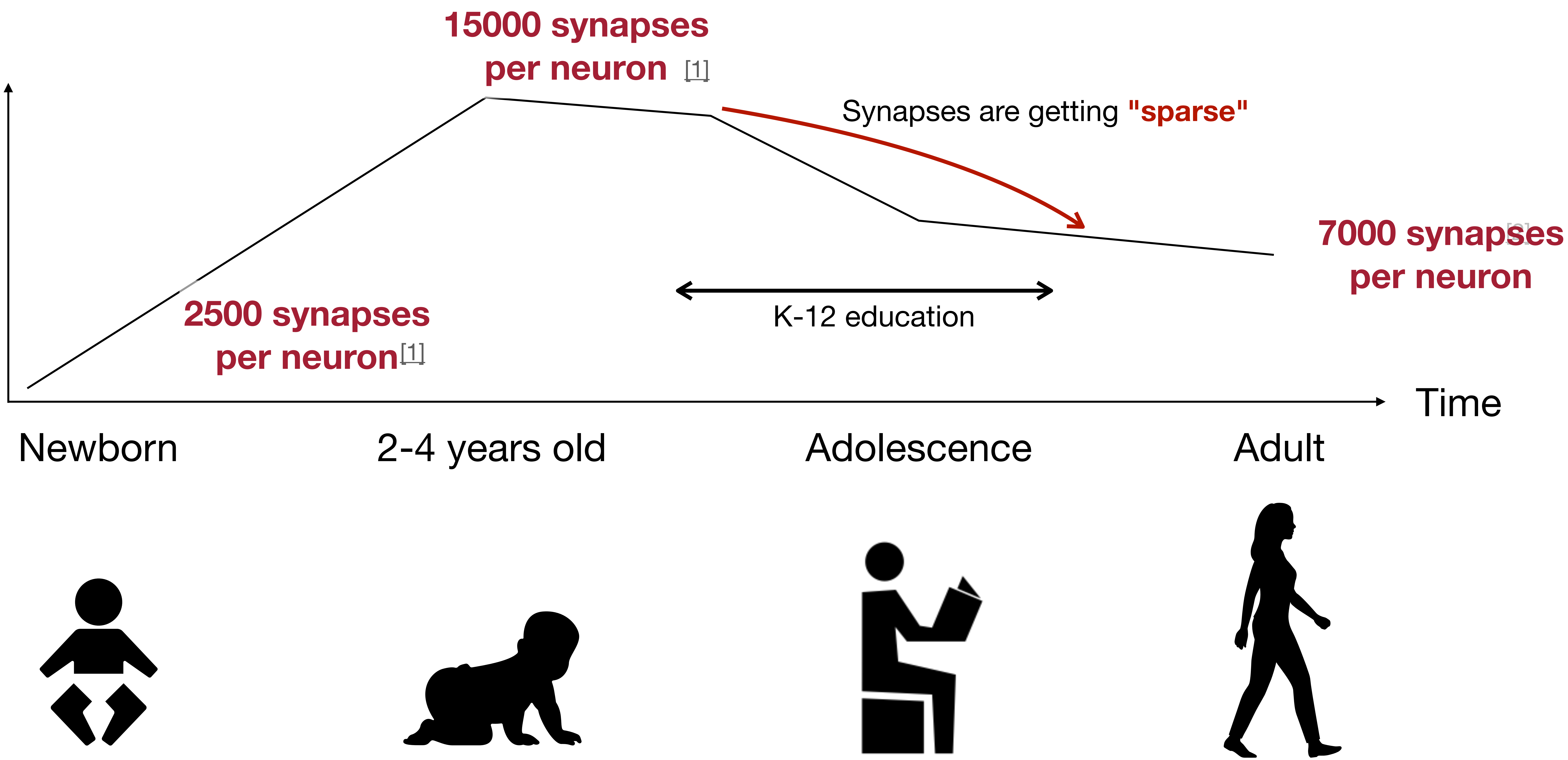
https://forms.gle/UW1uUmnfk1k6UJPPA

Thank you!

Appendix

1. Sparse Layer/Tensor Update

Updated synapses are sparse

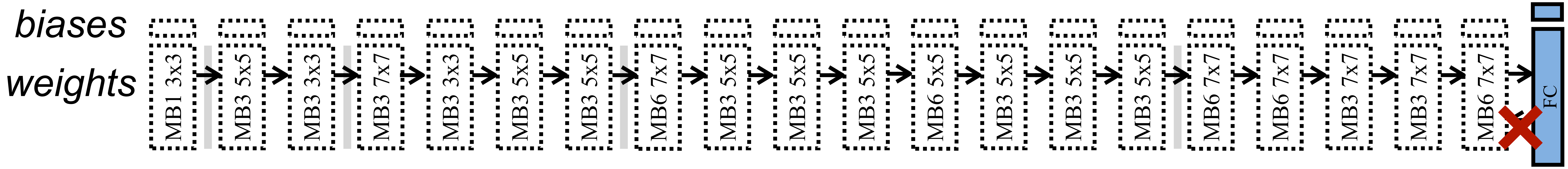


Do We Have Brain to Spare? [Drachman DA, Neurology 2004]
Peter Huttenlocher (1931–2013) [Walsh, C. A., Nature 2013]

