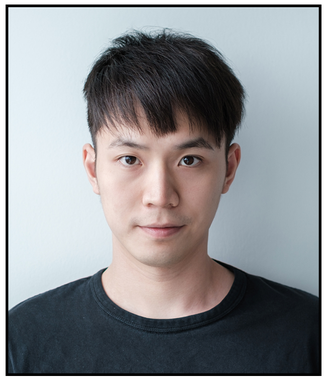
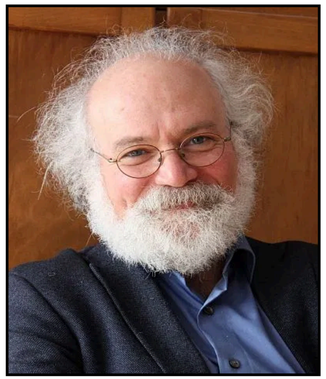
MCUNet: Tiny Deep Learning on IoT Devices



Ji Lin[1] Wei-Ming Chen[1,2] Yujun Lin[1] John Cohn[3] Chuang Gan[3] Song Han[1]

[1]MIT [2]National Taiwan University [3]MIT-IBM Watson AI Lab

NeurIPS 2020 (spotlight)

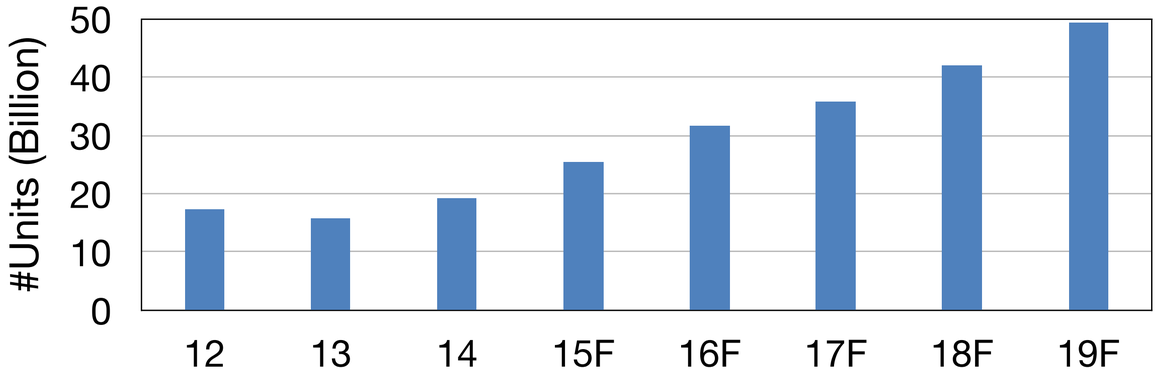
Background: The Era of AIoT on Microcontrollers (MCUs)

- **Low-cost, low-power**

Background: The Era of AIoT on Microcontrollers (MCUs)

- **Low-cost, low-power**

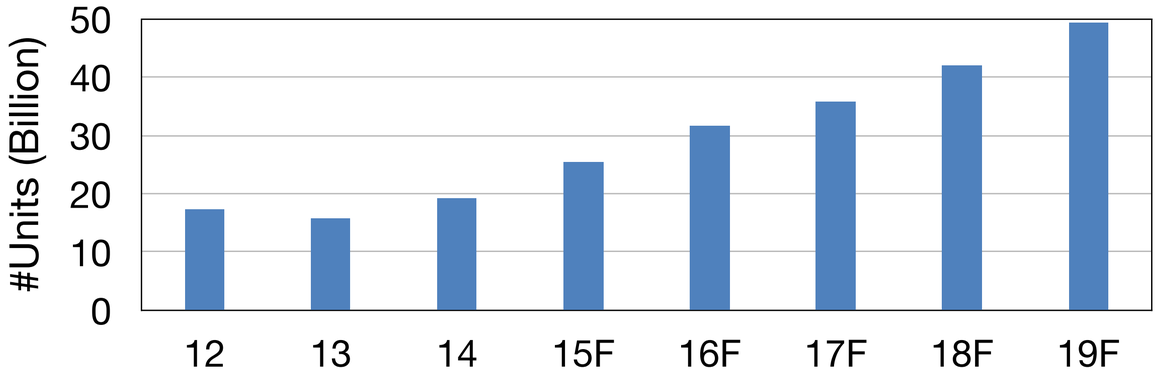
- **Rapid growth**



Background: The Era of AIoT on Microcontrollers (MCUs)

- **Low-cost, low-power**

- **Rapid growth**

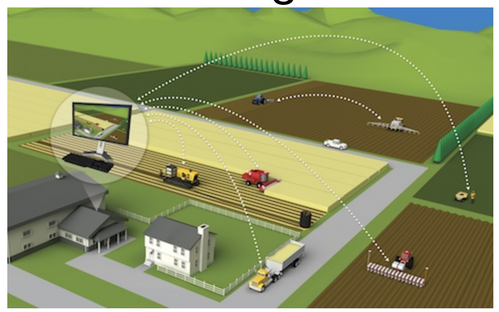


- **Wide applications**

Smart Retail

Personalized Healthcare

Precision Agriculture



Smart Home

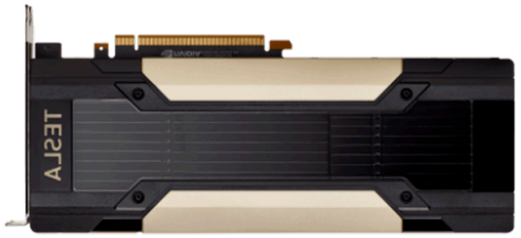
...

Challenge: Memory Too Small to Hold DNN

Memory (Activation)

Storage (Weights)

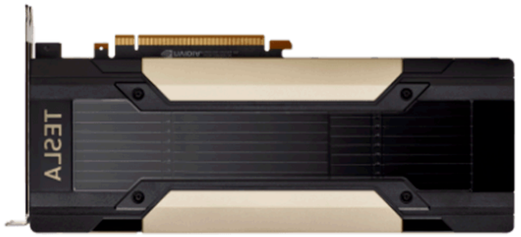
Challenge: Memory Too Small to Hold DNN



Cloud AI

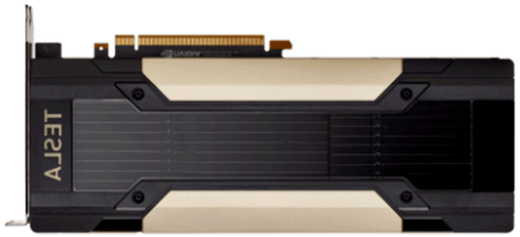
Memory (Activation)	16GB
Storage (Weights)	~TB/PB

Challenge: Memory Too Small to Hold DNN



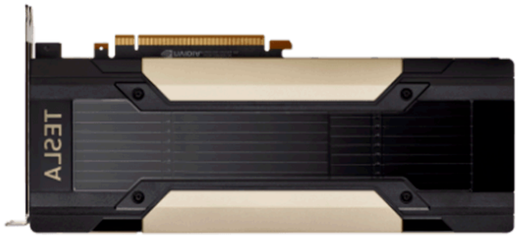
	Cloud AI	Mobile AI
Memory (Activation)	16GB	4GB
Storage (Weights)	~TB/PB	256GB

Challenge: Memory Too Small to Hold DNN



	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	16GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

Challenge: Memory Too Small to Hold DNN

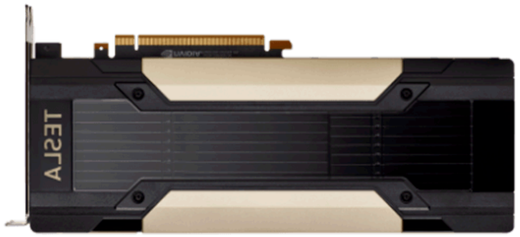


	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	16GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

13,000x smaller

50,000x smaller

Challenge: Memory Too Small to Hold DNN



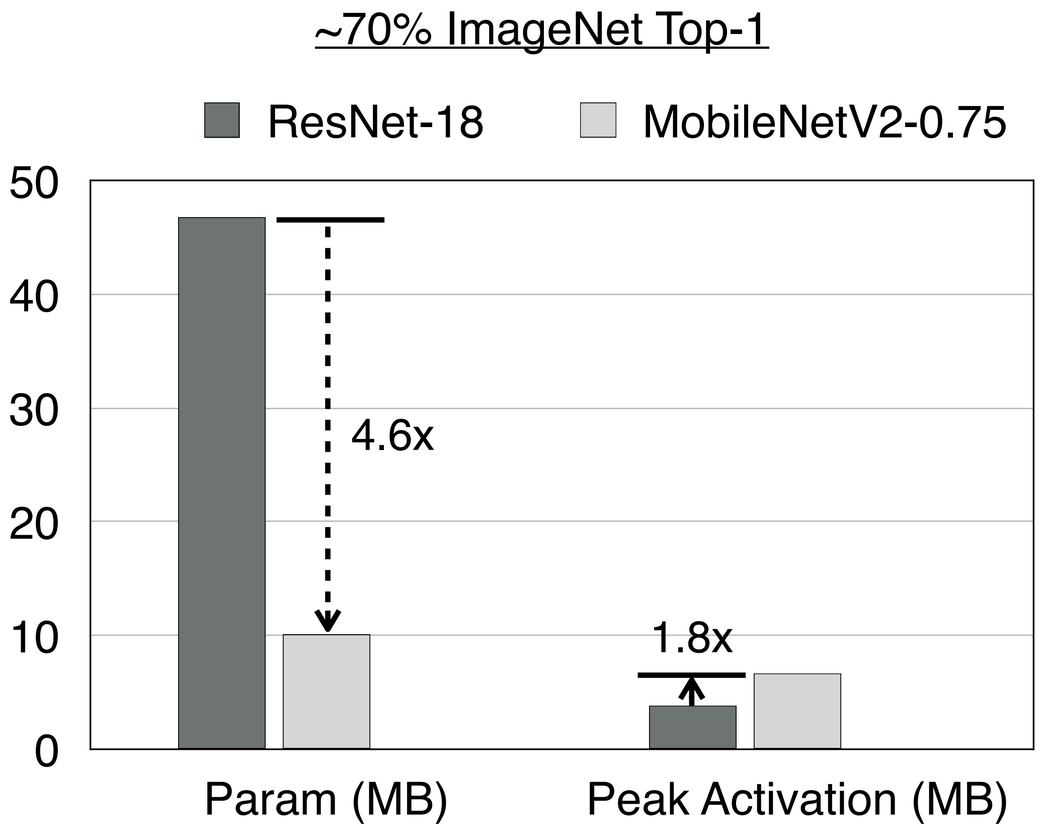
	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	16GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

13,000x smaller

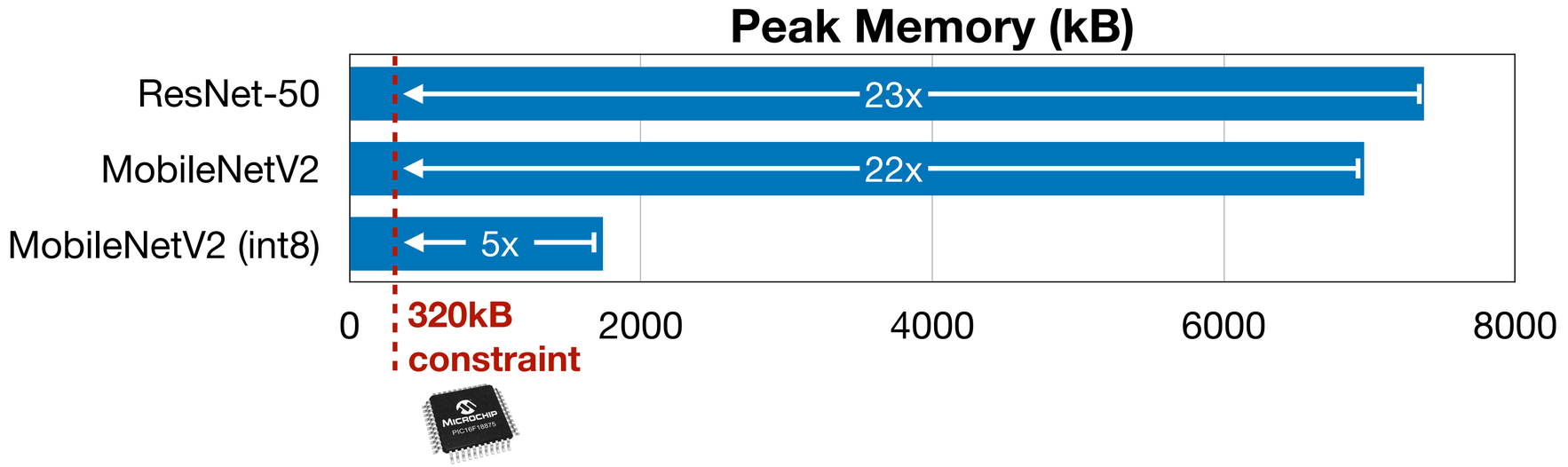
50,000x smaller

We need to reduce the peak activation size **AND** the model size to fit a DNN into MCUs.

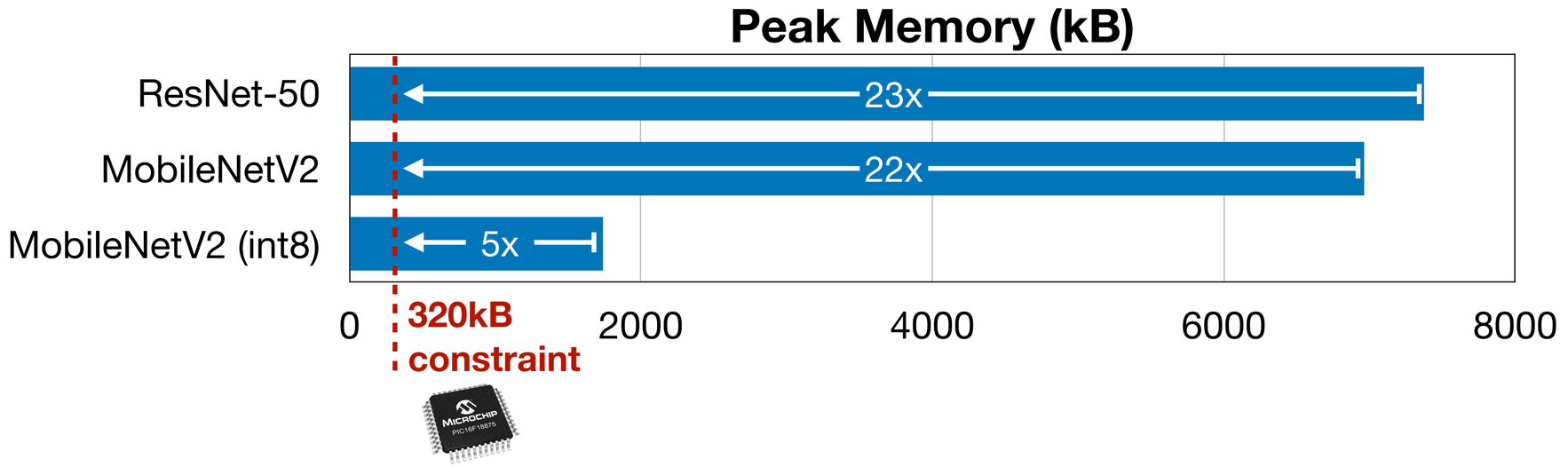
Existing efficient network only reduces model size but NOT activation size!

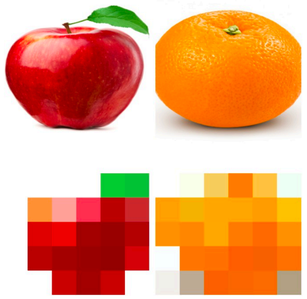


Challenge: Memory Too Small to Hold DNN

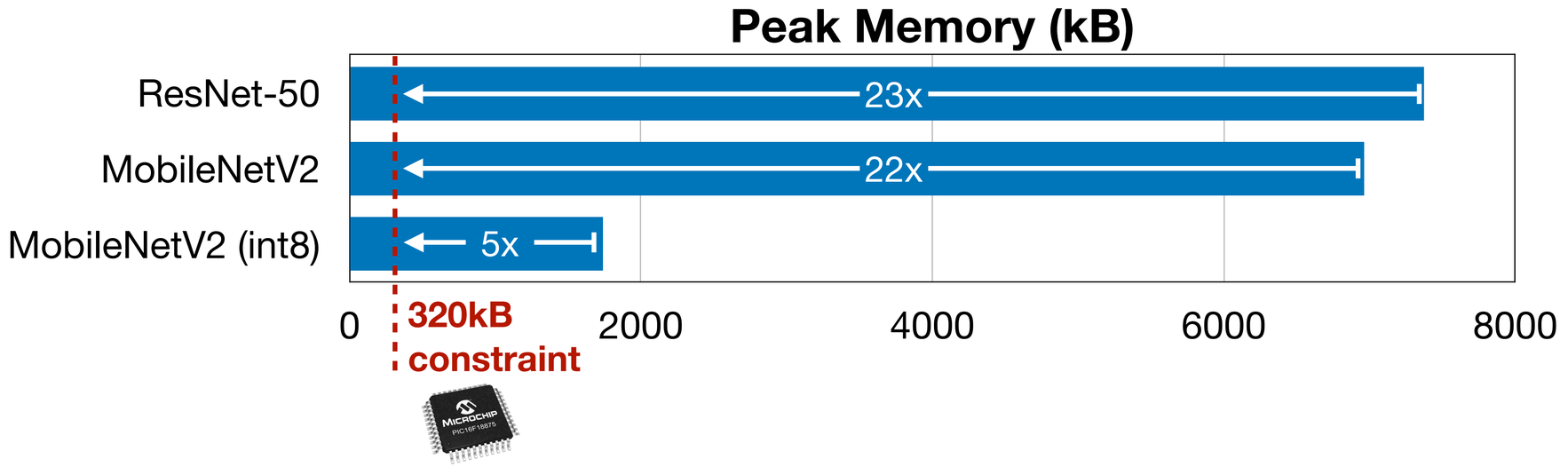


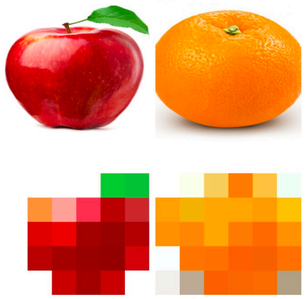
Challenge: Memory Too Small to Hold DNN



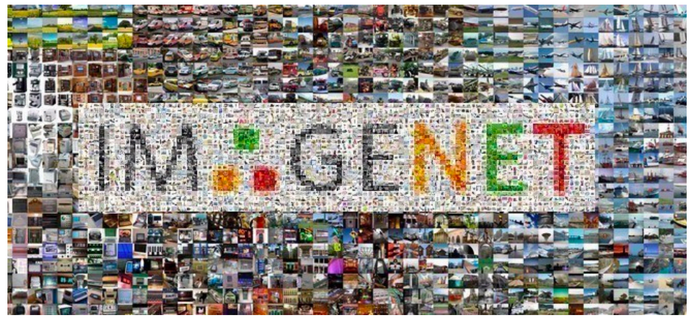


Challenge: Memory Too Small to Hold DNN



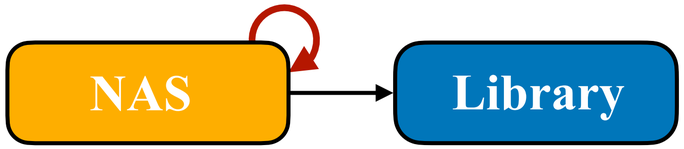


MCUNet



MCUNet: System-Algorithm Co-design

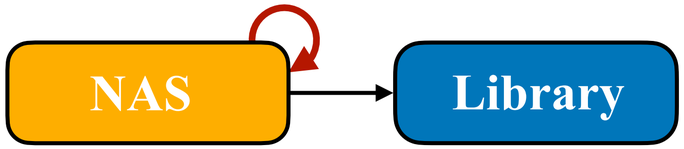
MCUNet: System-Algorithm Co-design



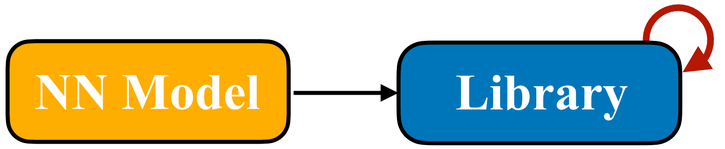
(a) Search NN model on an existing library

