GS-WGAN:
A Gradient-Sanitized Approach for Learning Differentially Private Generators

Dingfan Chen[1] Tribhuvanesh Orekondy[2] Mario Fritz[1]

[1] CISPA Helmholtz Center for Information Security [2] Max Planck Institute for Informatics

Presenter: **Dingfan Chen (dingfan.chen@cispa.saarland)**

Outline

- **Motivation & Task**
- **Background**
- **Problem**
- **Approach**
- **Results**

Motivation & Task

- **Motivation**
  - Progress in training ML models in sensitive domains (e.g., healthcare) is impeded by scarcity of dataset
  - Can we release synthetic datasets with rigorous privacy guarantees?

- **Task: Privacy-preserving data generation**
  - **Differential privacy:**

    (protect privacy of the individual)
    - Rigorous privacy guarantee

  - **Generative adversarial networks (GANs):**

    (preserve useful information of the population)
    - High-dimensional data
    - Arbitrary downstream task

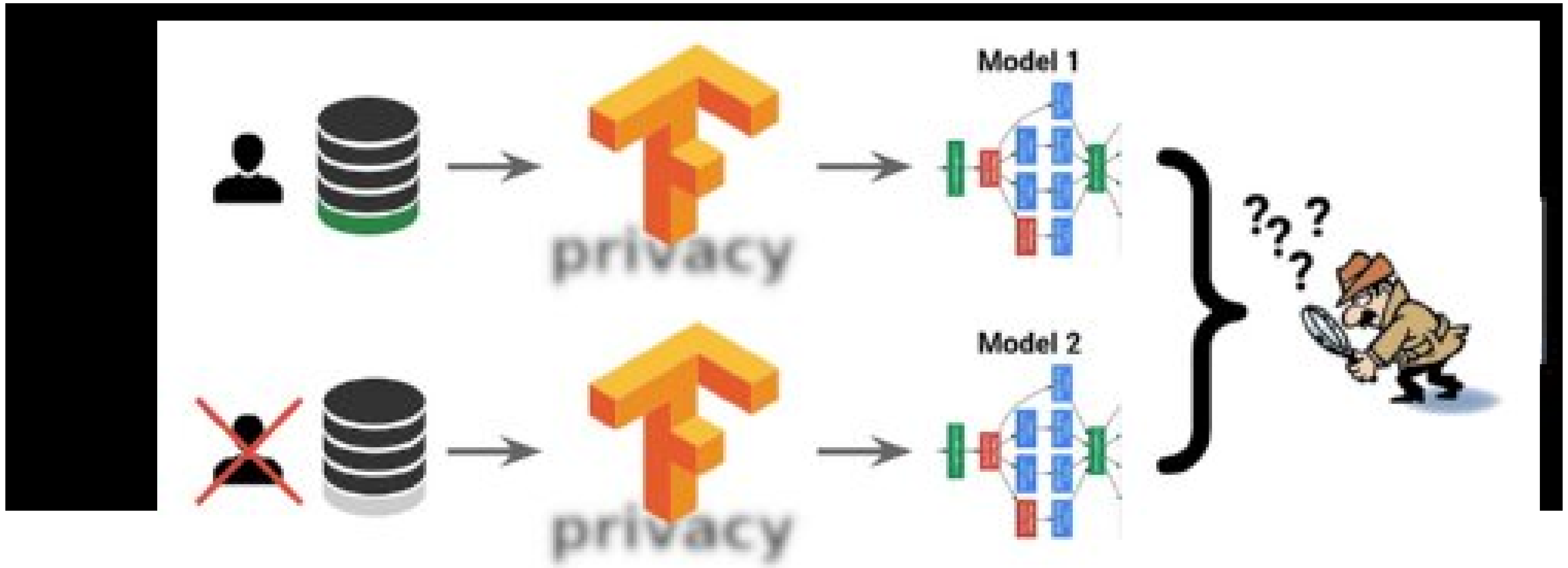
Background

- **Differential Privacy[1]**

Definition 3.1. (Differential Privacy (DP) [11]) A randomized mechanism $\mathcal{M}$ with range $\mathcal{R}$ is (ε, δ)-DP, if

$$Pr[\mathcal{M}(S) \in \mathcal{O}] \leq e^{\varepsilon} \cdot Pr[\mathcal{M}(S') \in \mathcal{O}] + \delta \tag{1}$$

holds for any subset of outputs $\mathcal{O} \subseteq \mathcal{R}$ and for any adjacent datasets S and S', where S and S' differ from each other with only one training example. $\mathcal{M}$ is the GAN training algorithm in our case, ε corresponds to the upper bound of privacy loss, and δ is the probability of breaching DP constraints.