Data Source: 1, 2
Slide Inspiration: Alila Medical Media

1. Sparse Layer/Tensor Update

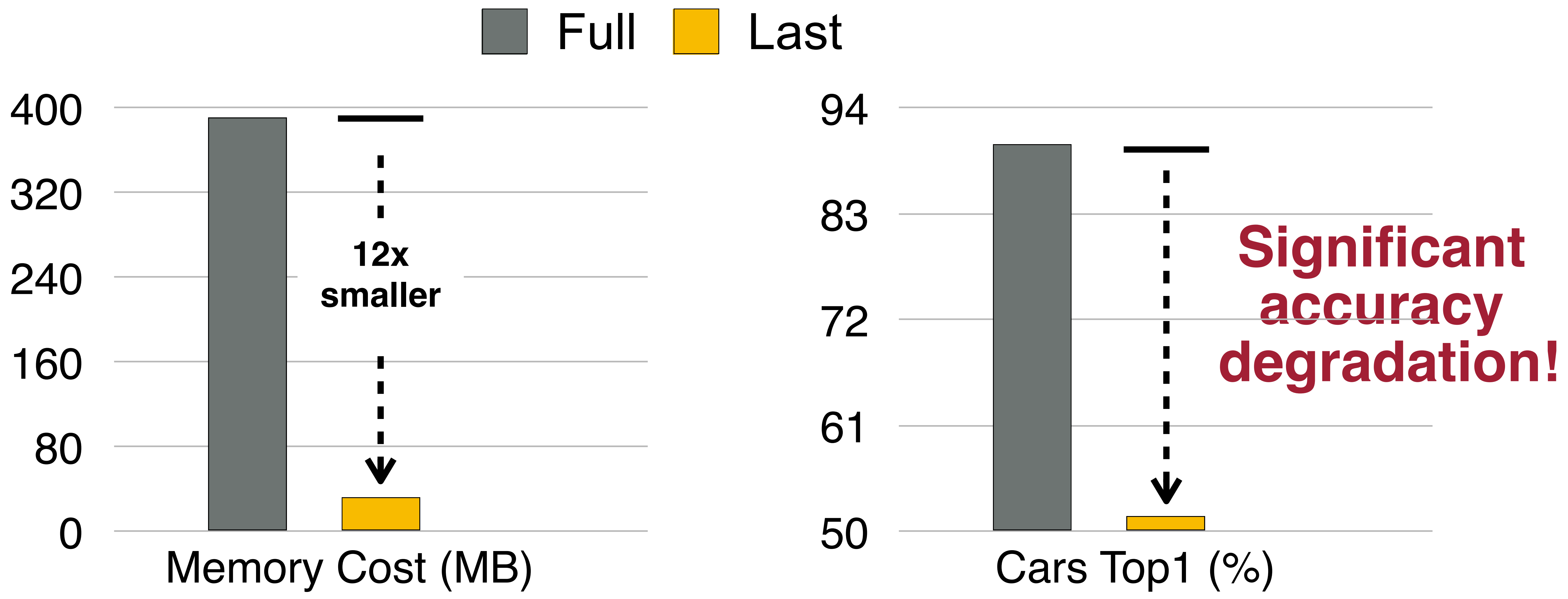
Last layer update



Model: ProxylessNAS-Mobile

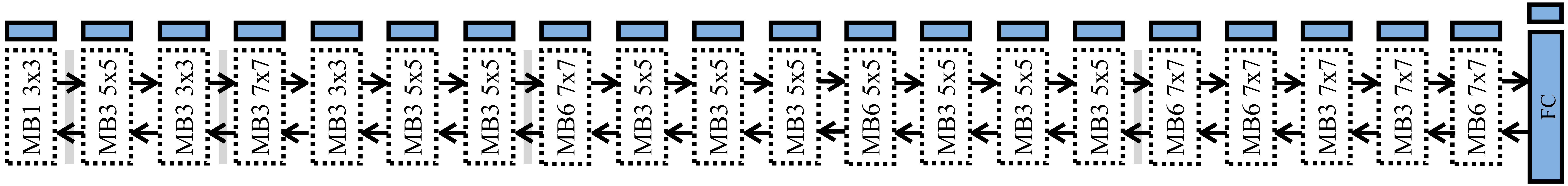
Updating only the last layer is cheap

- No need to backpropagate to previous layers
- But the accuracy is low and not ideal.