e.g., *ProxylessNAS, MnasNet*

MCUNet: System-Algorithm Co-design

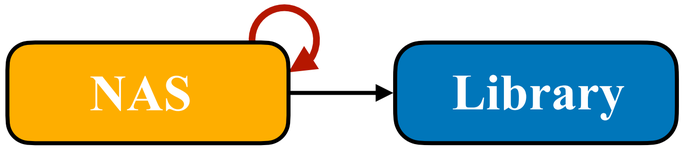


(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*

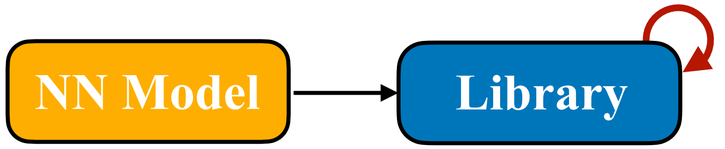


(b) Tune deep learning library given a NN model
e.g., *TVM*

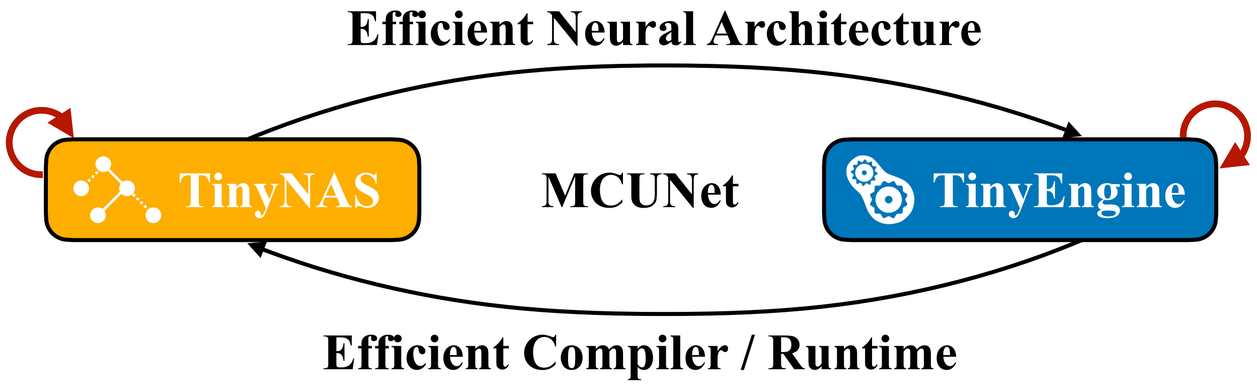
MCUNet: System-Algorithm Co-design



(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*



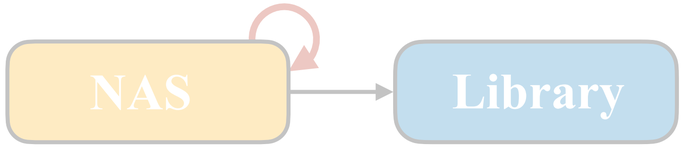
(b) Tune deep learning library given a NN model
e.g., *TVM*



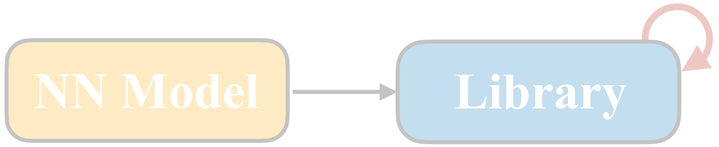
(c) *MCUNet*: system-algorithm co-design

```
# TinyNAS: sample a DNN arch
for arch in arch_space:
  # TinyEngine: find a good schedule
  for schedule in schedule_space:
    # check if satisfy mem. constraints
    if can_fit_memory(arch, schedule):
      # eval acc. and update best arch
      acc = get_valid_acc(arch)
      best_acc = max(best_acc, acc)
      break
```

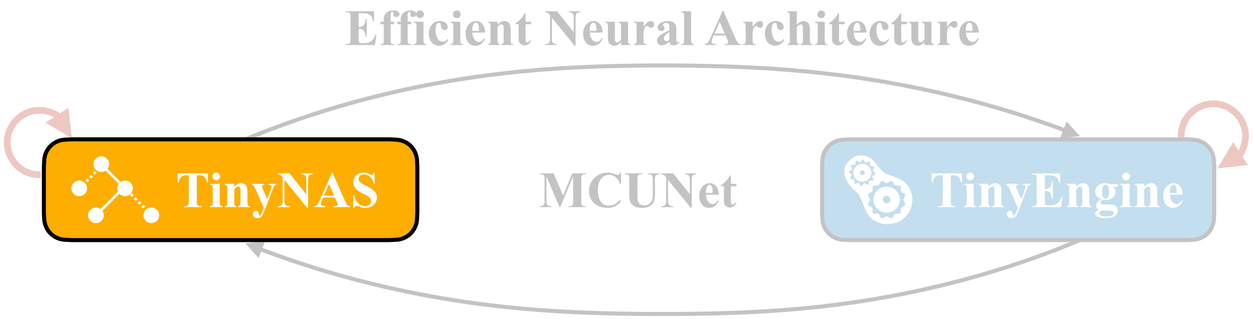
MCUNet: System-Algorithm Co-design



(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*



(b) Tune deep learning library given a NN model
e.g., *TVM*



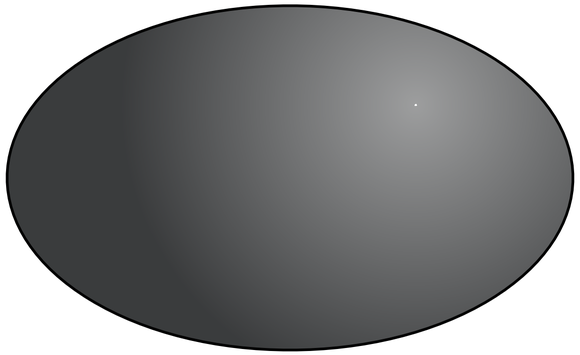
(c) *MCUNet*: system-algorithm co-design

```
# TinyNAS: sample a DNN arch
for arch in arch_space:
  # TinyEngine: find a good schedule
  for schedule in schedule_space:
    # check if satisfy mem. constraints
    if can_fit_memory(arch, schedule):
      # eval acc. and update best arch
      acc = get_valid_acc(arch)
      best_acc = max(best_acc, acc)
      break
```

TinyNAS: Two-Stage NAS for Tiny Memory Constraints

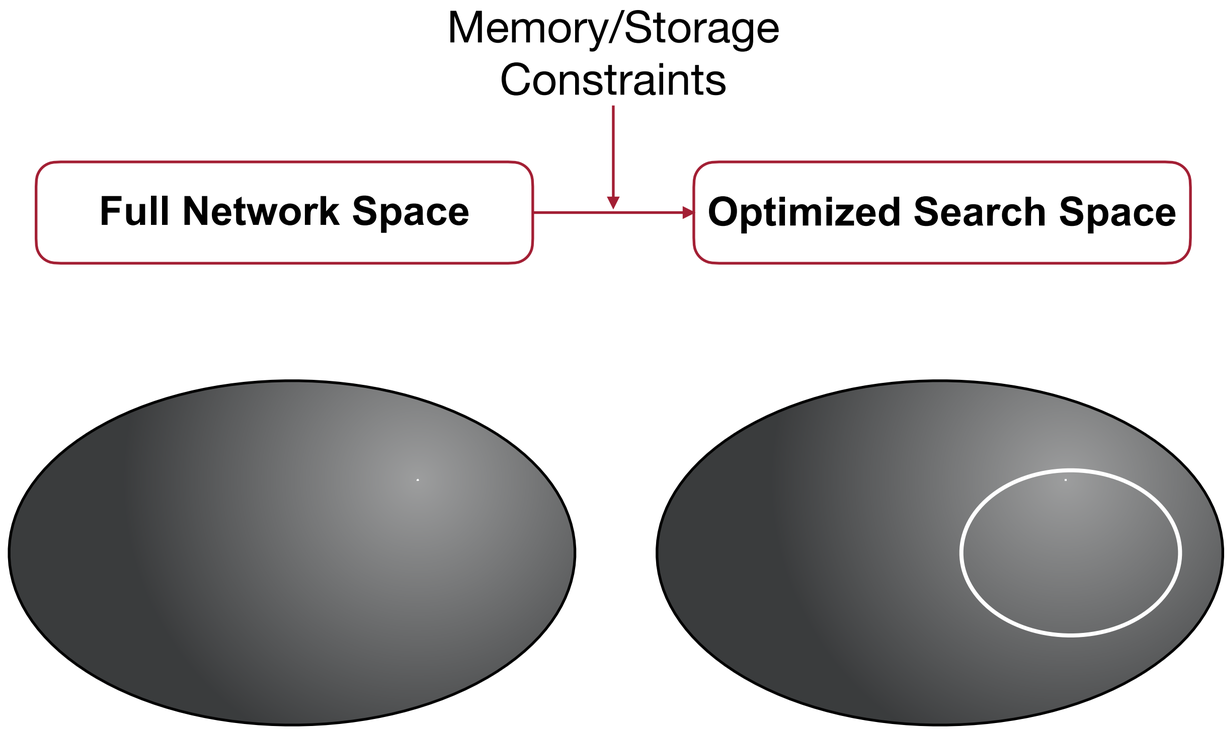
Search space design is crucial for NAS performance
There is no prior expertise on MCU model design

Full Network Space



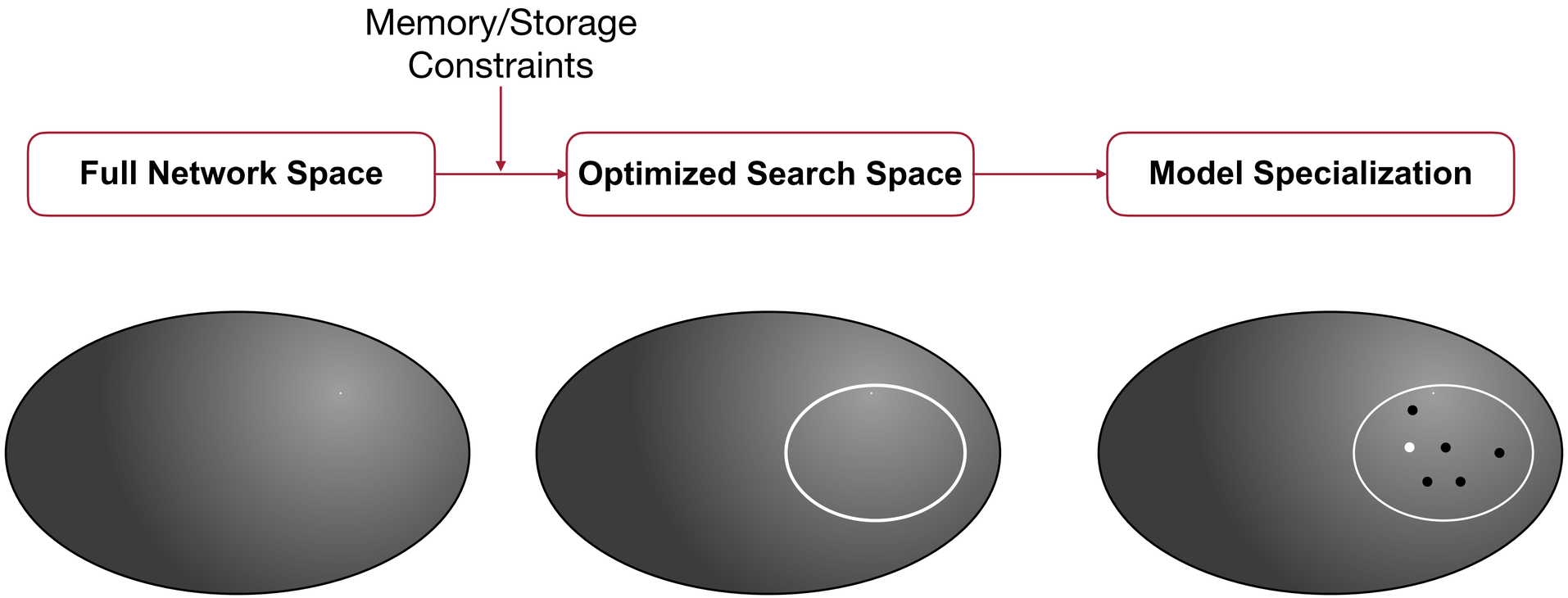
TinyNAS: Two-Stage NAS for Tiny Memory Constraints

Search space design is crucial for NAS performance
There is no prior expertise on MCU model design



TinyNAS: Two-Stage NAS for Tiny Memory Constraints

Search space design is crucial for NAS performance
There is no prior expertise on MCU model design



TinyNAS: (1) Automated search space optimization

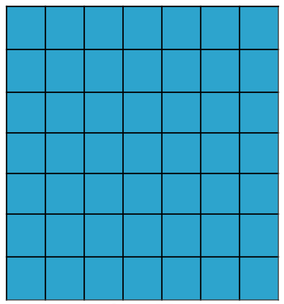
Revisit ProxylessNAS search space:

$$S = kernel\ size \times expansion\ ratio \times depth$$

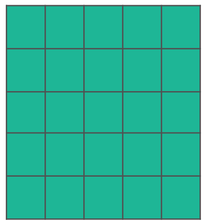
TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

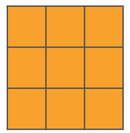
$S = \underline{\textit{kernel size}} \times \textit{expansion ratio} \times \textit{depth}$



k=7



k=5

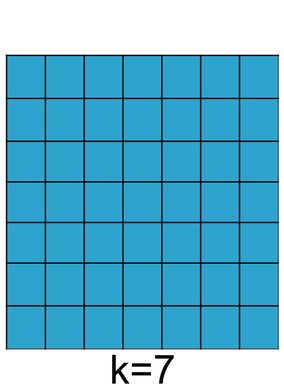


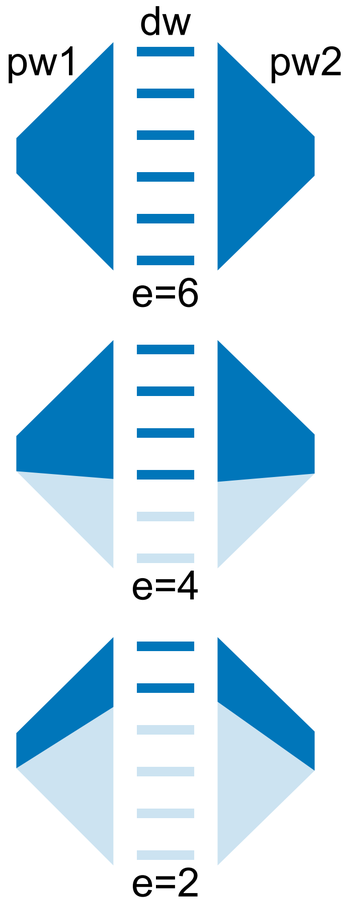
k=3

TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

$S = kernel\ size \times \underline{expansion\ ratio} \times depth$

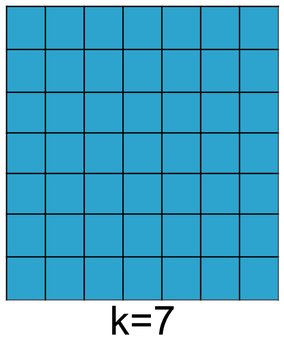


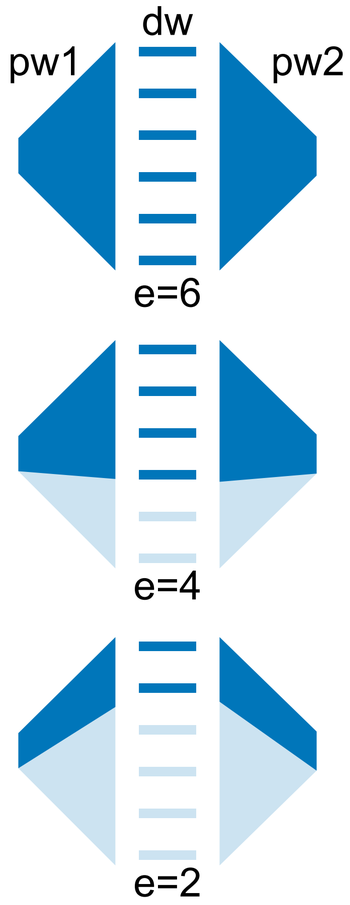


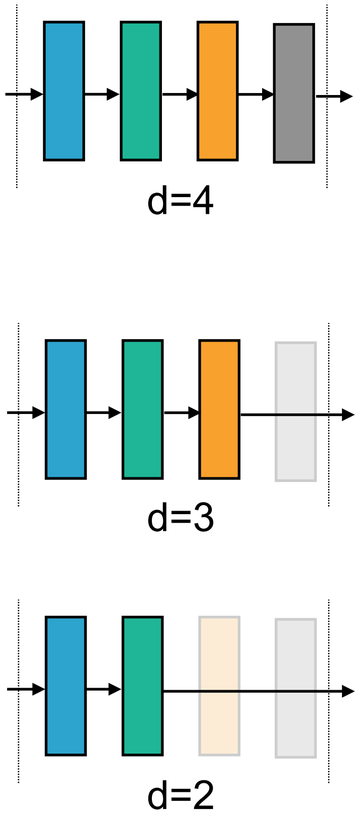
TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

$S = kernel\ size \times expansion\ ratio \times \underline{depth}$



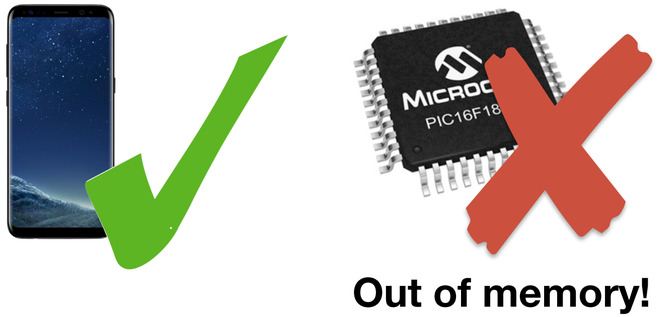




TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

$S = kernel\ size \times expansion\ ratio \times depth$



Out of memory!

TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

$S' = kernel\ size \times expansion\ ratio \times depth \times input\ resolution\ \underline{R} \times width\ multiplier\ \underline{W}$

TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

S' = *kernel size* × *expansion ratio* × *depth* × *input resolution* $\underline{R}$ × *width multiplier* $\underline{W}$

Different R and W for different hardware capacity
(i.e., different optimized sub-space)

R=224, W=1.0

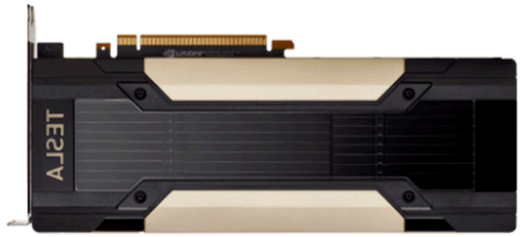
TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

S' = *kernel size* × *expansion ratio* × *depth* × *input resolution* $\underline{R}$ × *width multiplier* $\underline{W}$

Different R and W for different hardware capacity
(i.e., different optimized sub-space)

R=260, W=1.4 *



R=224, W=1.0

* Cai et al., Once-for-All: Train One Network and Specialize it for Efficient Deployment, ICLR'20

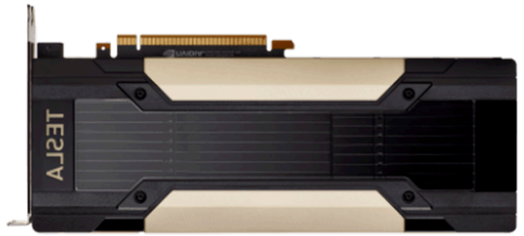
TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

$S' = kernel\ size \times expansion\ ratio \times depth \times input\ resolution\ \underline{R} \times width\ multiplier\ \underline{W}$

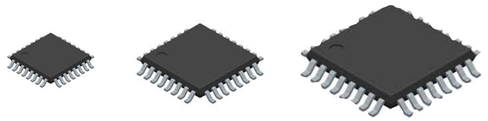
Different R and W for different hardware capacity
(i.e., different optimized sub-space)

R=260, W=1.4



R=224, W=1.0

R=?, W=?



F412/F743/H746/…
256kB/320kB/512kB/…

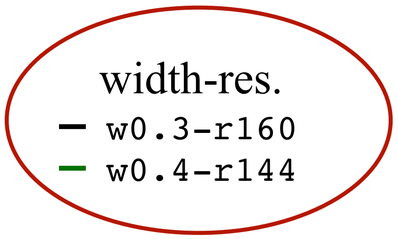
TinyNAS: (1) Automated search space optimization

Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy

TinyNAS: (1) Automated search space optimization

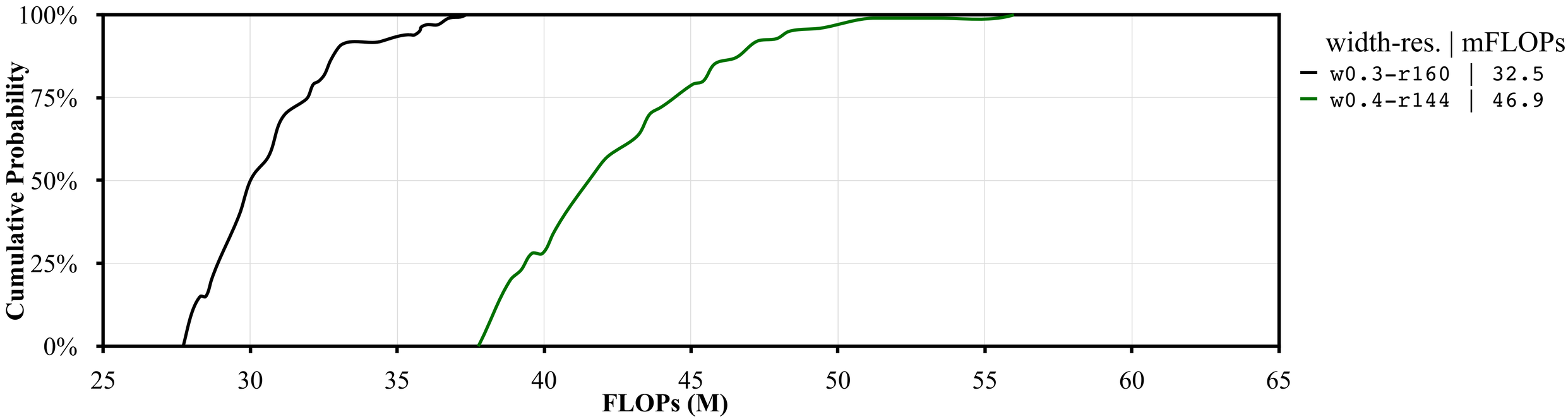
Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy

320kB?



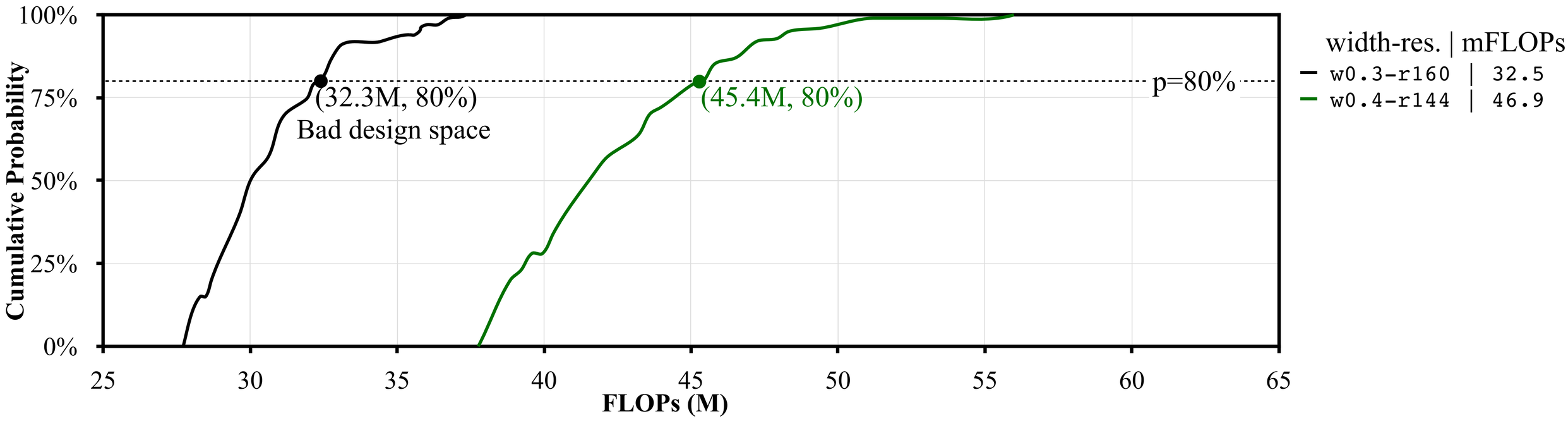
TinyNAS: (1) Automated search space optimization

Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



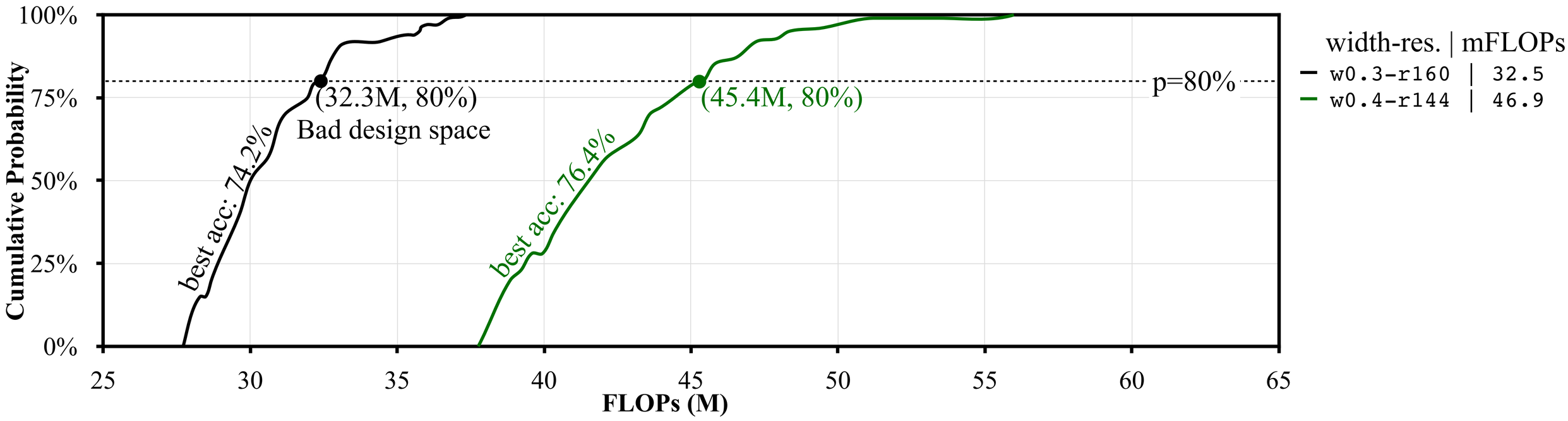
TinyNAS: (1) Automated search space optimization

Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



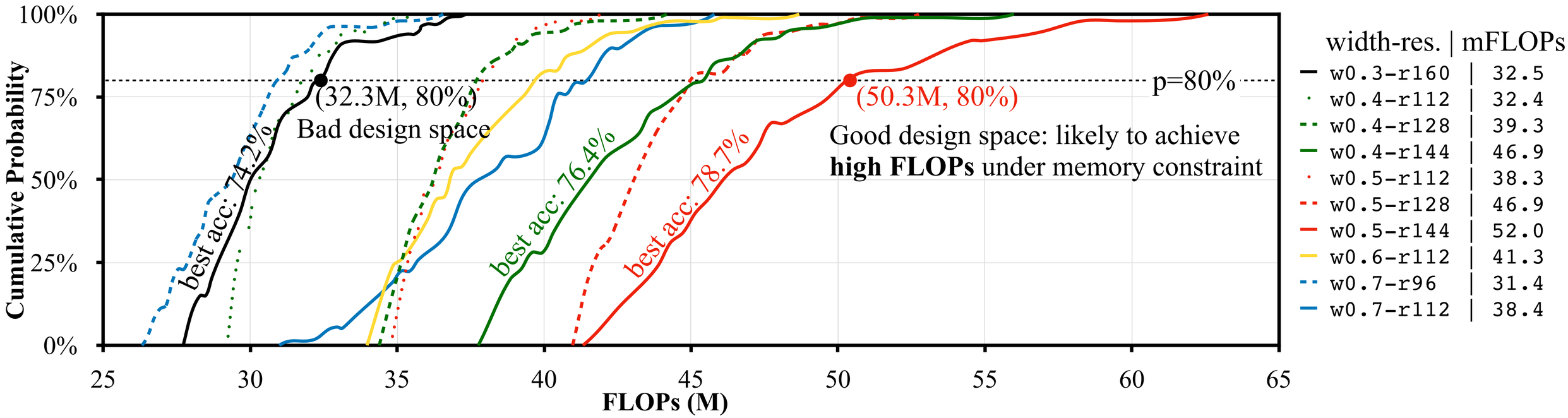
TinyNAS: (1) Automated search space optimization

Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



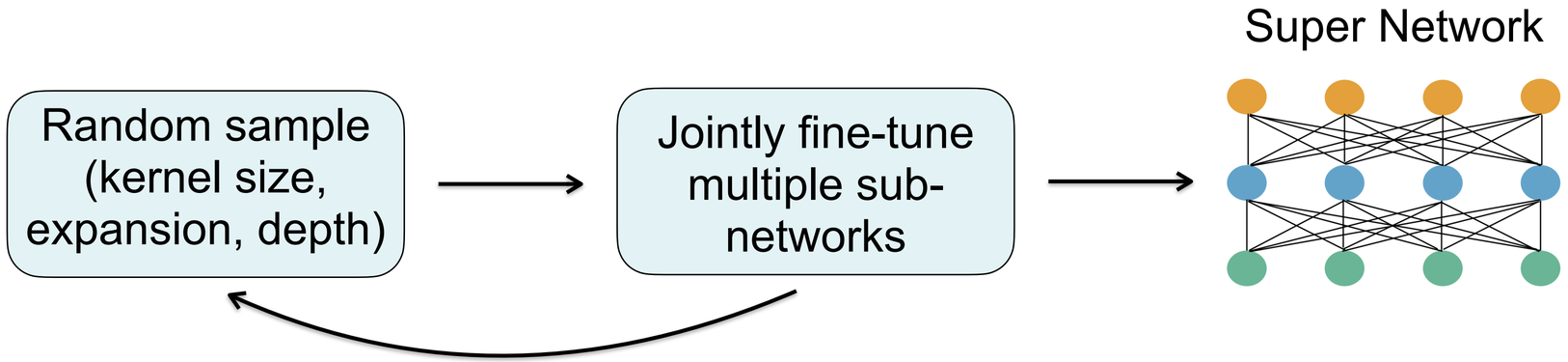
TinyNAS: (1) Automated search space optimization

Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy



TinyNAS: (2) Resource-constrained model specialization

- One-shot NAS through weight sharing

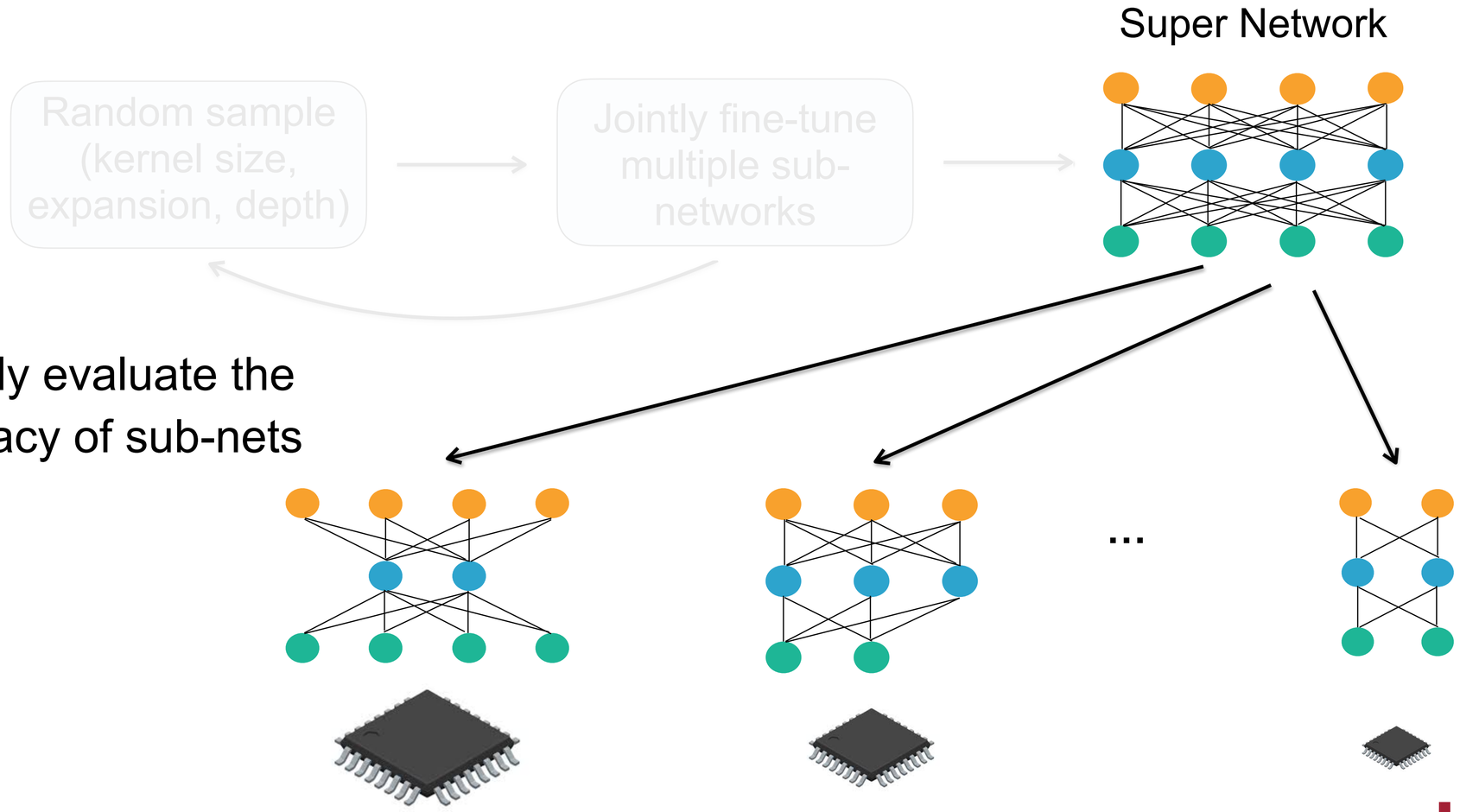


- Small sub-networks are nested in large sub-networks.

* Cai et al., Once-for-All: Train One Network and Specialize it for Efficient Deployment, ICLR'20

TinyNAS: (2) Resource-constrained model specialization

- One-shot NAS through weight sharing



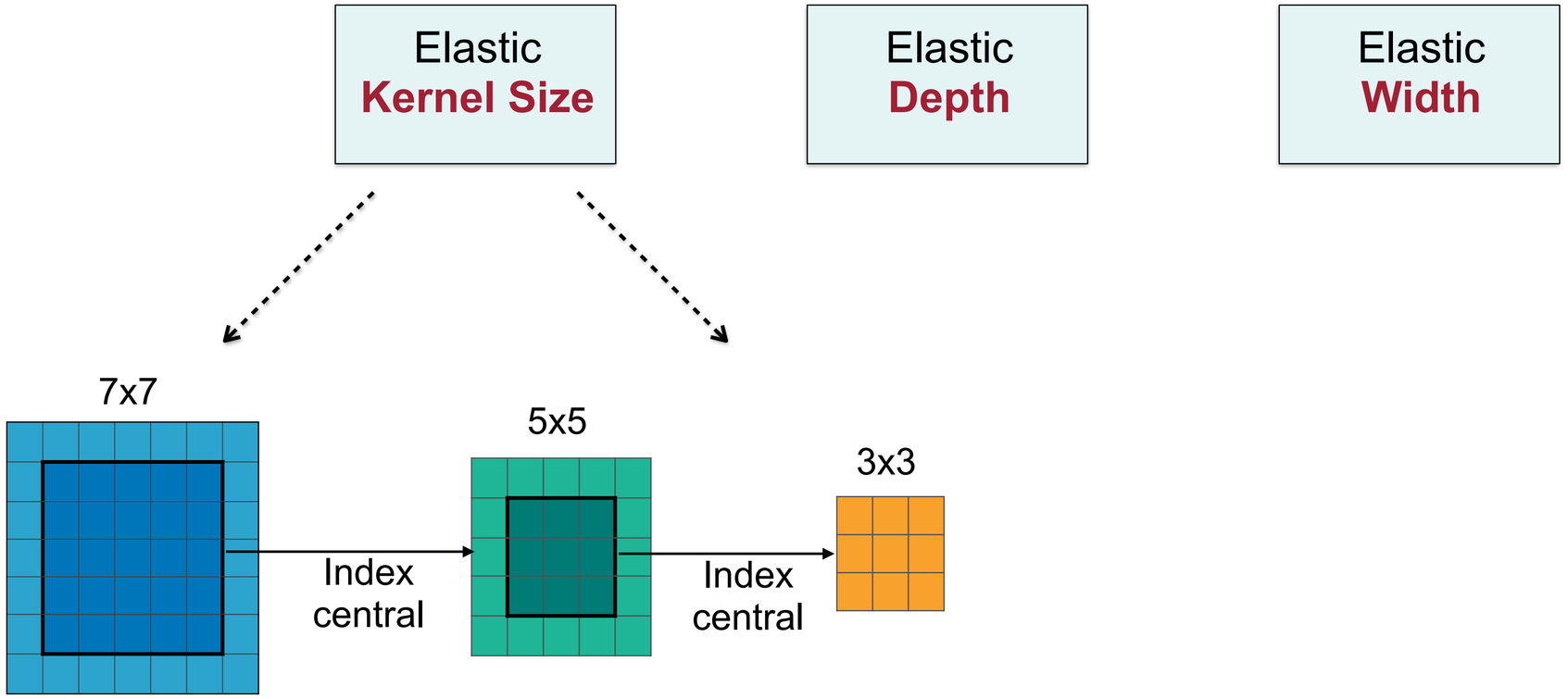
TinyNAS: (2) Resource-constrained model specialization