[1] Dwork et al., "The Algorithmic Foundations of Differential Privacy", Foundations and Trends in Theoretical Computer Science

Image source:
1. http://www.cleverhans.io/privacy/2018/04/29/privacy-and-machine-learning.html
2. https://hackernoon.com/differential-privacy-with-tensorflow-20-multi-class-text-classification-privacy-yk7a37uh

Background

- **Differential Privacy[1]**

Definition 3.1. (Differential Privacy (DP) [11]) A randomized mechanism $\mathcal{M}$ with range $\mathcal{R}$ is (ε, δ)-DP, if

$$Pr[\mathcal{M}(S) \in \mathcal{O}] \leq e^{\varepsilon} \cdot Pr[\mathcal{M}(S') \in \mathcal{O}] + \delta \tag{1}$$

holds for any subset of outputs $\mathcal{O} \subseteq \mathcal{R}$ and for any adjacent datasets S and S', where S and S' differ from each other with only one training example. $\mathcal{M}$ is the GAN training algorithm in our case, ε corresponds to the upper bound of privacy loss, and δ is the probability of breaching DP constraints.

Definition 3.3. (Gaussian Mechanism [14, 31]) Let $f : X \to \mathbb{R}^d$ be an arbitrary d-dimensional function with sensitivity being

$$\Delta_2 f = \max_{S,S'} \|f(S) - f(S')\|_2 \tag{3}$$

maximal effect of each individual

over all adjacent datasets S and S'. The Gaussian Mechanism $\mathcal{M}_\sigma$, parameterized by σ, adds noise into the output, i.e.,

$$\mathcal{M}_\sigma(x) = f(x) + \frac{\Delta_2 f}{\varepsilon}\mathcal{N}(0, 2\ln(1.25/\delta)). \tag{4}$$

introduce randomness

$\mathcal{M}$ is (ε, δ)-DP.

[1] Dwork et al., "The Algorithmic Foundations of Differential Privacy", Foundations and Trends in Theoretical Computer Science

Background

- **Differential Privacy[1] (Properties)**
  - Graceful **composition**:
    (For iterative method: accumulate privacy cost at each step)

    Theorem 3.1. (Composition) For a sequence of mechanisms $\mathcal{M}_1, \ldots, \mathcal{M}_k$ s.t. $\mathcal{M}_i$ is (λ, ε_i)-RDP $\forall i$, the composition $\mathcal{M}_1 \circ \ldots \circ \mathcal{M}_k$ is $(\lambda, \sum_i \varepsilon_i)$-RDP.

  - **Post-processing** invariance:
    (Risk doesn't increase if you don't touch the data again)

    Theorem 3.2. (Post-processing [15]) If $\mathcal{M}$ satisfies (ε, δ)-DP, $F \circ \mathcal{M}$ will satisfy (ε, δ)-DP for any function F with $\circ$ denoting the composition operator.

[1] Dwork et al., "The Algorithmic Foundations of Differential Privacy", Foundations and Trends in Theoretical Computer Science

Problem

- Privacy-preserving data generation
  - Rigorous privacy guarantee → Differential Privacy (DP)[1]
  - High-dimensional data
  - Arbitrary downstream task

  } → Generative Adversarial Networks (GANs)[2]

- Existing Approach

[1] Dwork et al., “The Algorithmic Foundations of Differential Privacy”, Foundations and Trends in Theoretical Computer Science
[2] Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
[3] Abadi et al., “Deep Learning with Differential Privacy”, CCS 2016

Problem

- Privacy-preserving data generation
  - Rigorous privacy guarantee → **Differential Privacy (DP)**[1]
  - High-dimensional data
  - Arbitrary downstream task