1. Sparse Layer/Tensor Update

Bias-only + last layer update



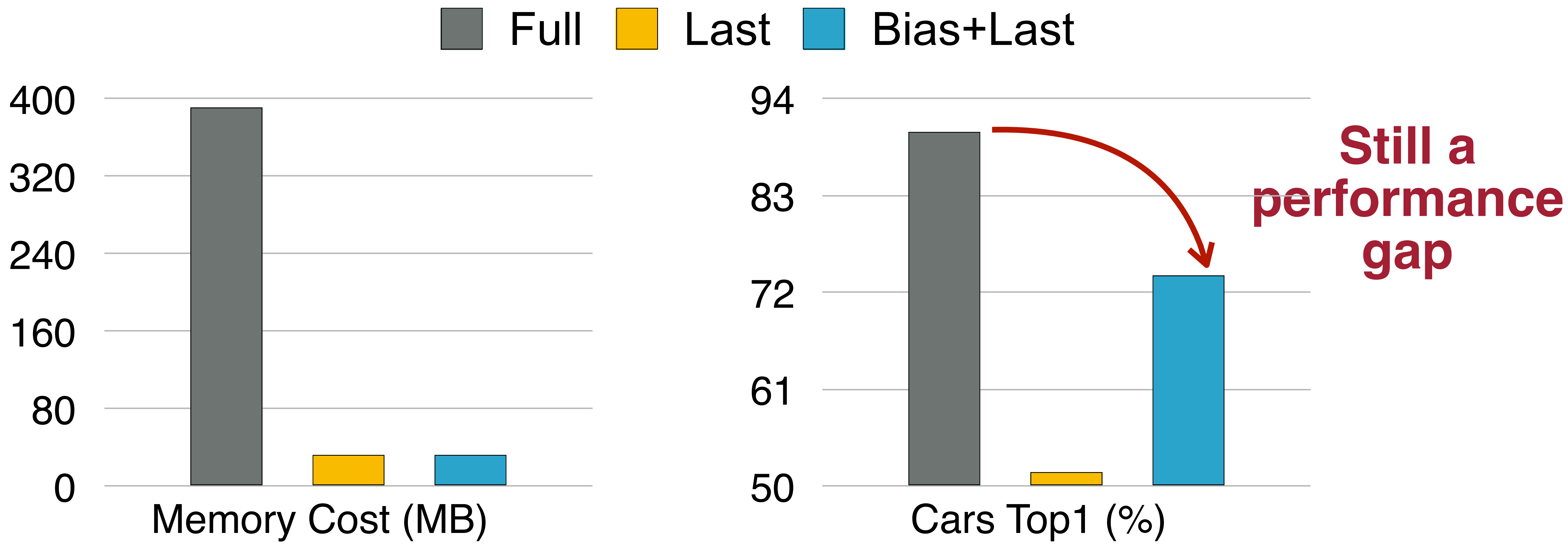
Model: ProxylessNAS-Mobile

Updating the only the bias part

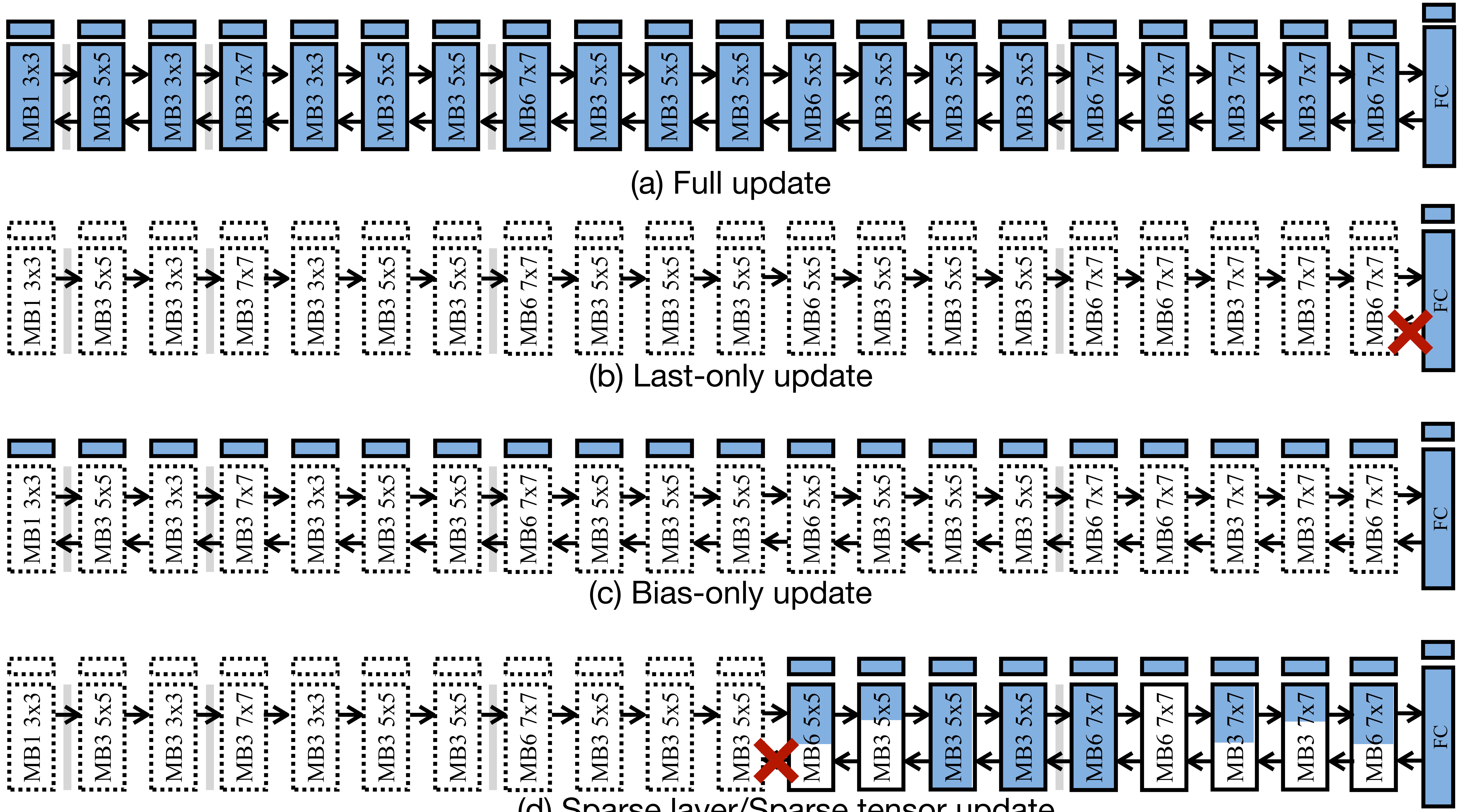
- No need to store the activations
- Back propagating to the first layer.