Elastic
Kernel Size

Elastic
Depth

Elastic
Width

TinyNAS: (2) Resource-constrained model specialization



Start with **full** kernel size

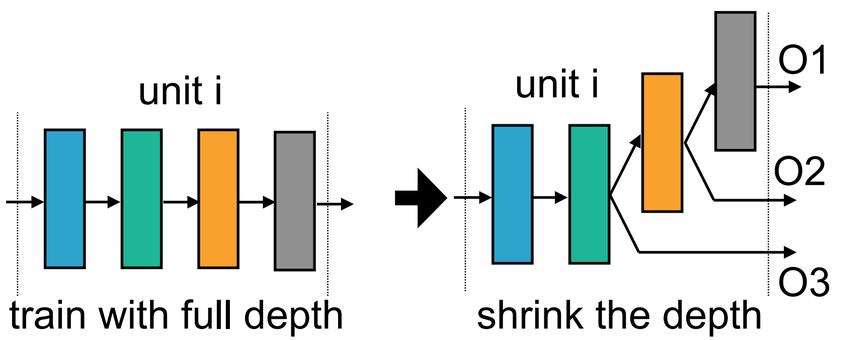
Smaller kernel takes centered weights

TinyNAS: (2) Resource-constrained model specialization

Elastic **Kernel Size**

Elastic **Depth**

Elastic **Width**



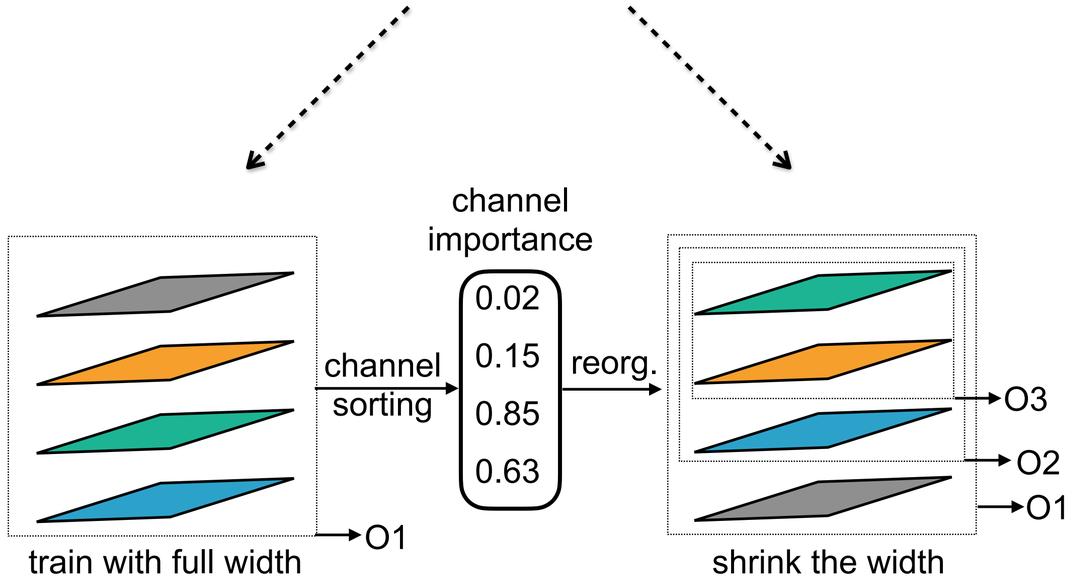
Allow **later** layers in each unit
to be skipped to reduce the depth

TinyNAS: (2) Resource-constrained model specialization

Elastic **Kernel Size**

Elastic **Depth**

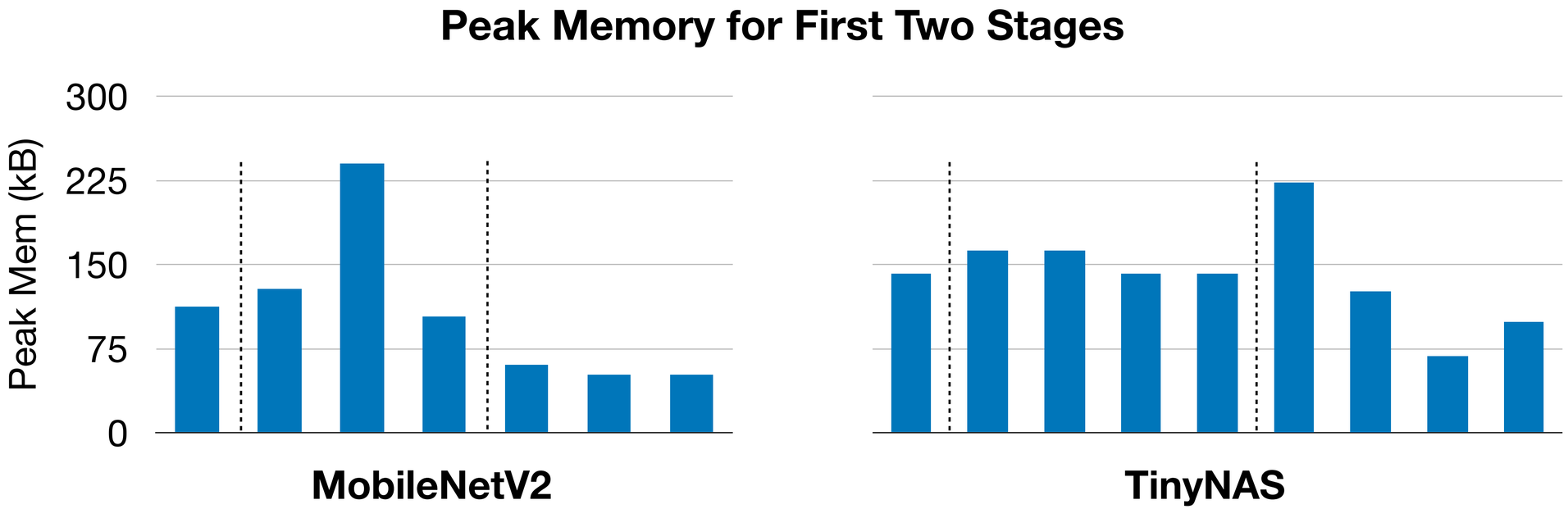
Elastic **Width**



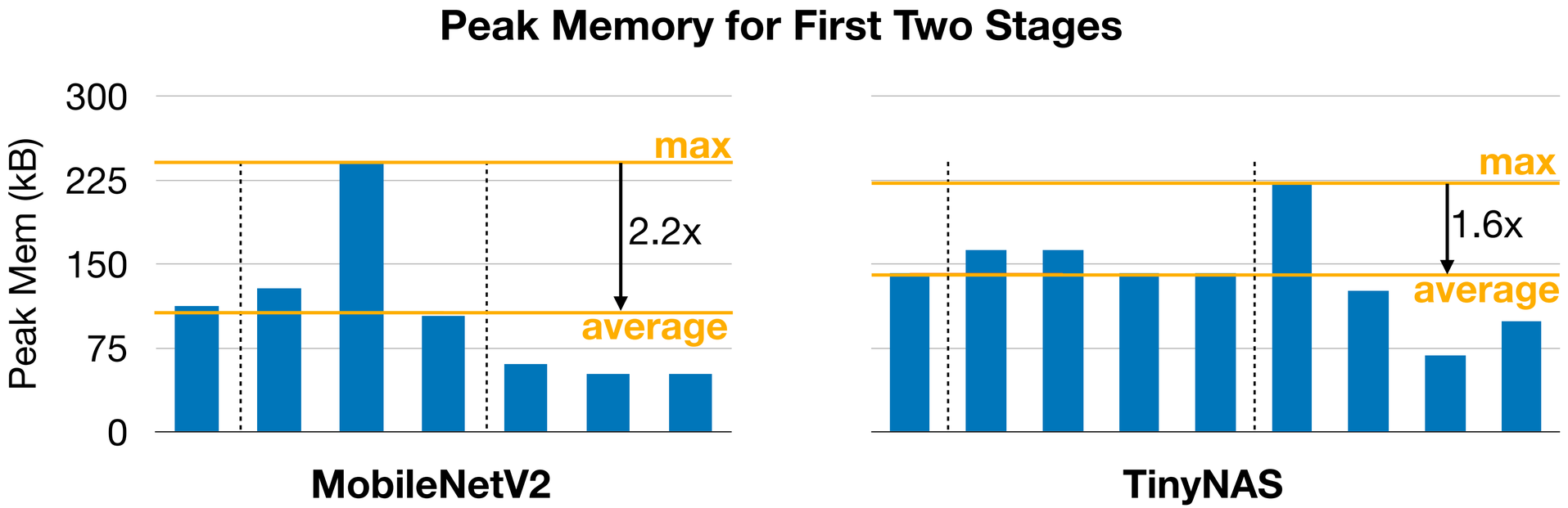
Shrink the width

Keep the most important channels when shrinking via **channel sorting**

TinyNAS Better Utilizes the Memory



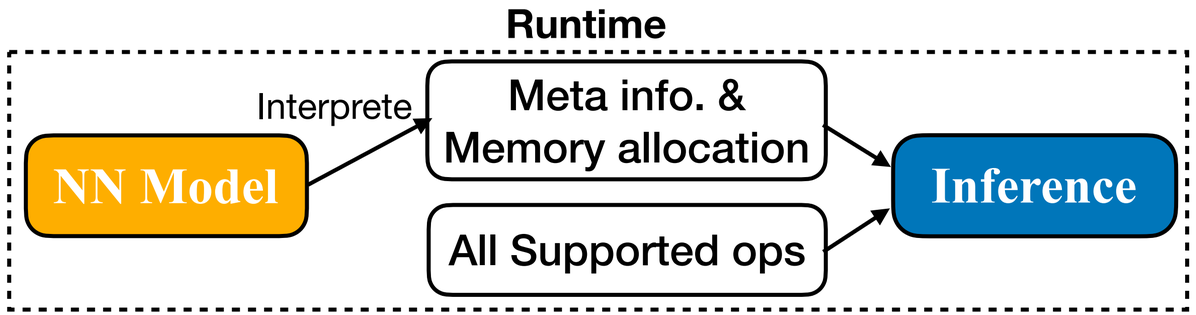
TinyNAS Better Utilizes the Memory



- TinyNAS designs networks with more uniform peak memory for each block, allowing us to fit a larger model at the same amount of memory

TinyEngine: A Memory-Efficient Inference Library

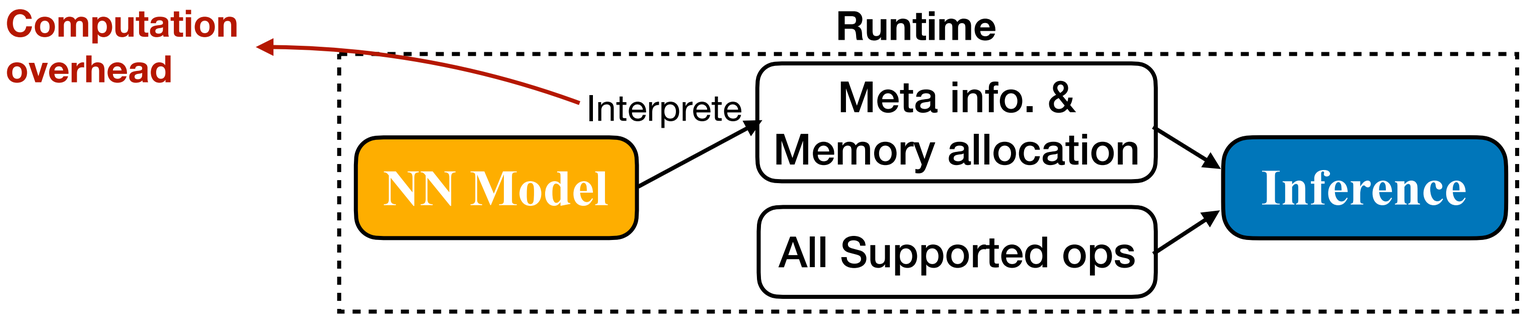
1. Reducing overhead with separated compilation & runtime



(a) Existing libraries based on **runtime interpretation**
e.g., *TF-Lite Micro, CMSIS-NN*

TinyEngine: A Memory-Efficient Inference Library

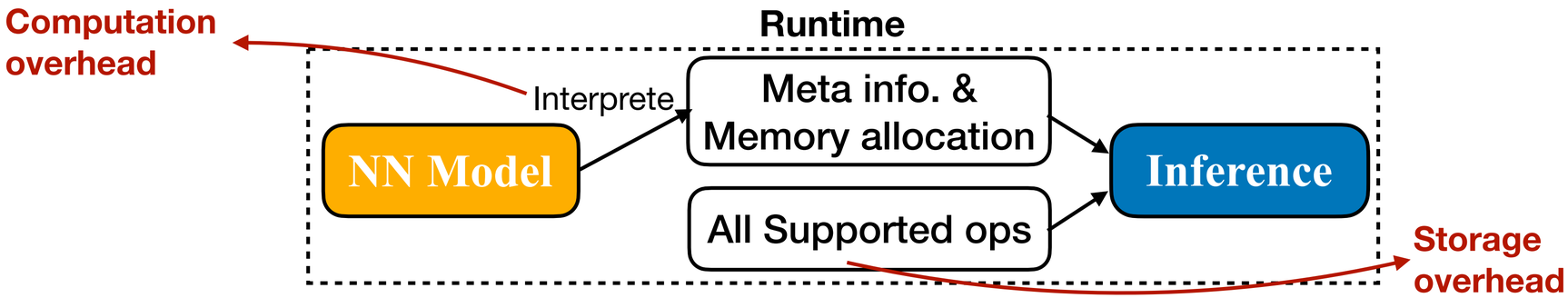
1. Reducing overhead with separated compilation & runtime



(a) Existing libraries based on **runtime interpretation**
e.g., *TF-Lite Micro, CMSIS-NN*

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime

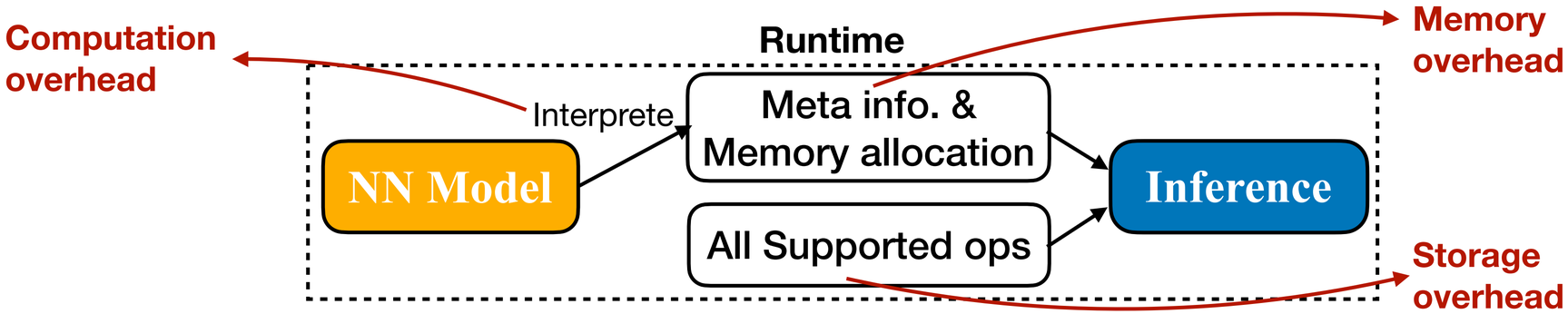


(a) Existing libraries based on **runtime interpretation**

e.g., *TF-Lite Micro, CMSIS-NN*

TinyEngine: A Memory-Efficient Inference Library

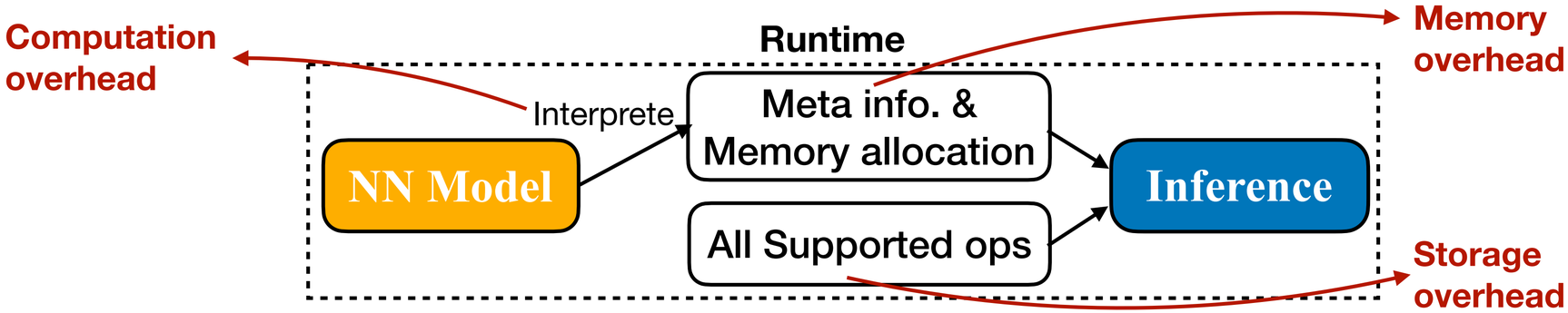
1. Reducing overhead with separated compilation & runtime



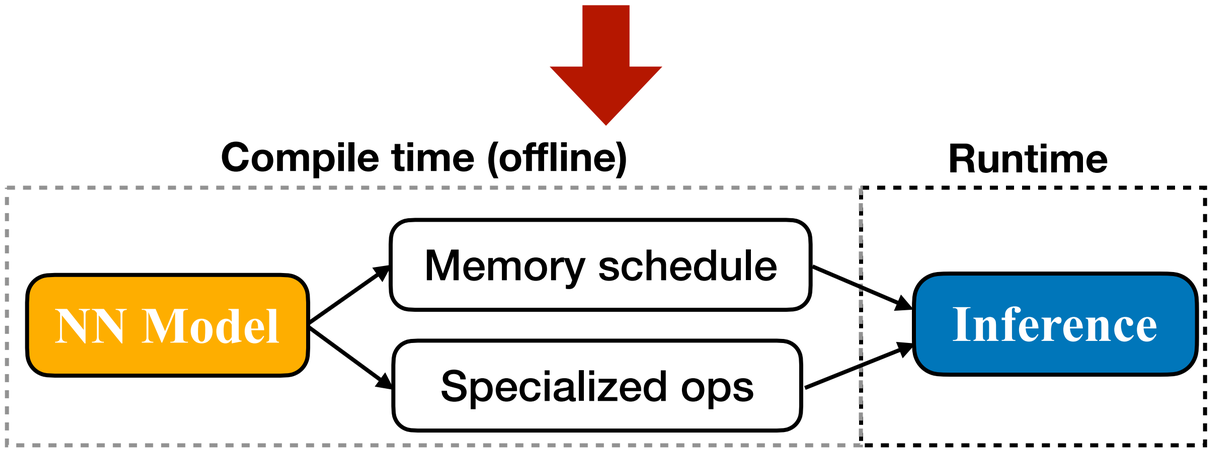
(a) Existing libraries based on **runtime interpretation**
e.g., *TF-Lite Micro, CMSIS-NN*

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime



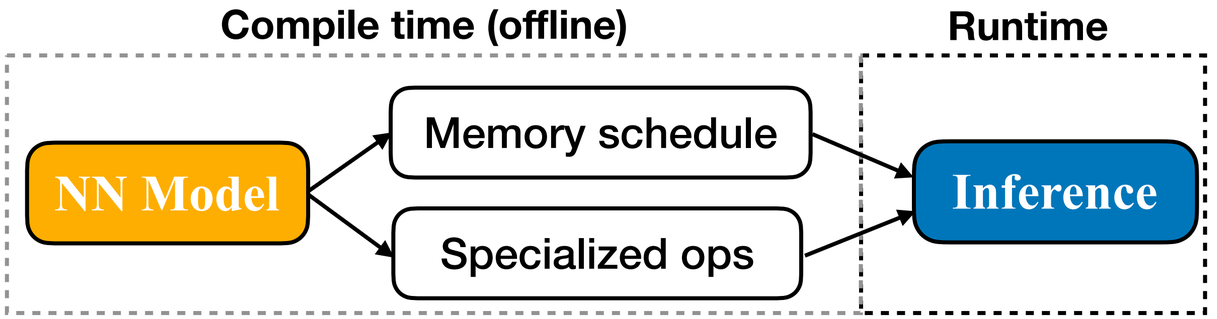
(a) Existing libraries based on **runtime interpretation**
e.g., *TF-Lite Micro, CMSIS-NN*



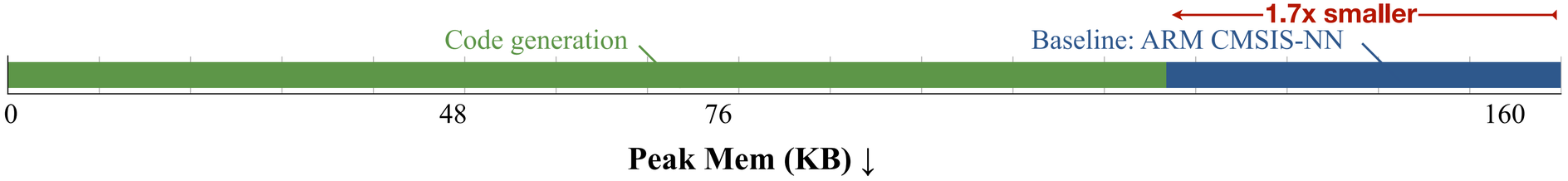
(b) TinyEngine: **Model-adaptive** code generation.