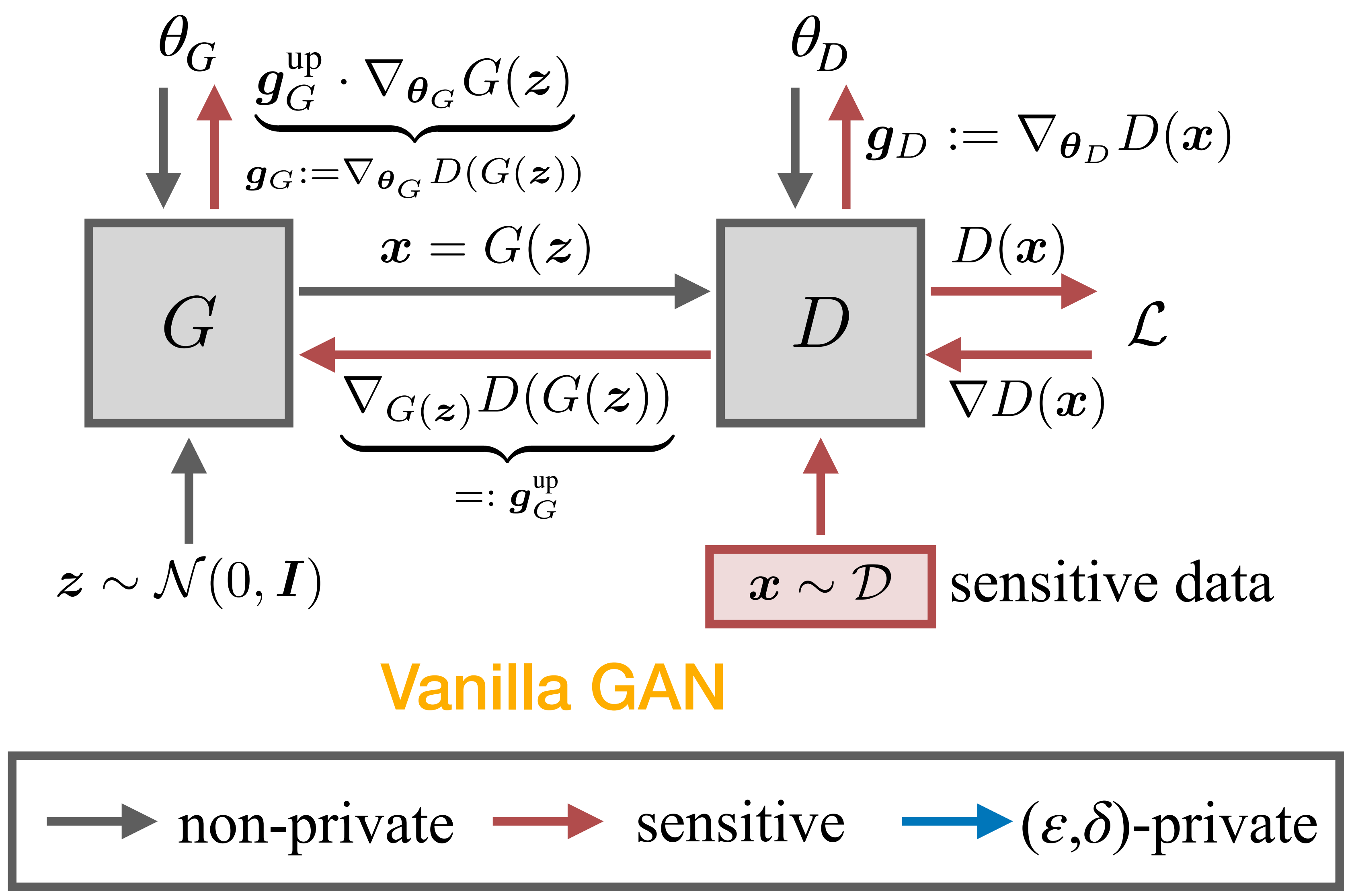
  (High-dimensional data, Arbitrary downstream task) → **Generative Adversarial Networks (GANs)**[2]

- Existing Approach
  - Gradient

    $$\boldsymbol{g}^{(t)} := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G)$$