$$d\mathbf{W} = f(\mathbf{X}, d\mathbf{Y})$$

$$d\mathbf{b} = f(d\mathbf{Y})$$



Update Paradigms Comparison



(a) Full update

(b) Last-only update

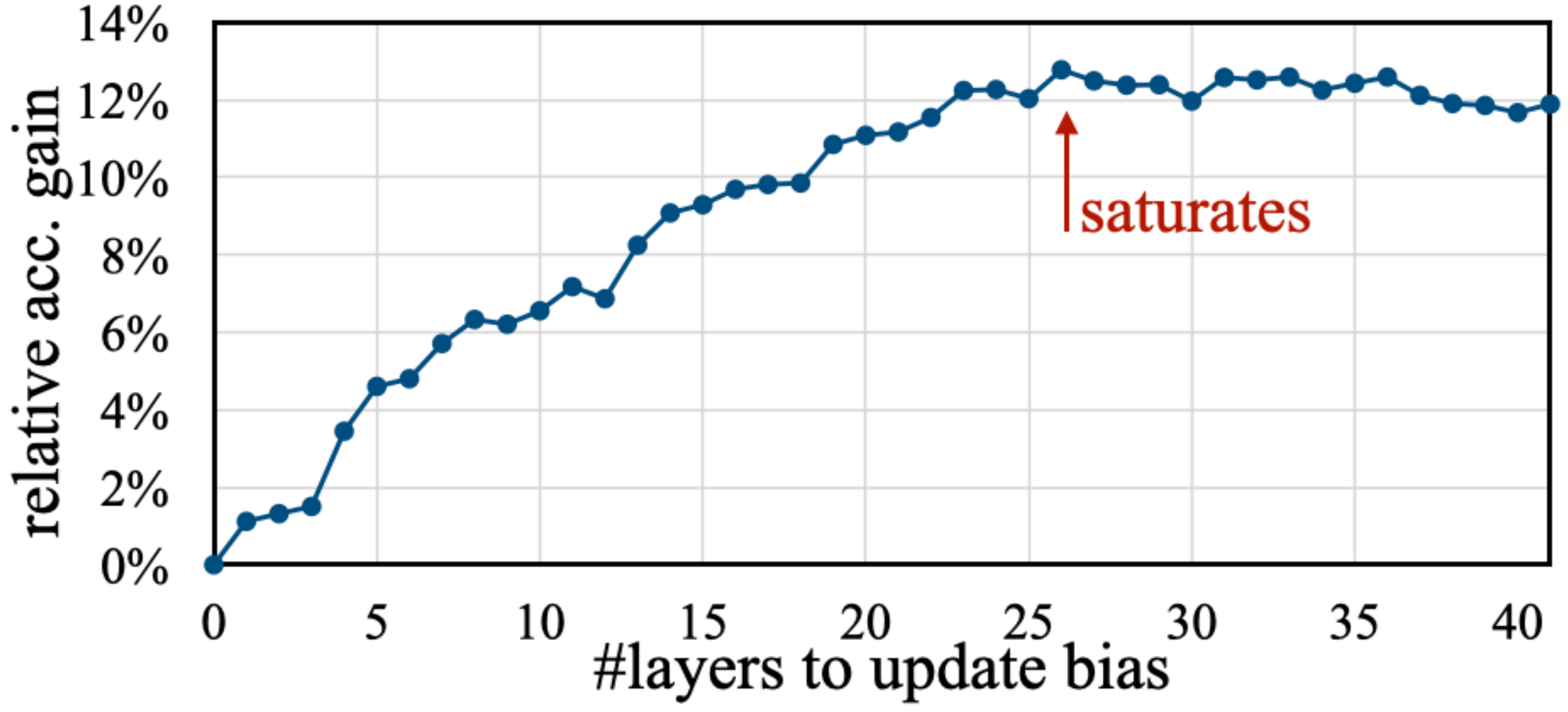
(c) Bias-only update

(d) Sparse layer/Sparse tensor update

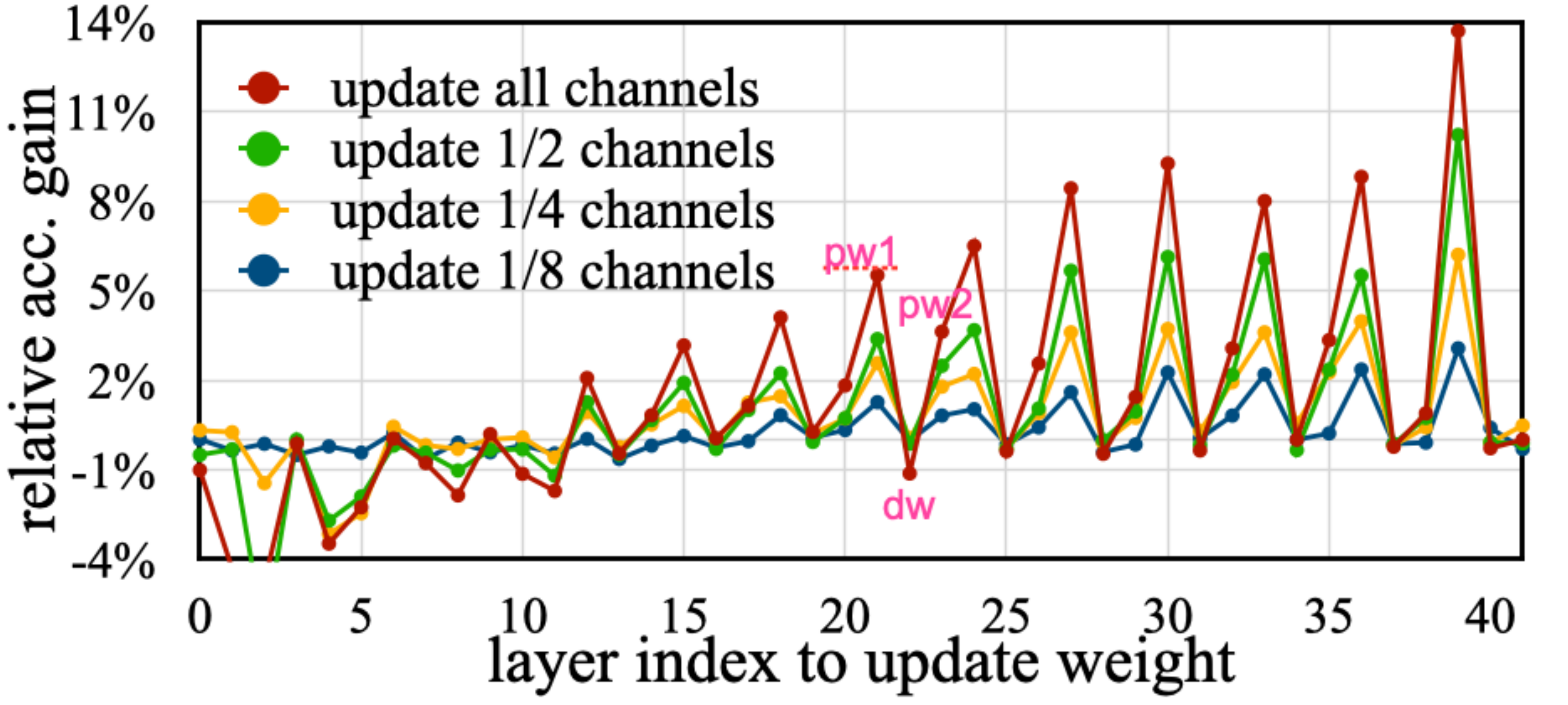
Acknowledgment to NSF Grant OAC-2117997

Find Layers to Update by Contribution Analysis

Finding layers to update with by optimization



(a) Investigate the contribution of last *k* biases $\Delta \mathrm{acc}_{\mathbf{b}[:k]}$



(b) Investigate the contribution of a certain weight $\Delta \mathrm{acc}_{\mathbf{W}_{i,r}}$

For bias update

* Accuracy goes higher as more layers are updated, but plateaus soon.