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime



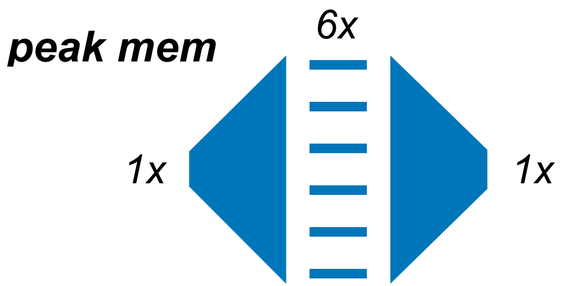
(b) TinyEngine: **Model-adaptive** code generation.



TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

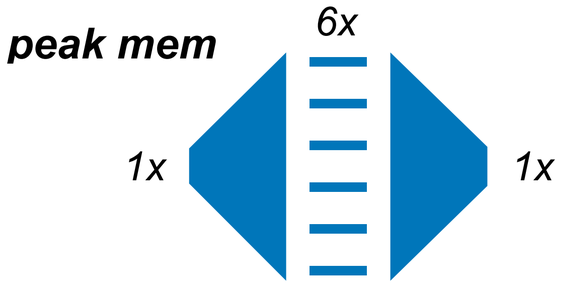
The dilemma of *inverted bottleneck*

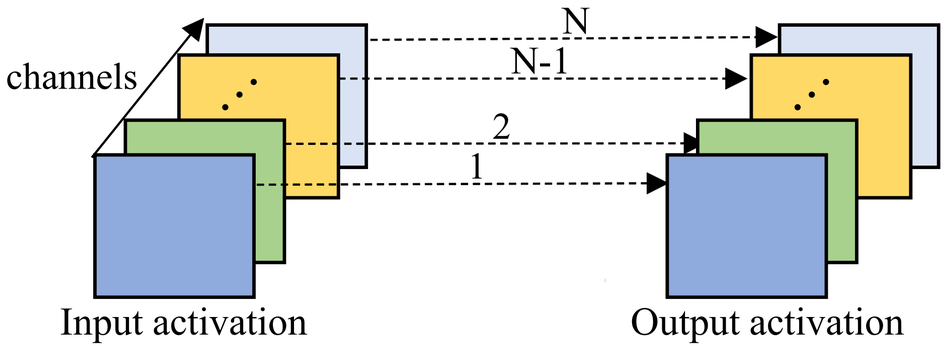


TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

The dilemma of *inverted bottleneck*





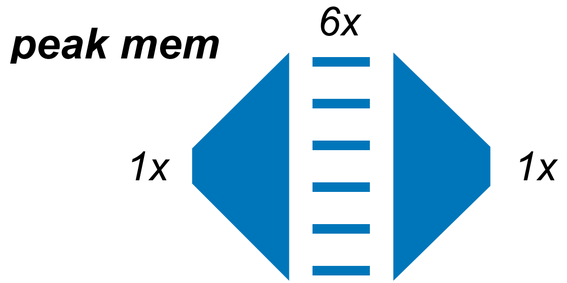
(a) Depth-wise convolution

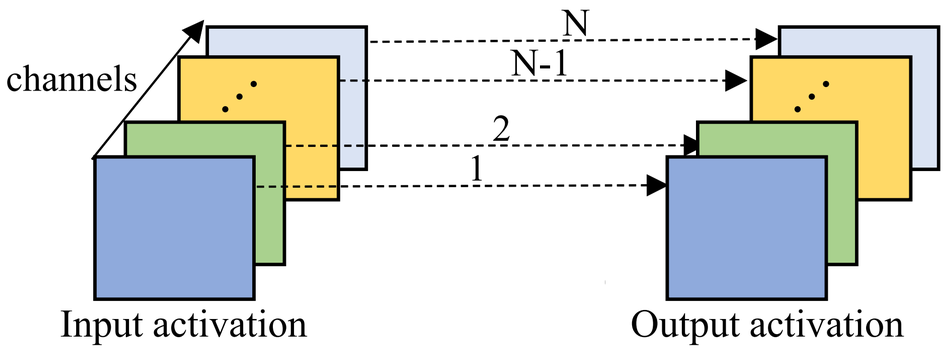
Peak Memory: 2N

TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

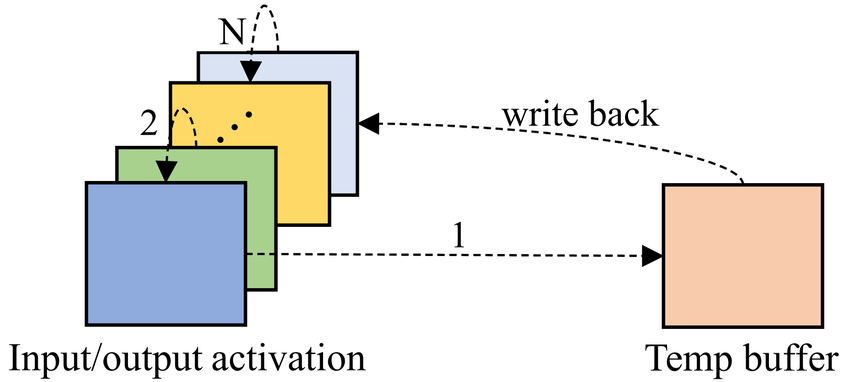
The dilemma of *inverted bottleneck*





(a) Depth-wise convolution

Peak Memory: 2N



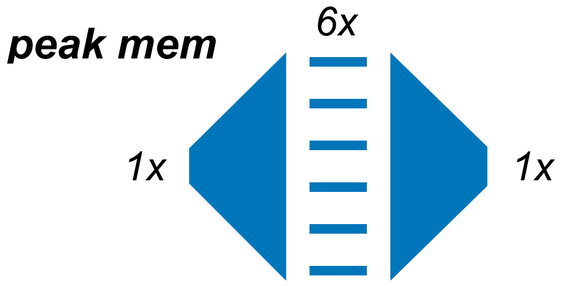
(b) In-place depth-wise convolution

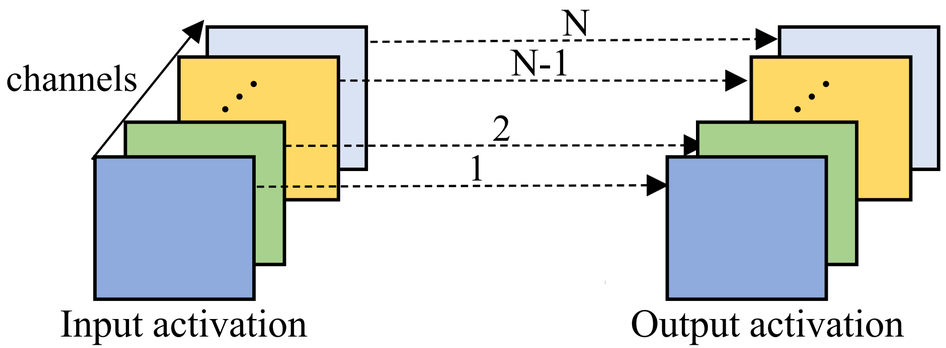
Peak Memory: N+1

TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

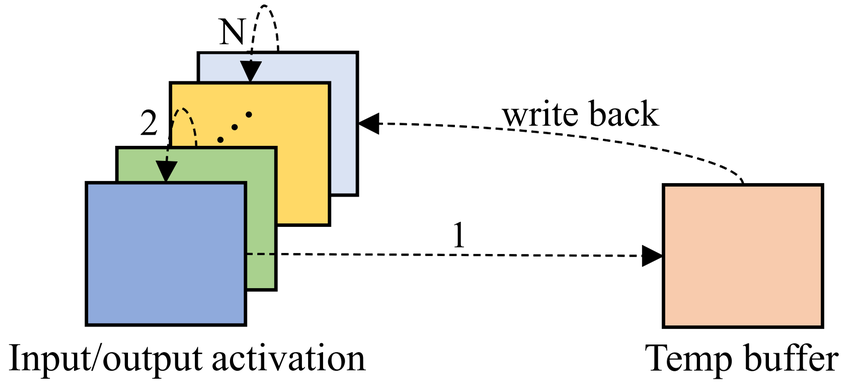
The dilemma of *inverted bottleneck*





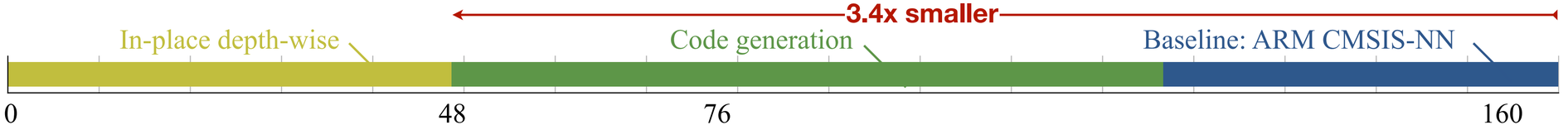
(a) Depth-wise convolution

Peak Memory: 2N



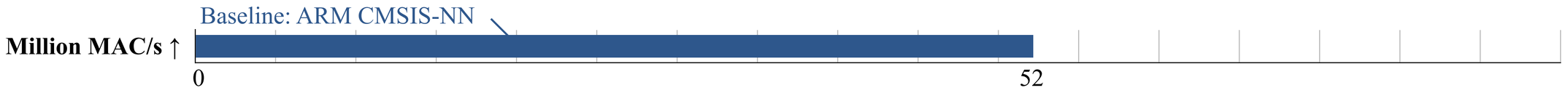
(b) In-place depth-wise convolution

Peak Memory: N+1



TinyEngine: Faster Inference Speed

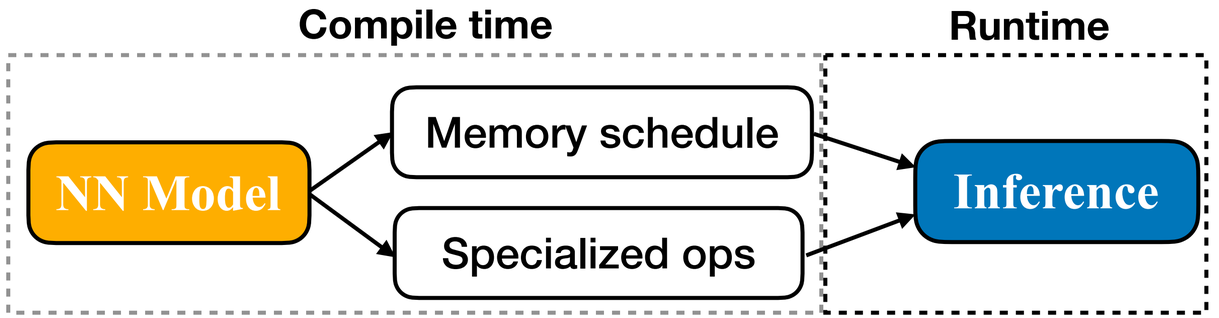
Analyzing **Million MAC/s** improved by each technique



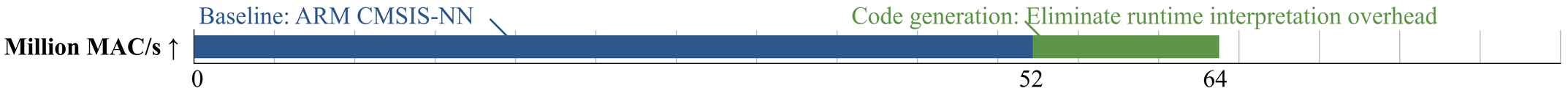
TinyEngine: Faster Inference Speed

Analyzing **Million MAC/s** improved by each technique

(1) Code generator-based compilation -> Eliminate overheads of runtime interpretation



TinyEngine: Model-adaptive code generation.



TinyEngine: Faster Inference Speed

Analyzing **Million MAC/s** improved by each technique

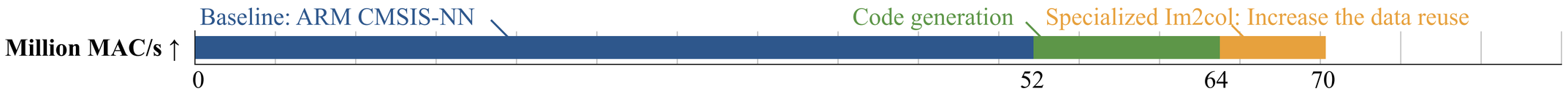
(2) Model-adaptive memory scheduling -> Increase data reuse for each layer

(a) Model-level memory scheduling

$$M = \max\left(\text{kernel size}_{L_i}^2 \cdot \text{in channels}_{L_i}; \forall L_i \in \boldsymbol{L}\right)$$

(b) Tile size configuration for Im2col

$$\text{tiling size of feature map width}_{L_j} = \lfloor M / \left(\text{kernel size}_{L_j}^2 \cdot \text{in channels}_{L_j}\right) \rfloor$$

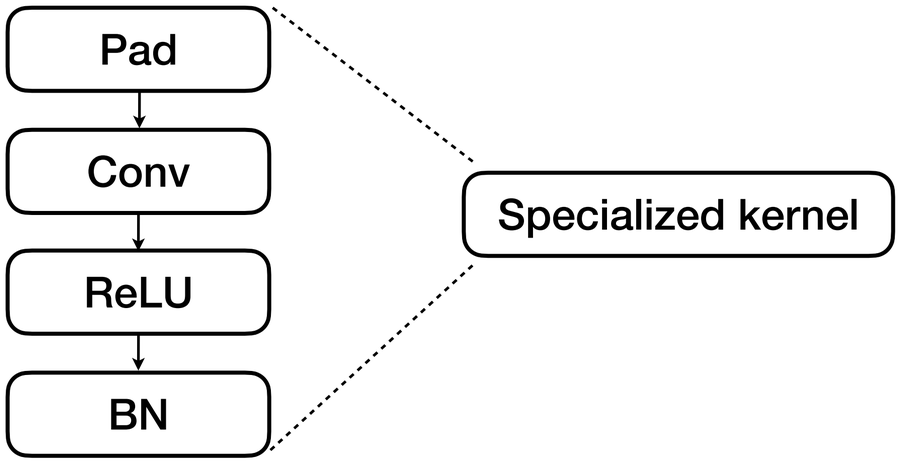


TinyEngine: Faster Inference Speed

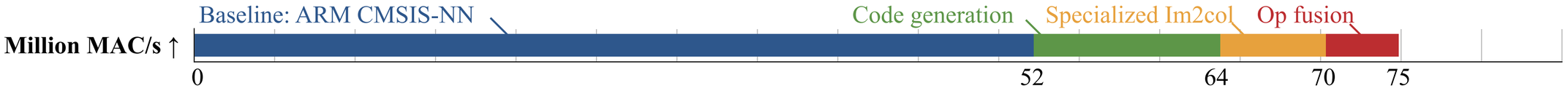
Analyzing **Million MAC/s** improved by each technique

(3) Computation Kernel Specialization: Operation fusion

e.g., fuse Pad+Conv+ReLU+BN



- Minimize memory footprint
- Optimize the overall computation



TinyEngine: Faster Inference Speed

Analyzing **Million MAC/s** improved by each technique

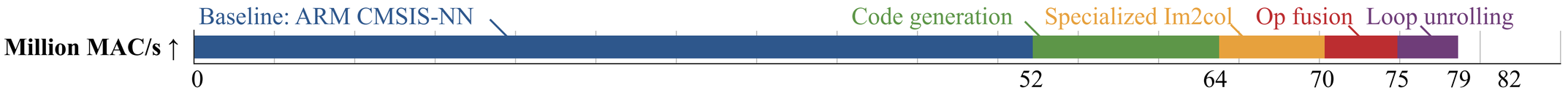
(3) Computation Kernel Specialization: Loop unrolling

```
/* dot products of convolution */
for (i = 0; i < kernel_x; i++)
    for(j = 0; j < kernel_y; j++)
        sum += x[i][j] * w[i][j];
```

e.g., fully unroll for 3x3 conv

```
/* dot products of convolution */
sum += x[0][0] * w[0][0];
sum += x[0][1] * w[0][1];
                ⋮
sum += x[2][2] * w[2][2];
```