  - Gradient descent step

    $$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \eta \cdot \boldsymbol{g}^{(t)}$$



Vanilla GAN

[1] Dwork et al., "The Algorithmic Foundations of Differential Privacy", Foundations and Trends in Theoretical Computer Science
[2] Goodfellow et al., "Generative Adversarial Nets", NIPS 2014
[3] Abadi et al., "Deep Learning with Differential Privacy", CCS 2016

Problem

- Privacy-preserving data generation
  - Rigorous privacy guarantee → **Differential Privacy (DP)**[1]
  - High-dimensional data
  - Arbitrary downstream task → **Generative Adversarial Networks (GANs)**[2]

- Existing Approach
  - Differentially private stochastic gradient descent (DP-SGD)[3]
    - Gradient

$$\boldsymbol{g}^{(t)} := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G)$$

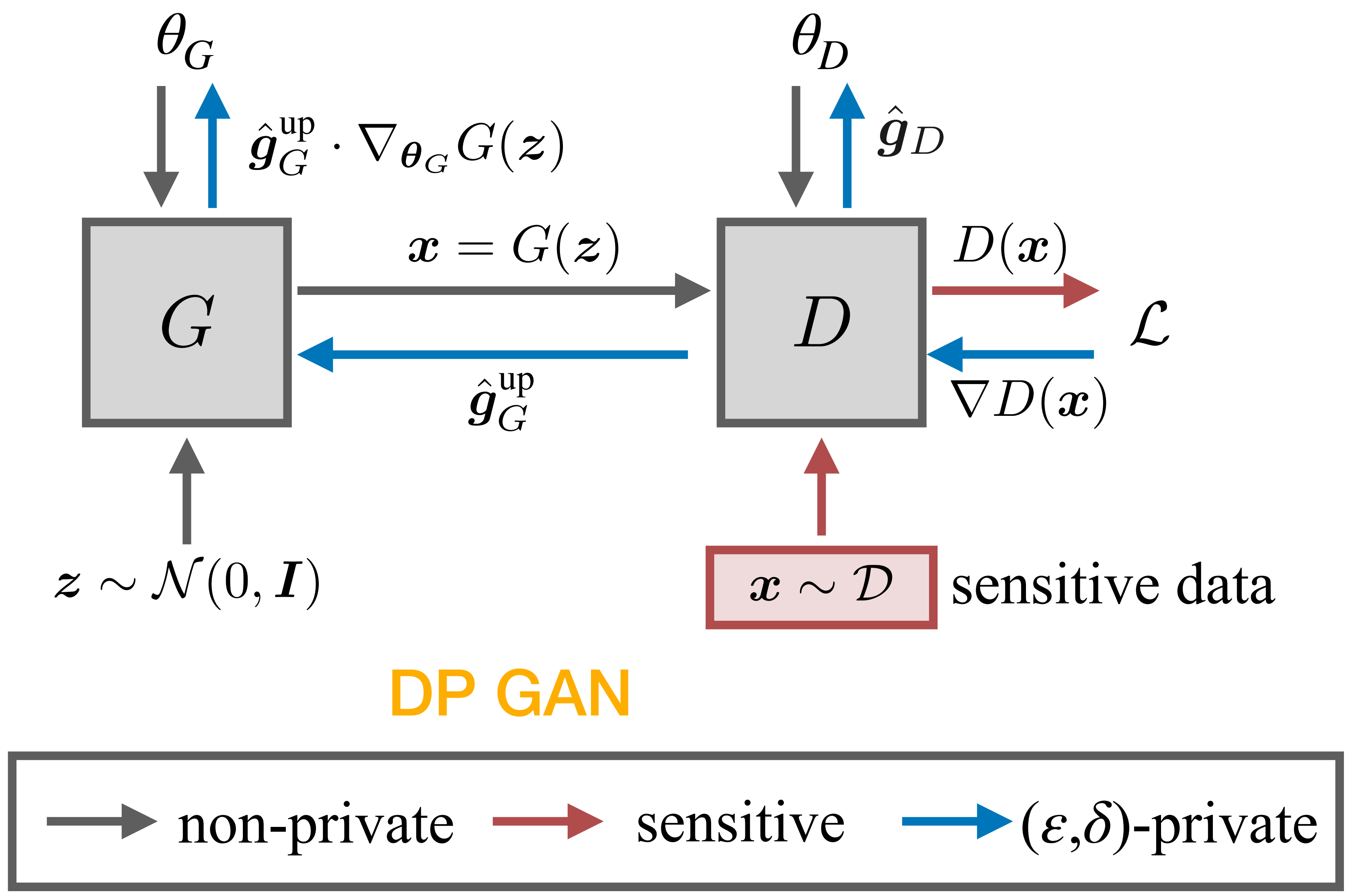
    - Sanitization mechanism

$$\hat{\boldsymbol{g}}^{(t)} := \mathcal{M}_{\sigma,C}(\boldsymbol{g}^{(t)}) = \mathrm{clip}(\boldsymbol{g}^{(t)}, \boxed{C}) + \mathcal{N}(0, \sigma^2 C^2 \boldsymbol{I})$$

clipping bound

    - Gradient descent step

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \eta \cdot \hat{\boldsymbol{g}}^{(t)}$$



[1] Dwork et al., “The Algorithmic Foundations of Differential Privacy”, Foundations and Trends in Theoretical Computer Science
[2] Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014
[3] Abadi et al., “Deep Learning with Differential Privacy”, CCS 2016