For weight update

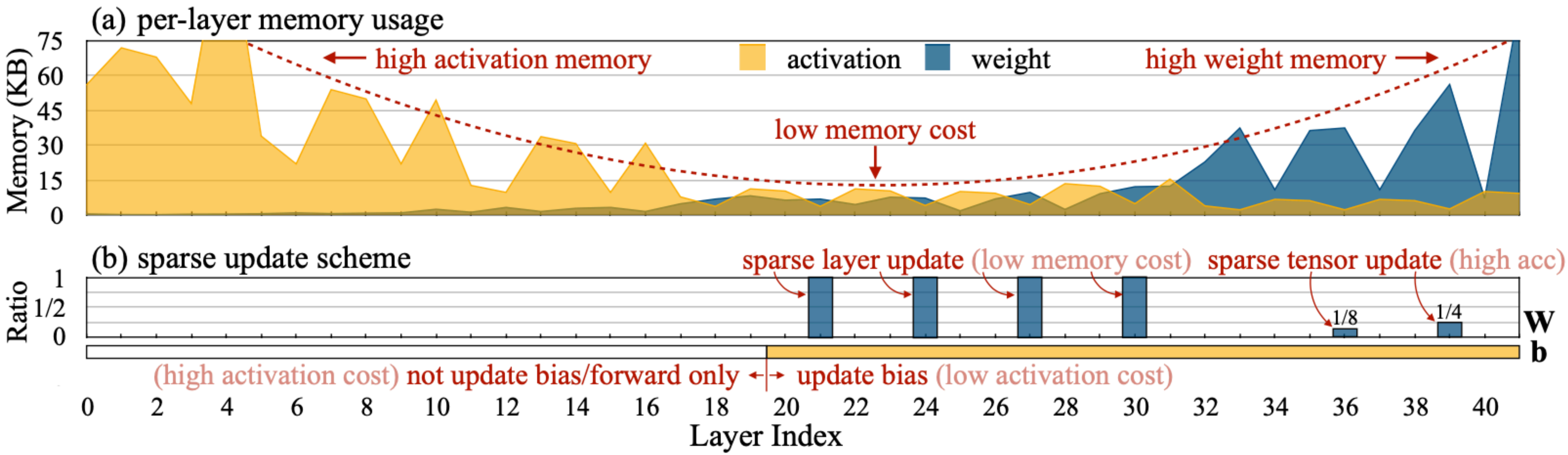
* later layers are more important
* The first point-wise conv contributes more

$$k^*, \mathbf{i}^*, \mathbf{r}^* = \max_{k,\mathbf{i},\mathbf{r}} (\Delta \mathrm{acc}_{\mathbf{b}[:k]} + \sum_{i \in \mathbf{i}, r \in \mathbf{r}} \Delta \mathrm{acc}_{\mathbf{W}_{i,r}}) \quad \text{s.t. Memory}(k, \mathbf{i}, \mathbf{r}) \leq \text{constraint},$$

Acknowledgment to NSF Grant OAC-2117997

Find Layers to Update by Contribution Analysis

Case study: MobileNetV2 update scheme



1. Sparse Layer/Tensor Update

Full update is too expensive

(a) full update

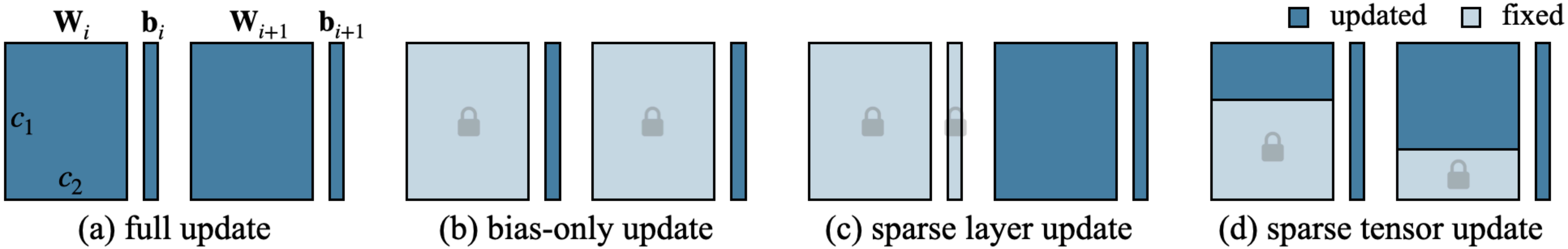
Updating the whole model is **too expensive**:

- Need to save all intermediate activation (quite large)
- Need to store the updated weights in SRAM (Flash is read-only)

 Acknowledgment to NSF Grant OAC-2117997

1. Sparse Layer/Tensor Update

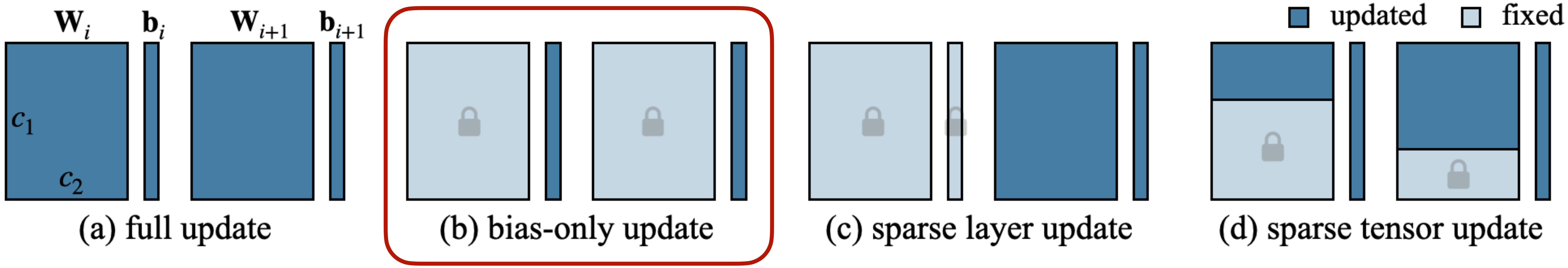
More efficient variants



Acknowledgment to NSF Grant OAC-2117997

1. Sparse Layer/Tensor Update

More efficient variants



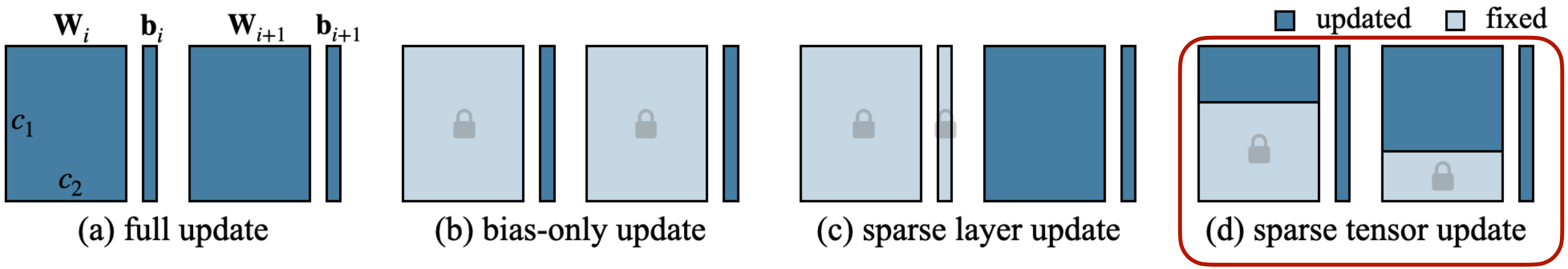
No need to save intermediate activation:

$d\mathbf{W} = f(\mathbf{X}, d\mathbf{Y})$

$d\mathbf{b} = f(d\mathbf{Y})$

1. Sparse Layer/Tensor Update

More efficient variants

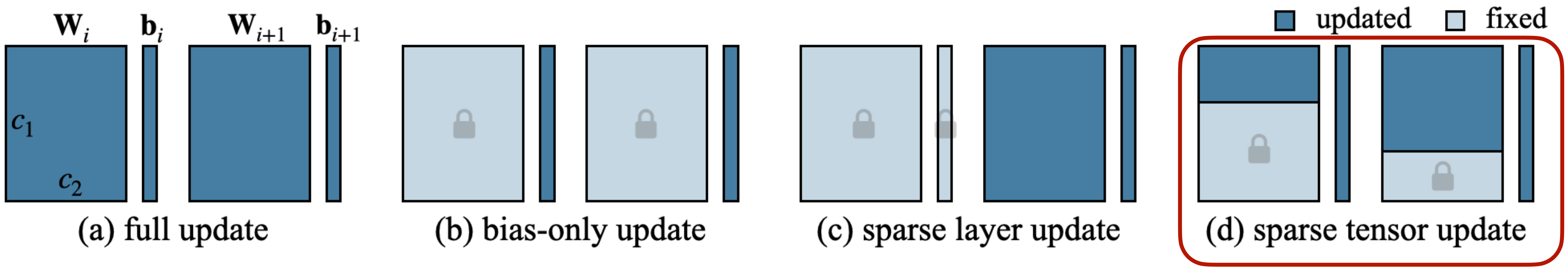


Reduce weight and activation buffer

Acknowledgment to NSF Grant OAC-2117997

1. Sparse Layer/Tensor Update

More efficient variants



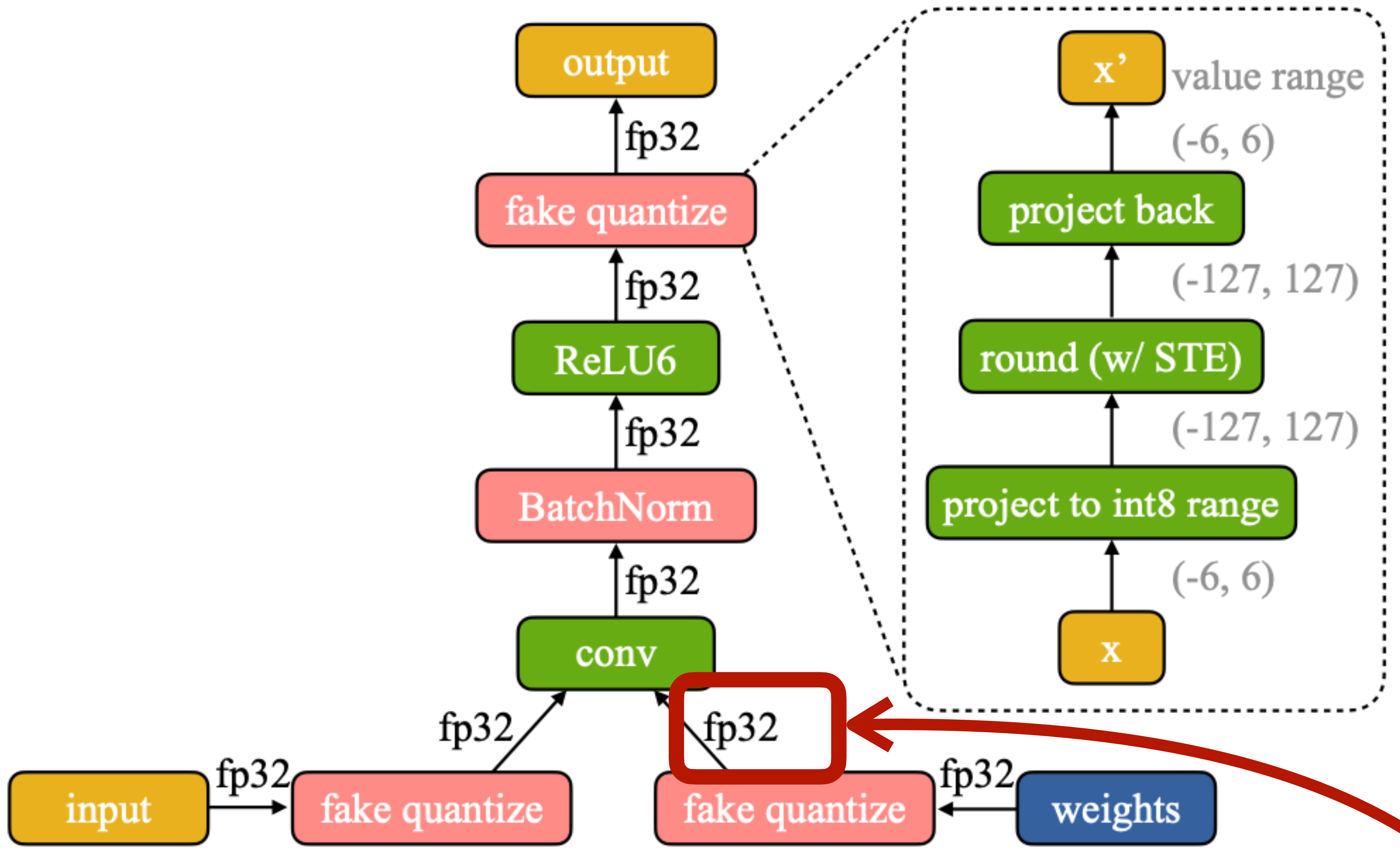
Activation to store: (N, M)
Weight in SRAM: (M, H)

Reduce by 4x

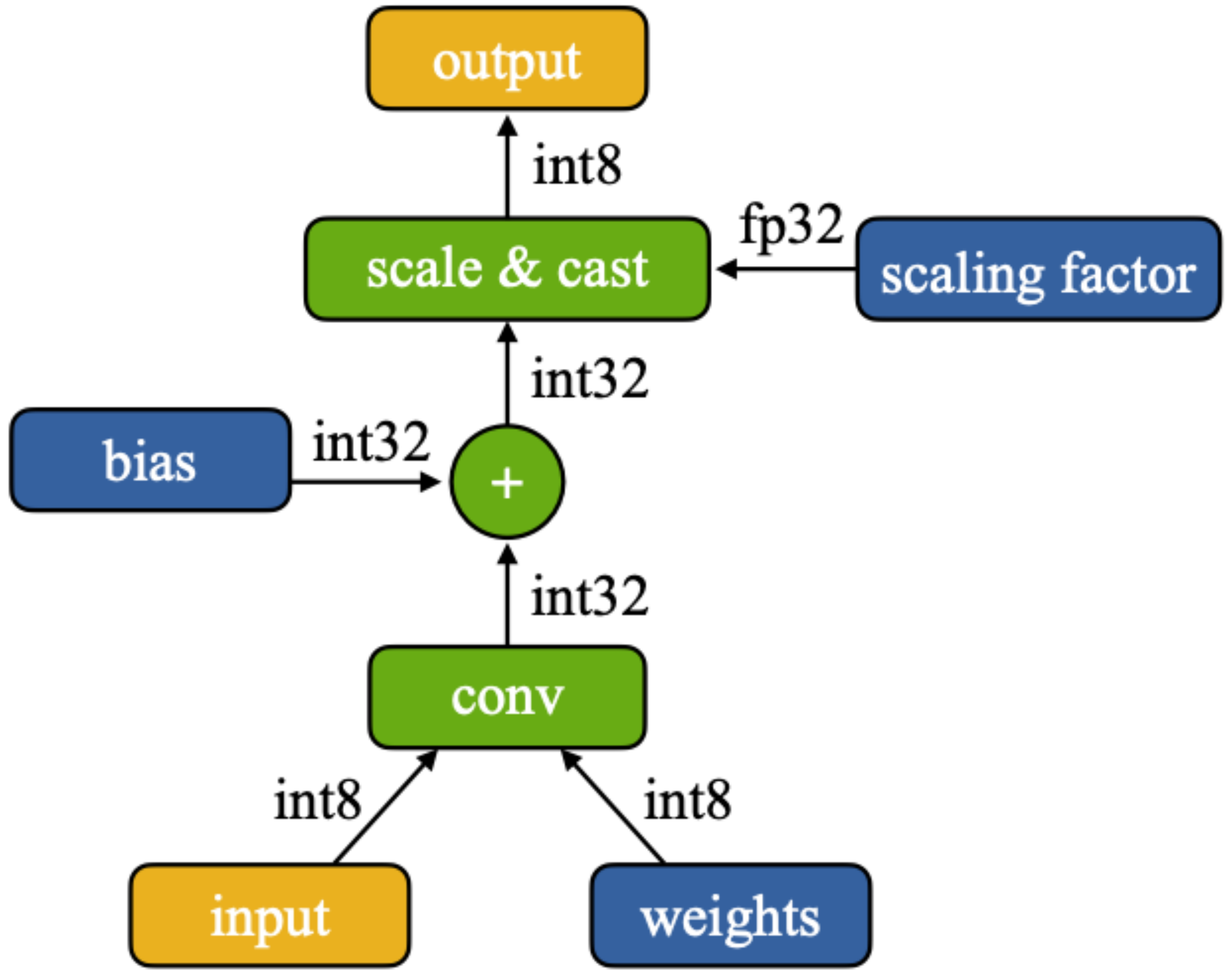
Activation to store: (N, 0.25*M)
Weight in SRAM: (0.25*M, H)