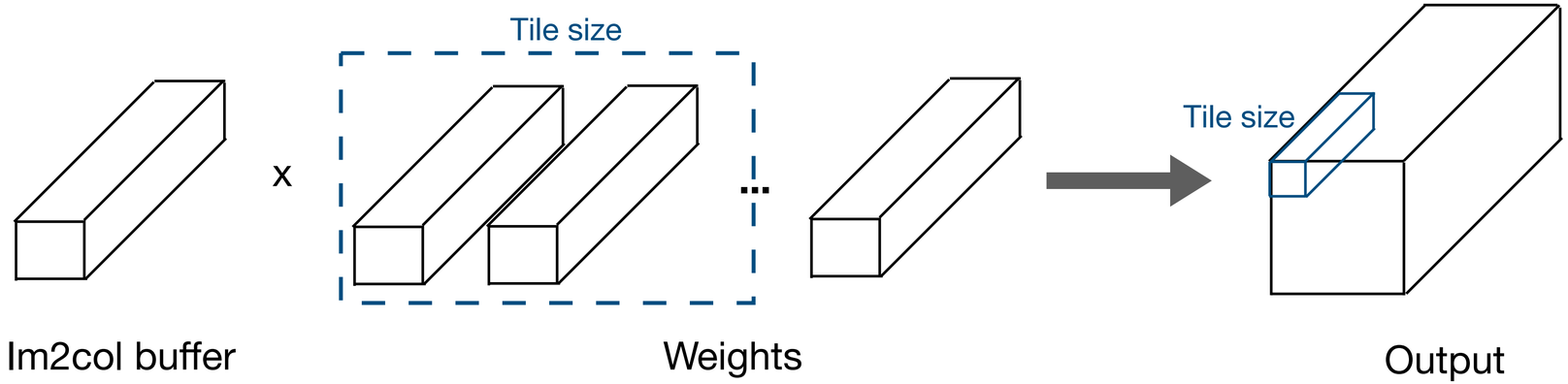
- Eliminate the branch instruction overheads of loops



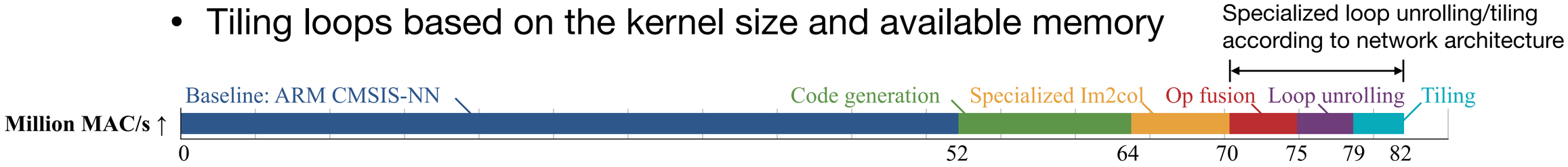
TinyEngine: Faster Inference Speed

Analyzing **Million MAC/s** improved by each technique

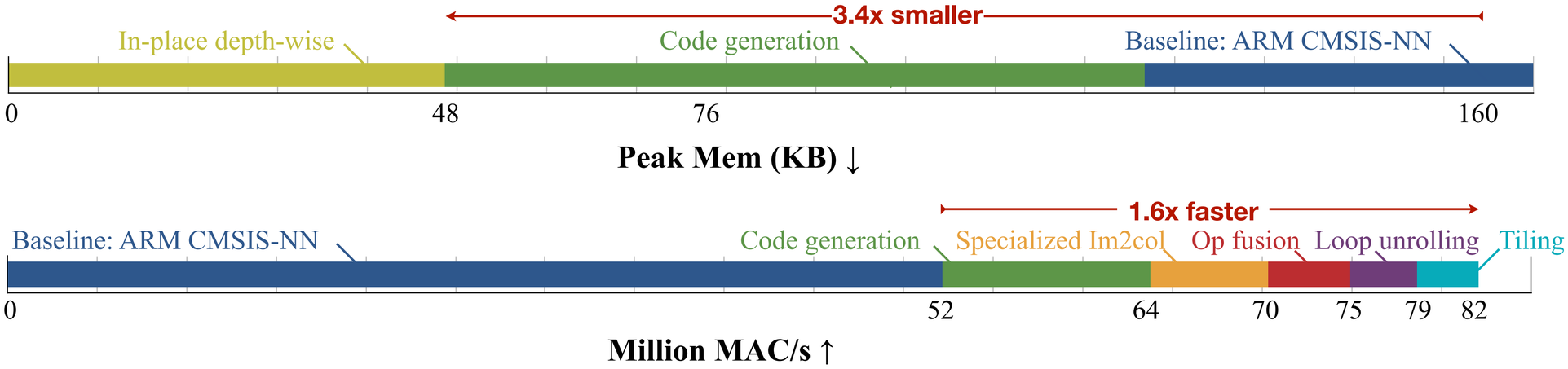
(3) Computation Kernel Specialization: Loop tiling for each layer



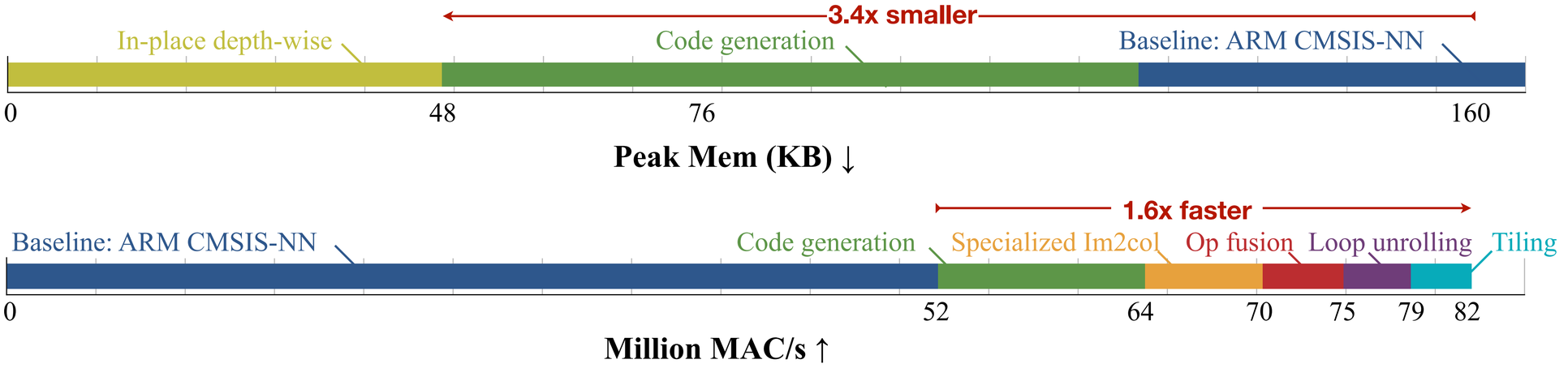
- Tiling loops based on the kernel size and available memory



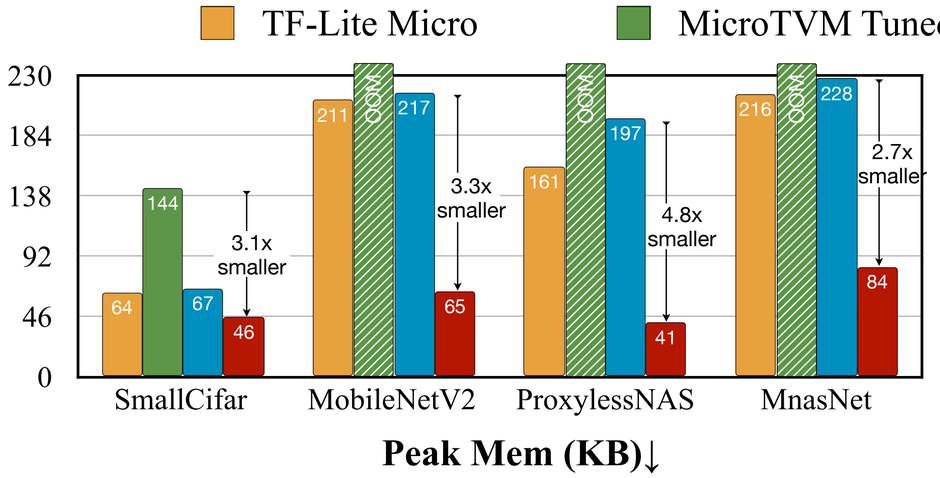
TinyEngine: A Memory-Efficient Inference Library

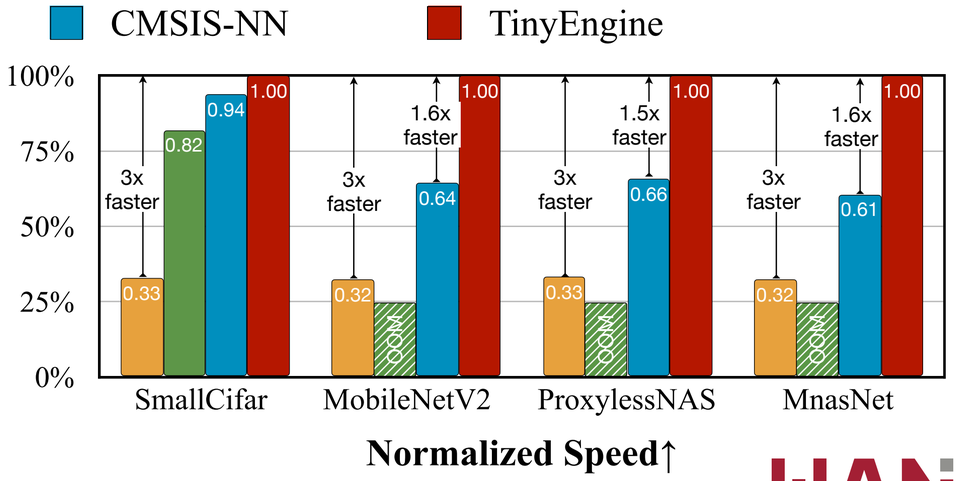


TinyEngine: A Memory-Efficient Inference Library

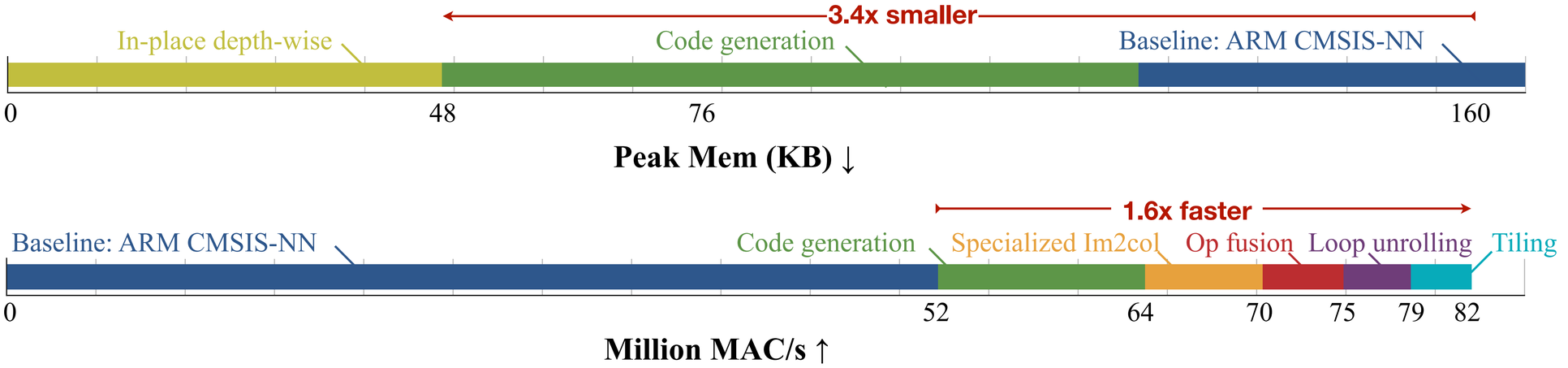


- Consistent improvement on different networks

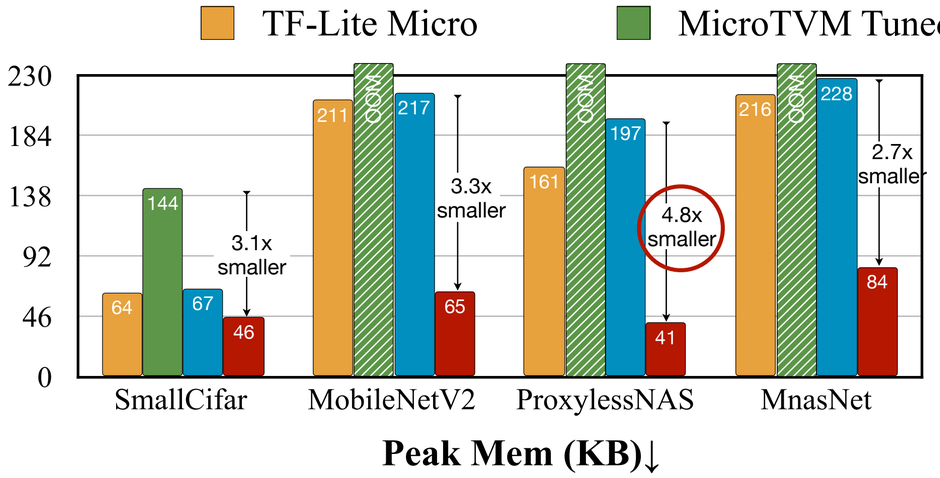


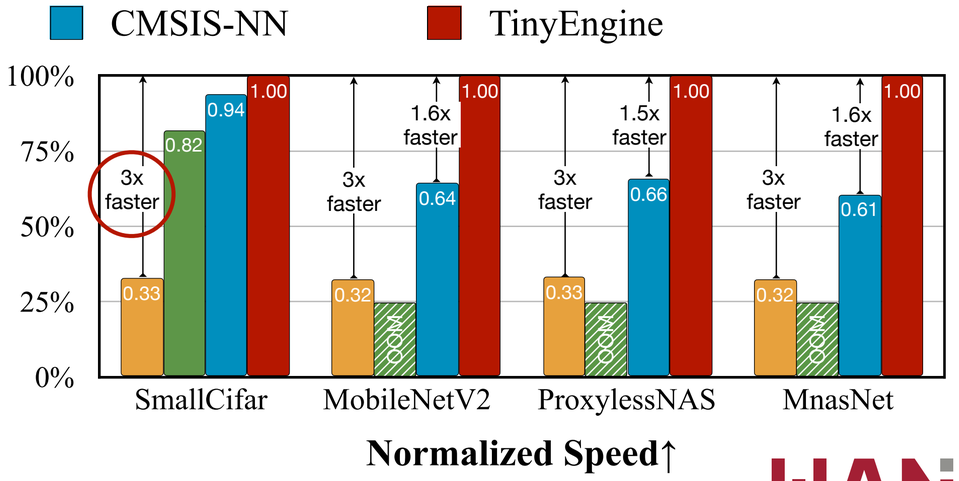


TinyEngine: A Memory-Efficient Inference Library



- Consistent improvement on different networks





Experimental Results

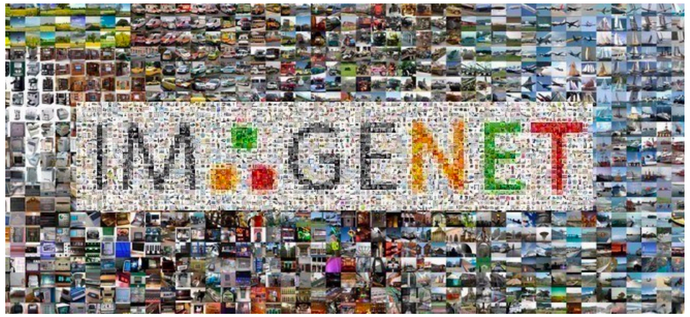
We focus on large-scale datasets to reflect real-life use cases.

Datasets:

(1) ImageNet-1000

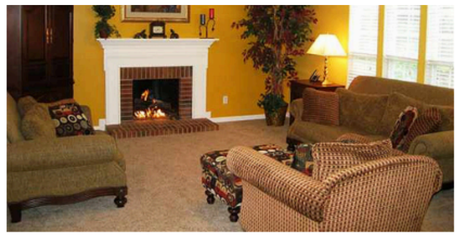
(2) Wake Words

- Visual: Visual Wake Words
- Audio: Google Speech Commands

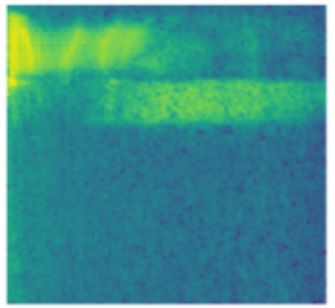


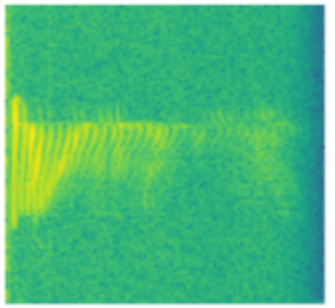
yes no

(a) ‘Person’



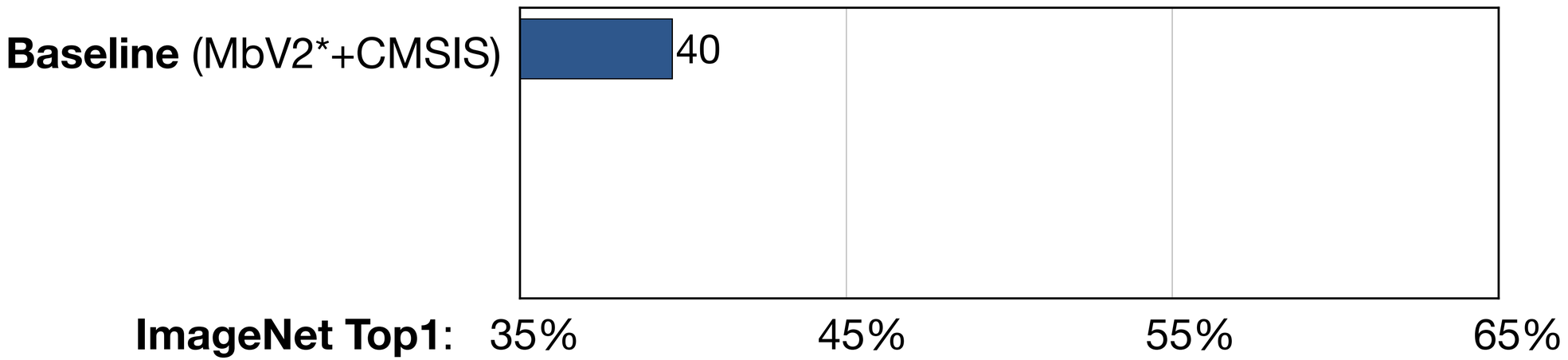
(b) ‘Not-person’





System-Algorithm Co-design Gives the Best Results

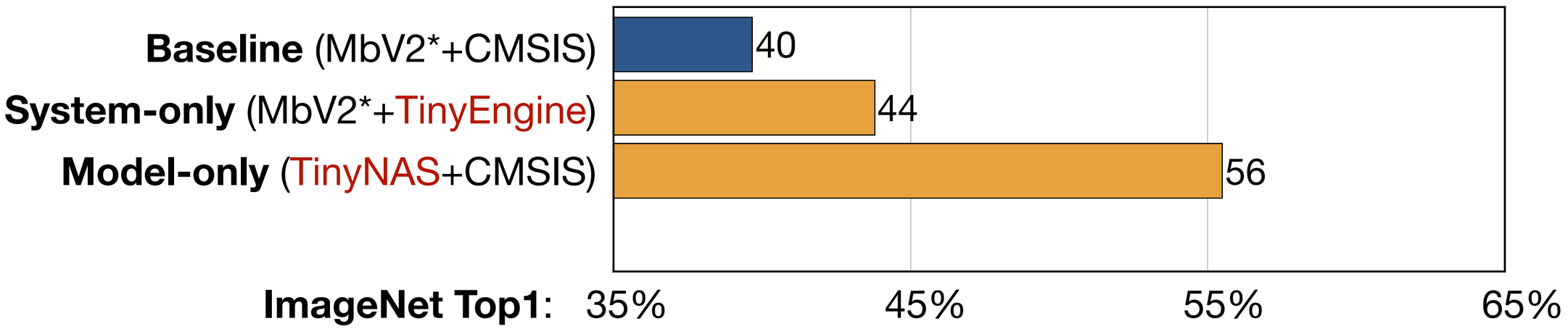
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