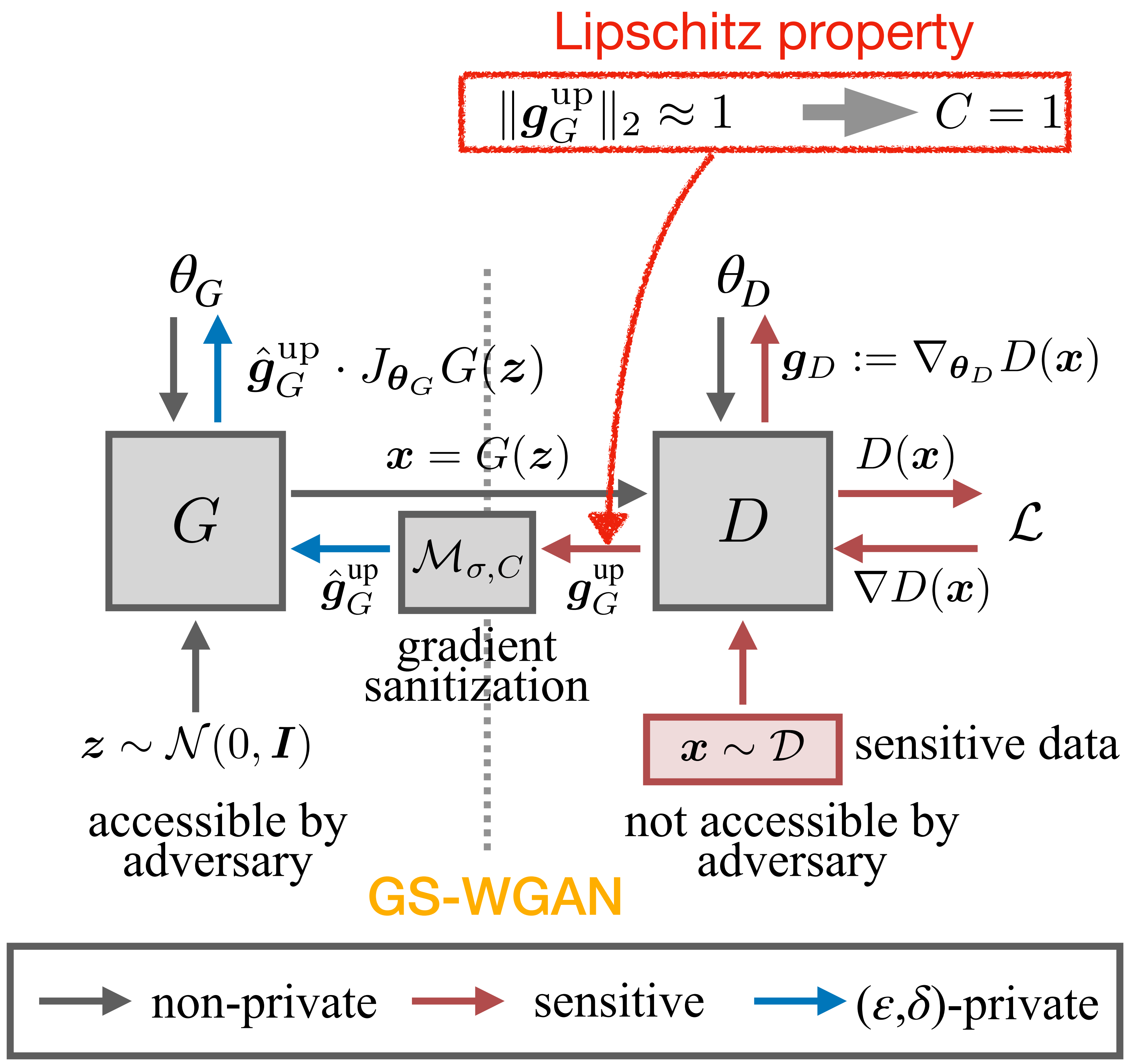
Approach

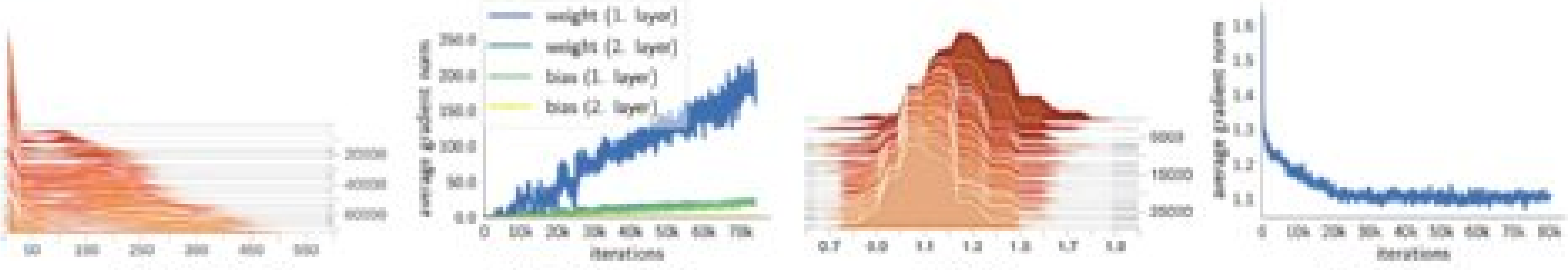
- Insight:
    - Only the ***generator*** need to be publicly-released

- Our framework:
    1. Selectively applying sanitization mechanism
        - Train the ***discriminator*** non-privately
        - Sanitize gradients transferred to the ***generator***

$$\hat{\boldsymbol{g}}_G = \mathcal{M}_{\sigma,C}\underbrace{(\nabla_{G(\boldsymbol{z})}\mathcal{L}_G(\boldsymbol{\theta}_G))}_{\boldsymbol{g}_G^{\text{up}}} \cdot \underbrace{J_{\boldsymbol{\theta}_G} G(\boldsymbol{z};\boldsymbol{\theta}_G)}_{\boldsymbol{J}_G^{\text{local}}}$$