2. Address Optimization Difficulty of Quantized Graphs

Real quantized graphs save memory…



(a) Fake Quantization
(quantization aware training)



(b) Real Quantization
(inference/on-device training)

Intermediate tensors are still in FP32 format in fake quantization,
thus cannot save memory footprint

2. QAS: Quantization-Aware Scaling

QAS addresses the optimization difficulty of quantized graphs

Quantization overview

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

Scaling

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

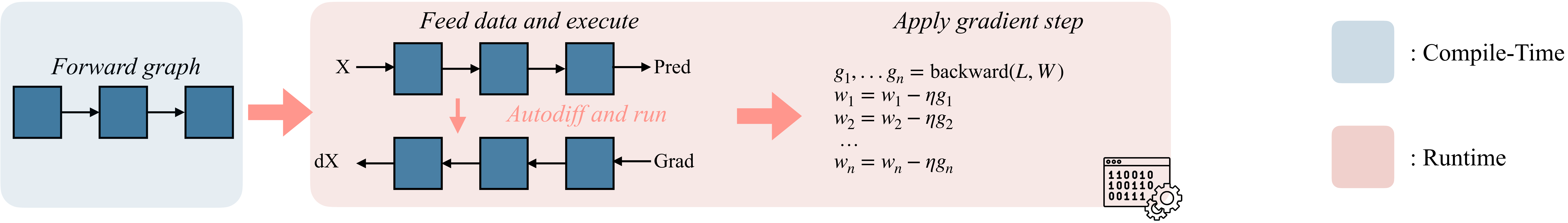
Weight and gradient ratios are off by Sw

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

Thus, re-scale the gradients

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

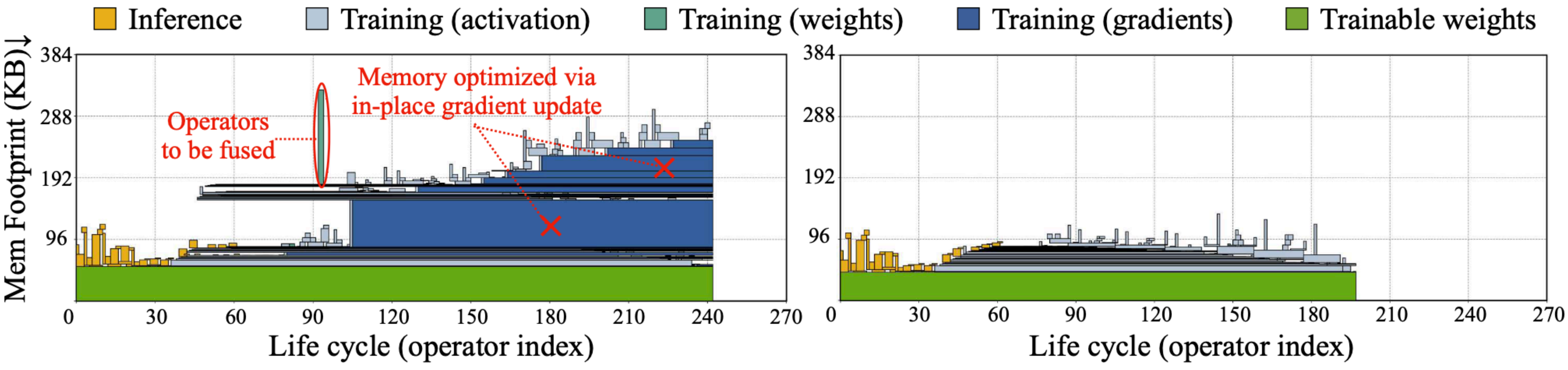
3. Tiny Training Engine (TTE)



Conventional training framework performs most tasks at runtime.

3. Tiny Training Engine (TTE)

Re-ordering reduces memory footprint



(a) Vanilla backward graph

(b) Optimized backward graph

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