* scaled down version: width multiplier 0.3, input resolution 80

System-Algorithm Co-design Gives the Best Results

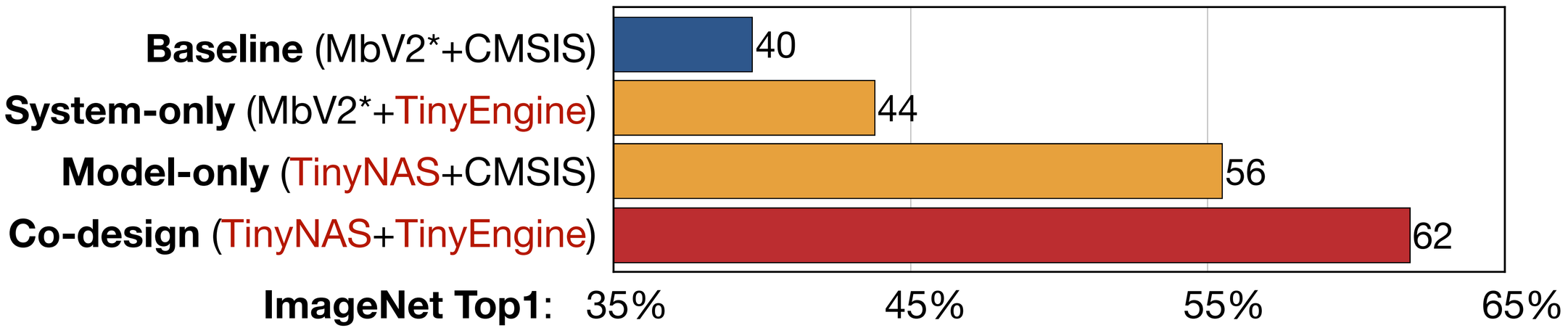
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



* scaled down version: width multiplier 0.3, input resolution 80

System-Algorithm Co-design Gives the Best Results

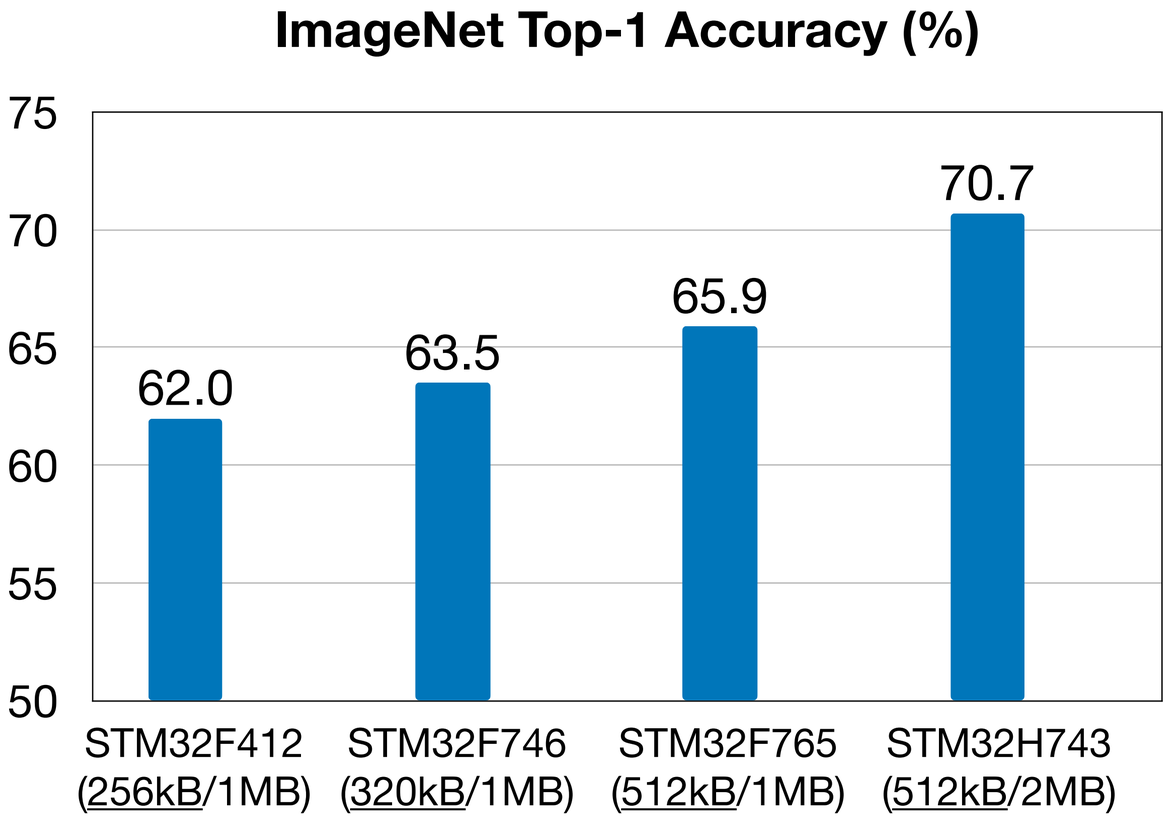
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



* scaled down version: width multiplier 0.3, input resolution 80

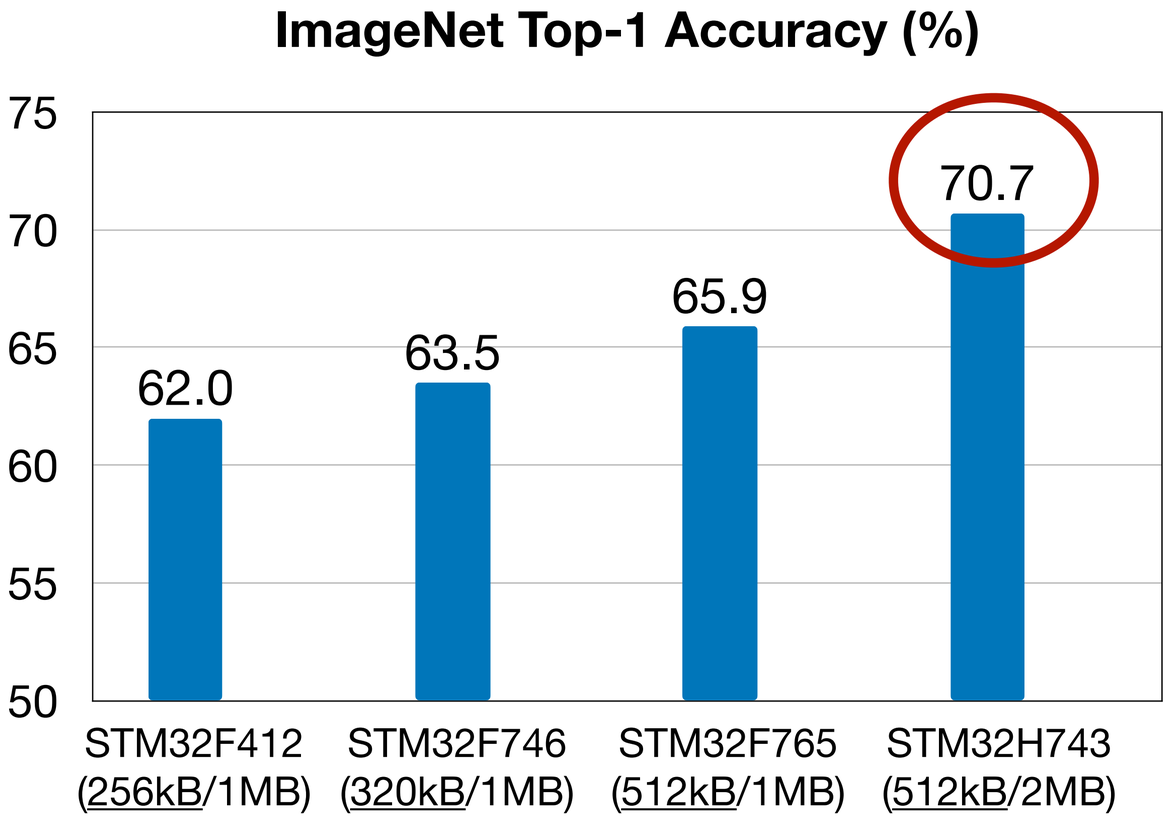
Handling Diverse Hardware

- Specializing models (int4) for different MCUs (SRAM/Flash)



Handling Diverse Hardware

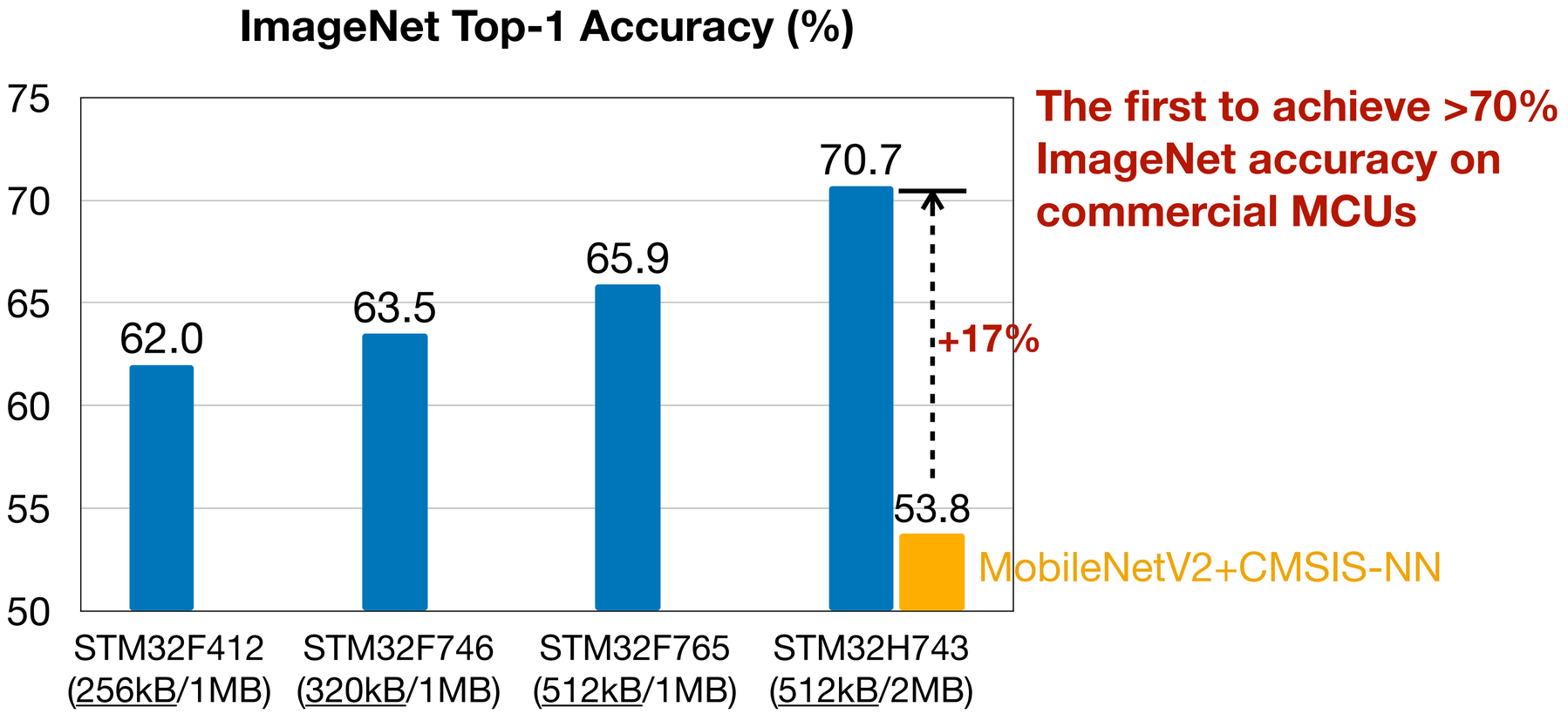
- Specializing models (int4) for different MCUs (SRAM/Flash)



The first to achieve >70% ImageNet accuracy on commercial MCUs

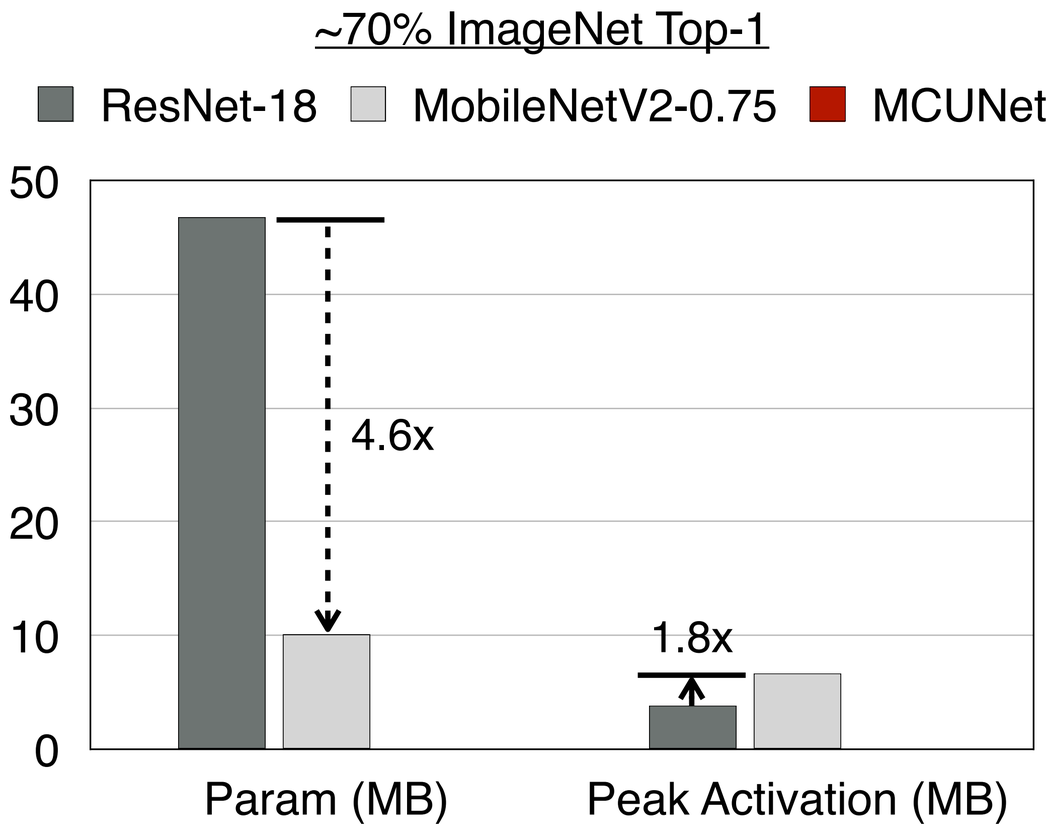
Handling Diverse Hardware

- Specializing models (int4) for different MCUs (SRAM/Flash)

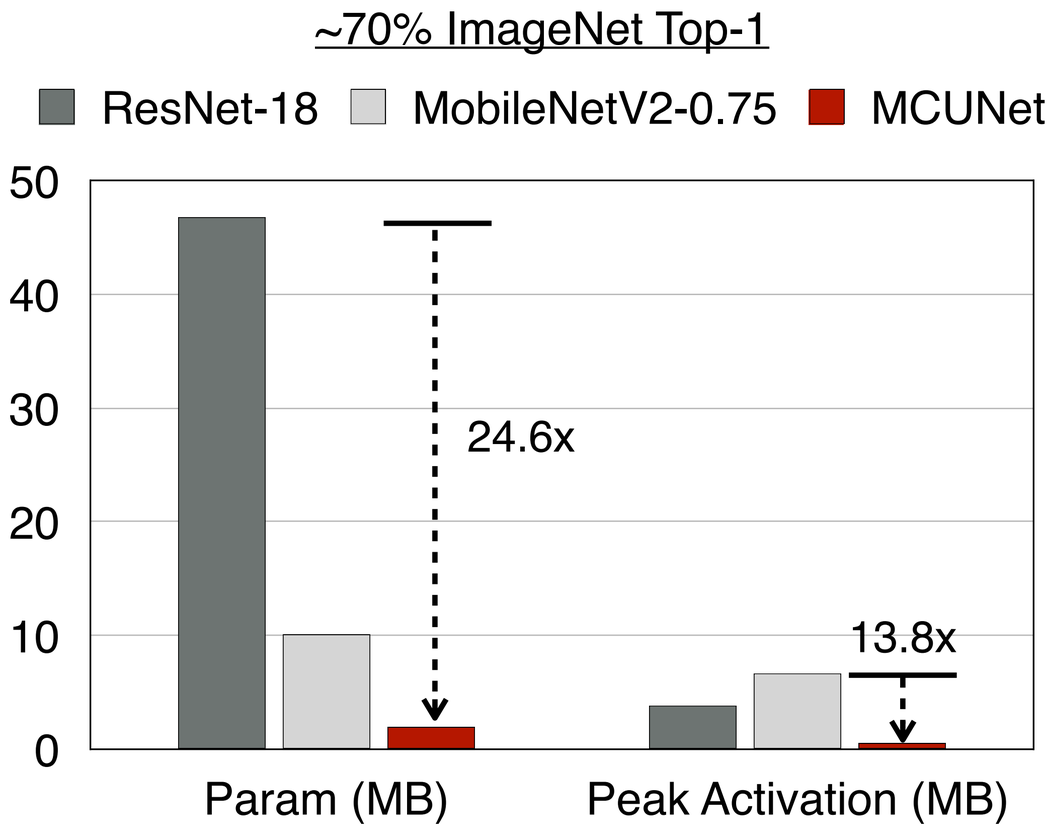


The first to achieve >70% ImageNet accuracy on commercial MCUs

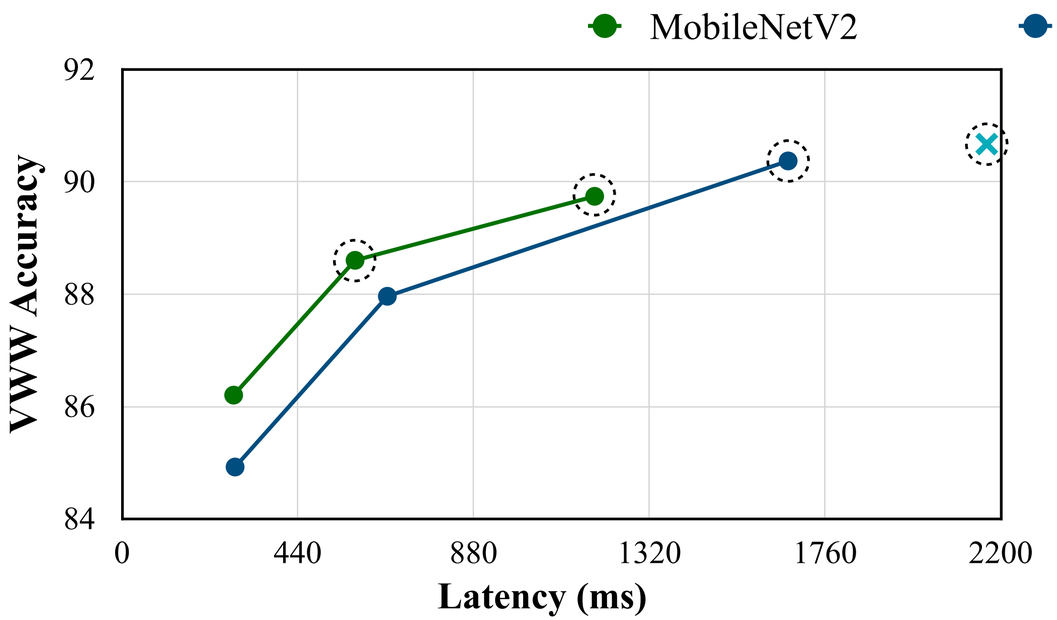
Reduce Both Model Size and Activation Size



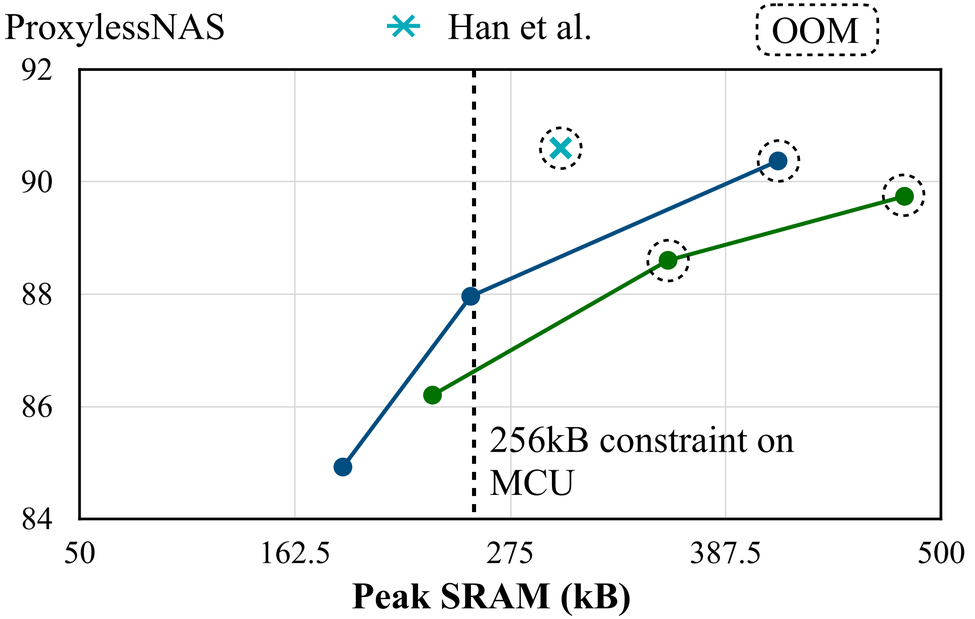
Reduce Both Model Size and Activation Size



Visual Wake Words (VWW)

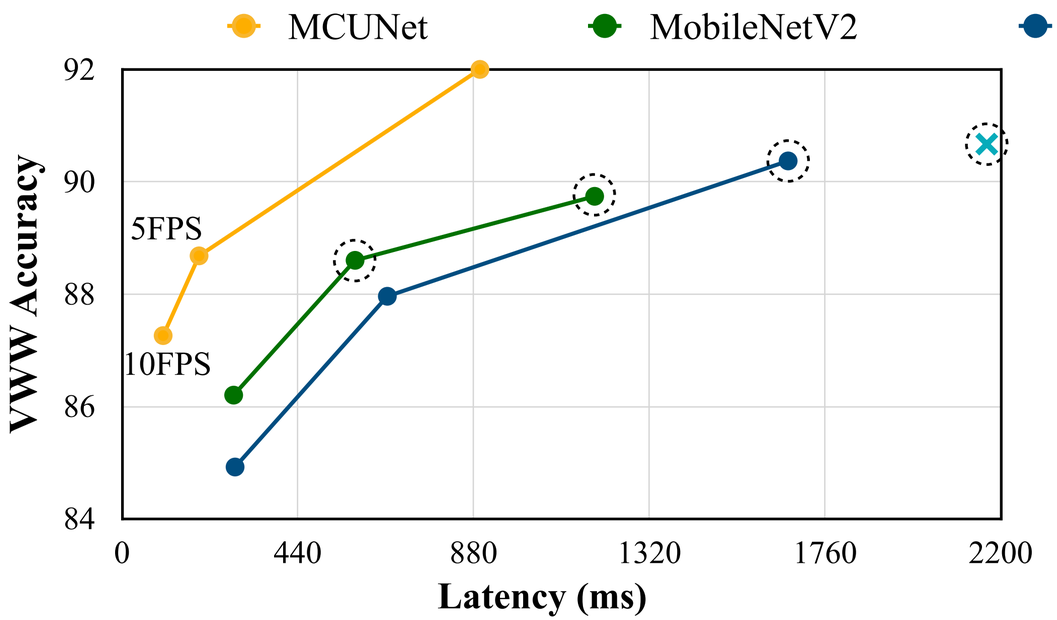


(a) Trade-off: accuracy *vs.* measured latency

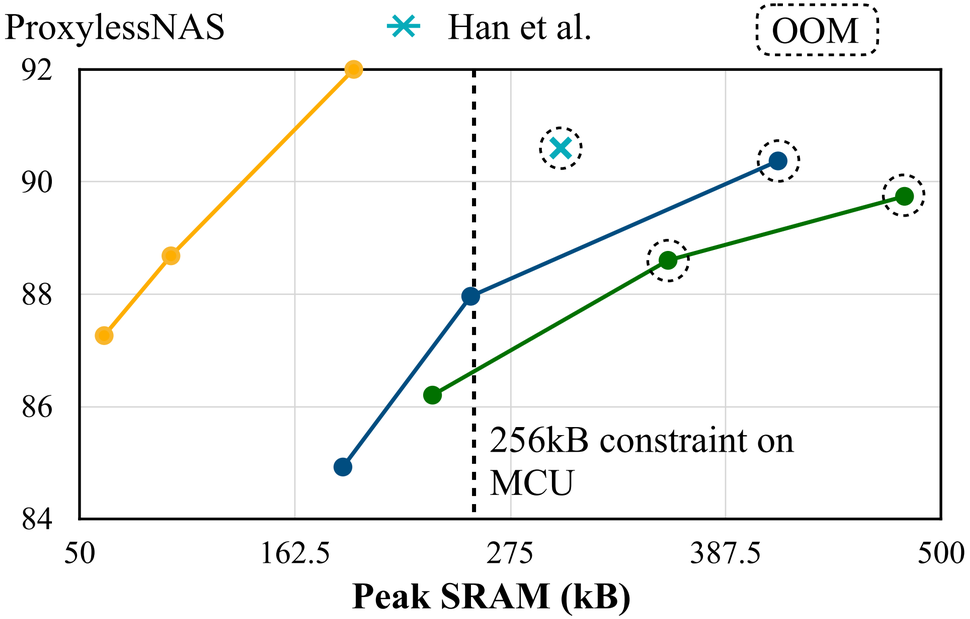


(b) Trade-off: accuracy *vs.* peak memory

Visual Wake Words (VWW)

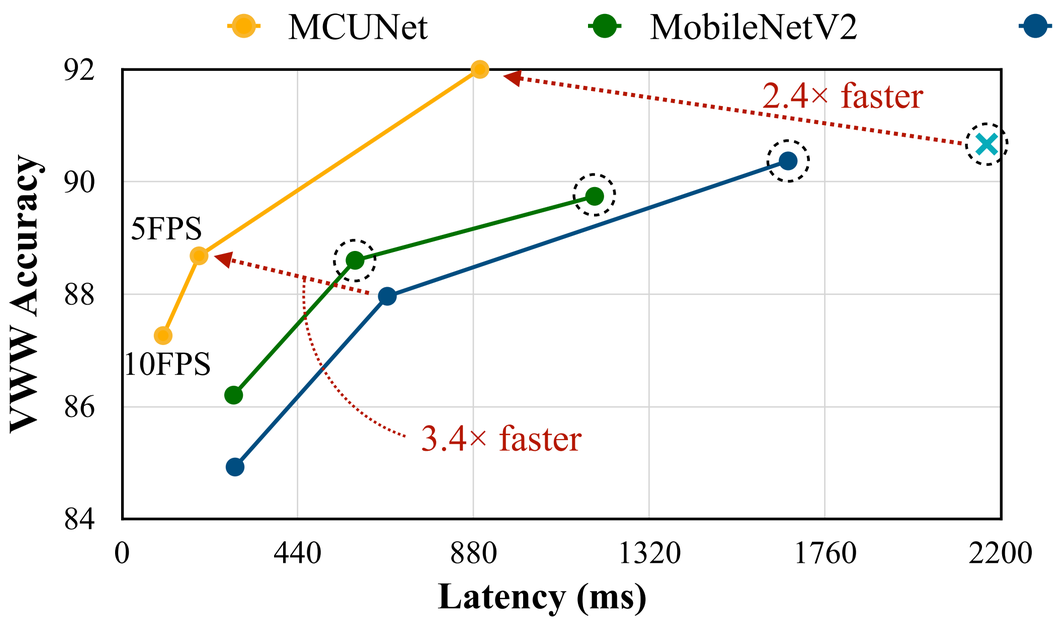


(a) Trade-off: accuracy *vs.* measured latency

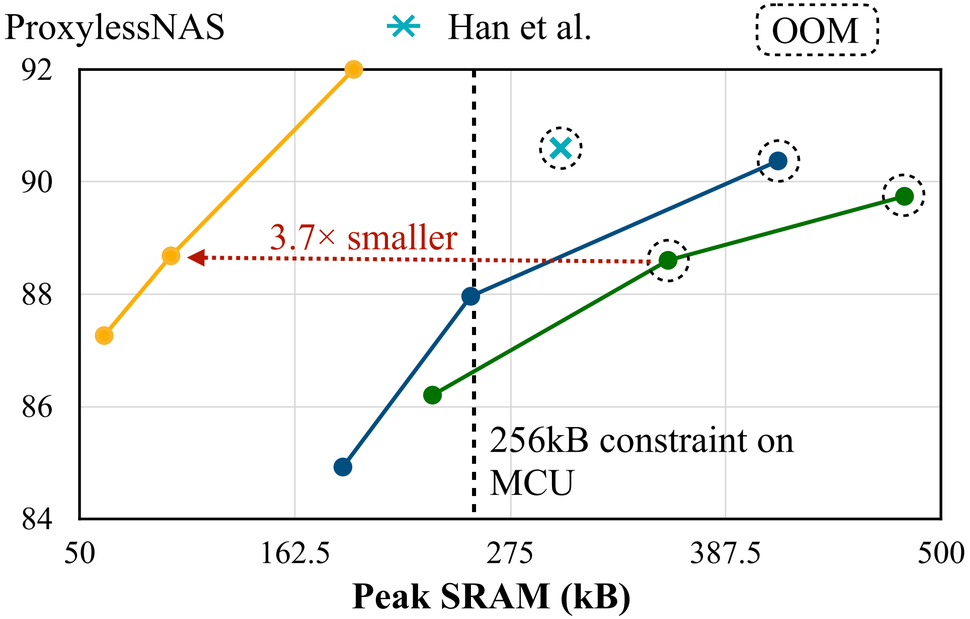


(b) Trade-off: accuracy *vs.* peak memory

Visual Wake Words (VWW)

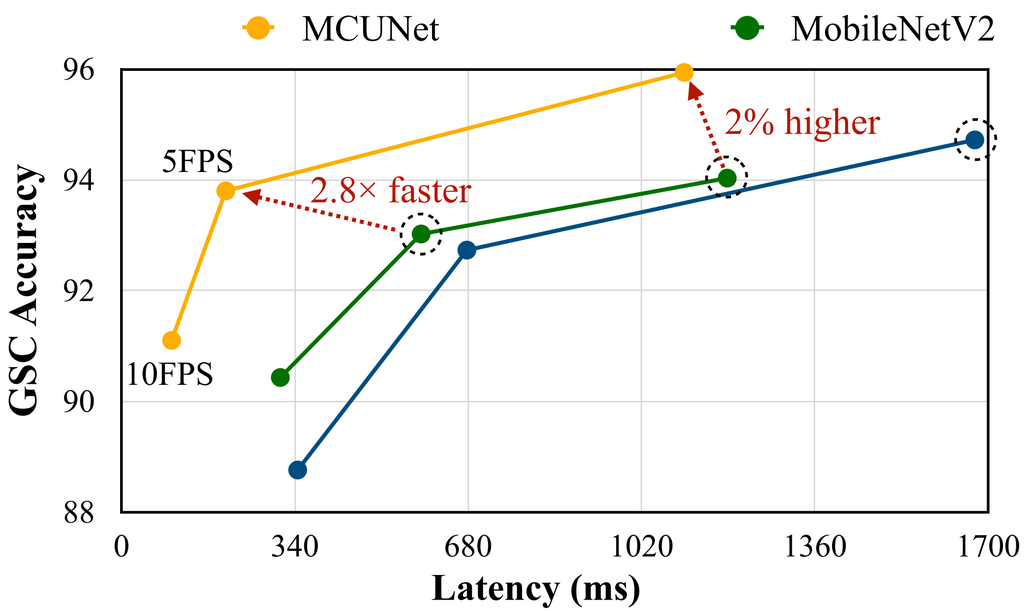


(a) Trade-off: accuracy *vs.* measured latency

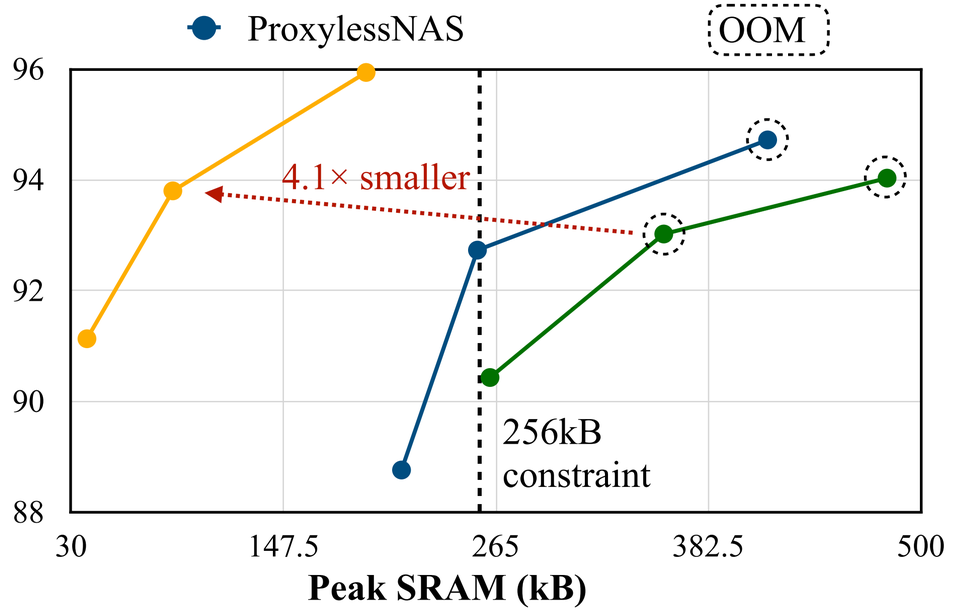


(b) Trade-off: accuracy *vs.* peak memory

Audio Wake Words (Speech Commands)



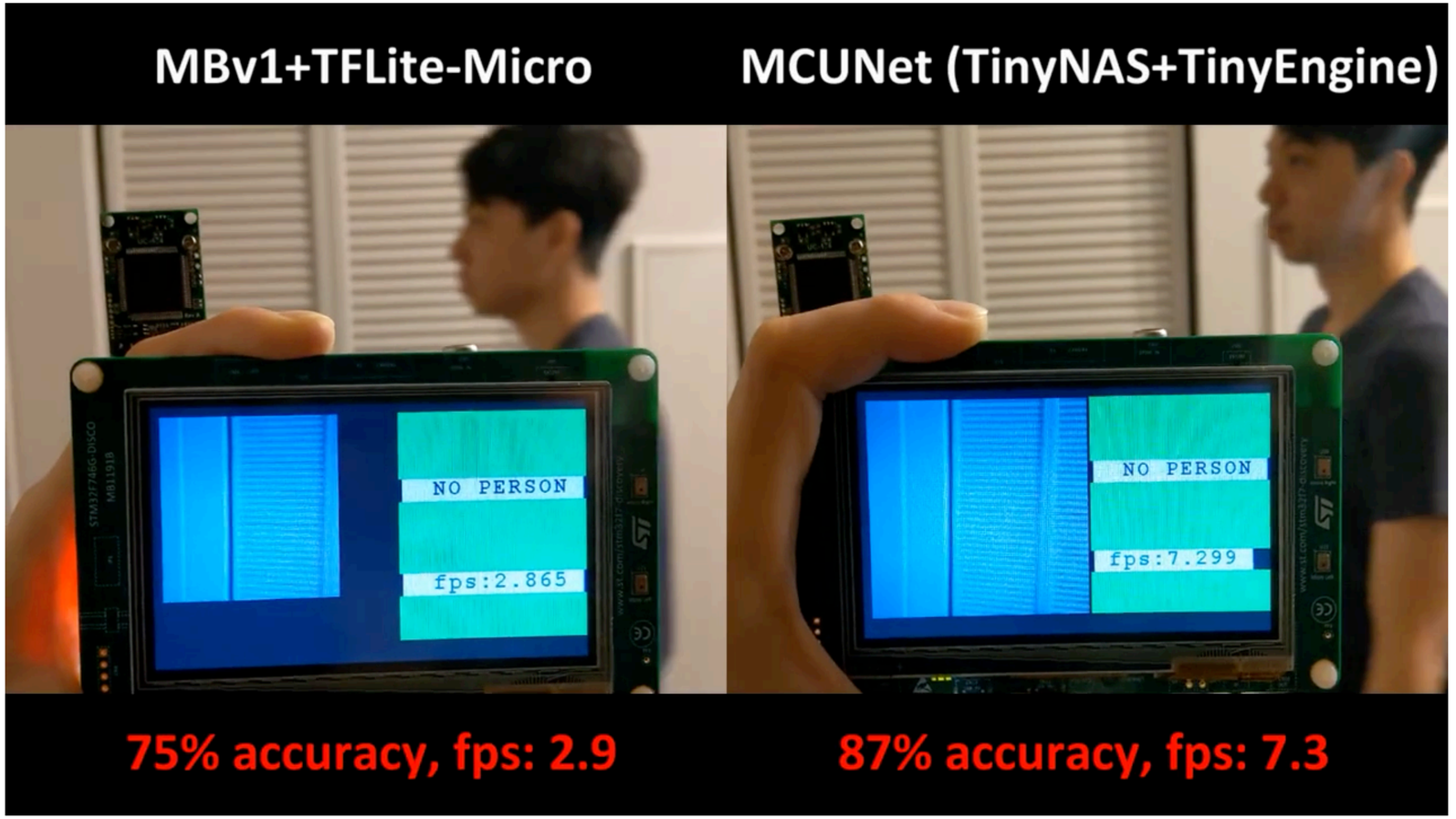
(a) Trade-off: accuracy *vs.* measured latency



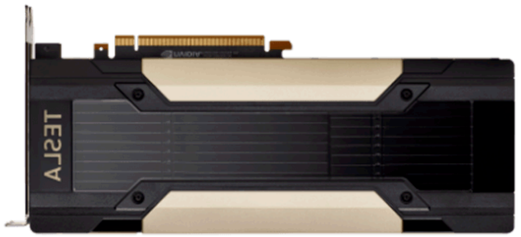
(b) Trade-off: accuracy *vs.* peak memory

Visual Wake Word Detection

- Detecting whether a person is present in the frame



MCUNet: Tiny Deep Learning on IoT Devices



Cloud AI

ResNet

Mobile AI

MobileNet

Tiny AI

MCUNet

- Our study suggests that the **era of tiny machine learning** on IoT devices has arrived

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