    2. Bounding sensitivity using Wasserstein distance[1,2]





(a) DP-SGD (b) DP-SGD (c) Ours (d) Ours

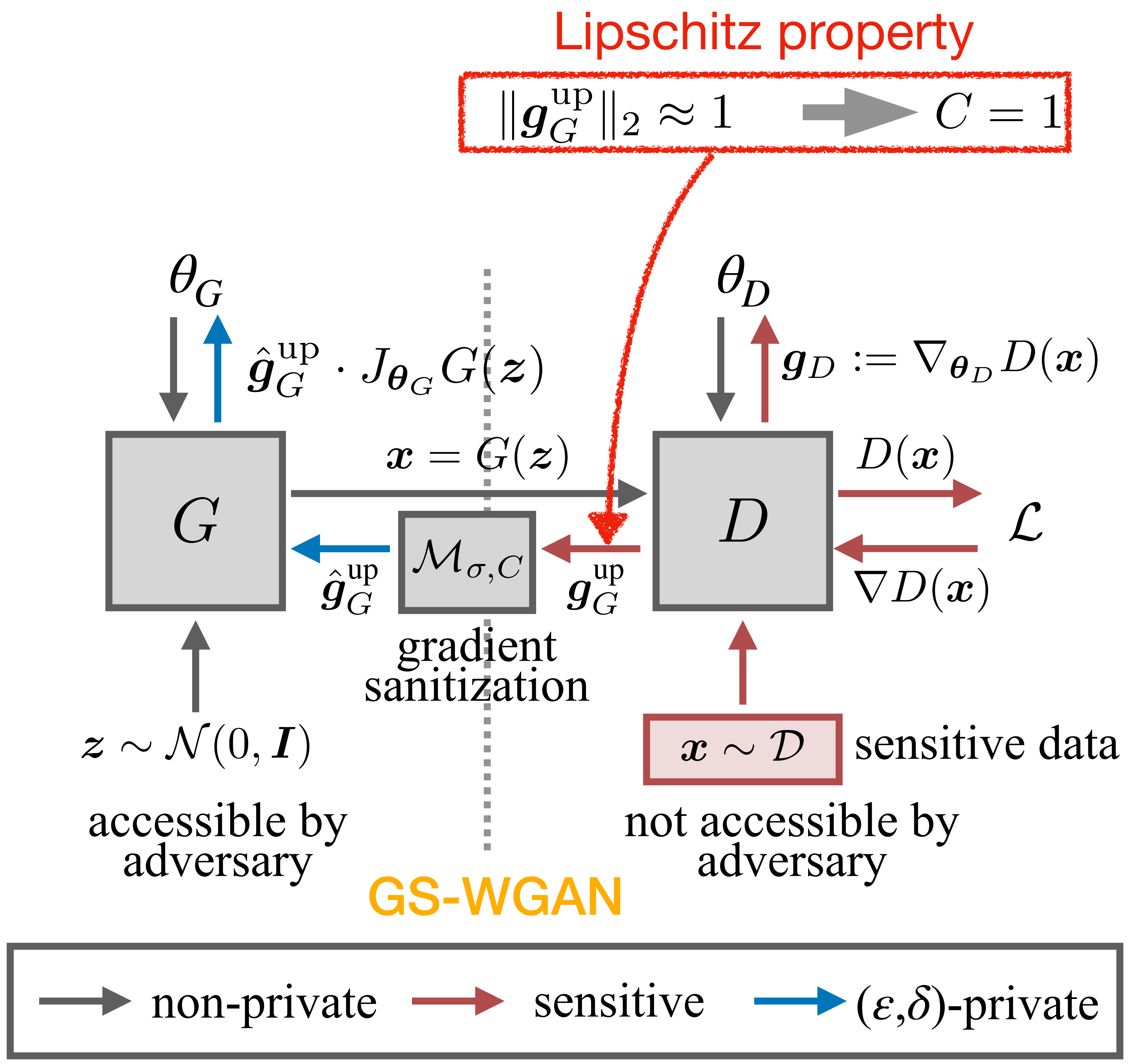
[1] Arjovsky et al., “Wasserstein Generative Adversarial Network”, ICML 2017
[2] Gulrajani et al., “Improved Training of Wasserstein GANs”, NIPS 2017

Approach

- Insight:
  - Only the ***generator*** need to be publicly-released
- Our framework:
  1. Selectively applying sanitization mechanism
     - Train the ***discriminator*** non-privately
     - Sanitize gradients transferred to the ***generator***

$$\hat{\boldsymbol{g}}_G = \underbrace{\mathcal{M}_{\sigma,C}(\nabla_{G(\boldsymbol{z})}\mathcal{L}_G(\boldsymbol{\theta}_G))}_{\boldsymbol{g}_G^{\text{up}}} \cdot \underbrace{J_{\boldsymbol{\theta}_G} G(\boldsymbol{z};\boldsymbol{\theta}_G)}_{\boldsymbol{J}_G^{\text{local}}}$$

  2. Bounding sensitivity using Wasserstein distance[1,2]

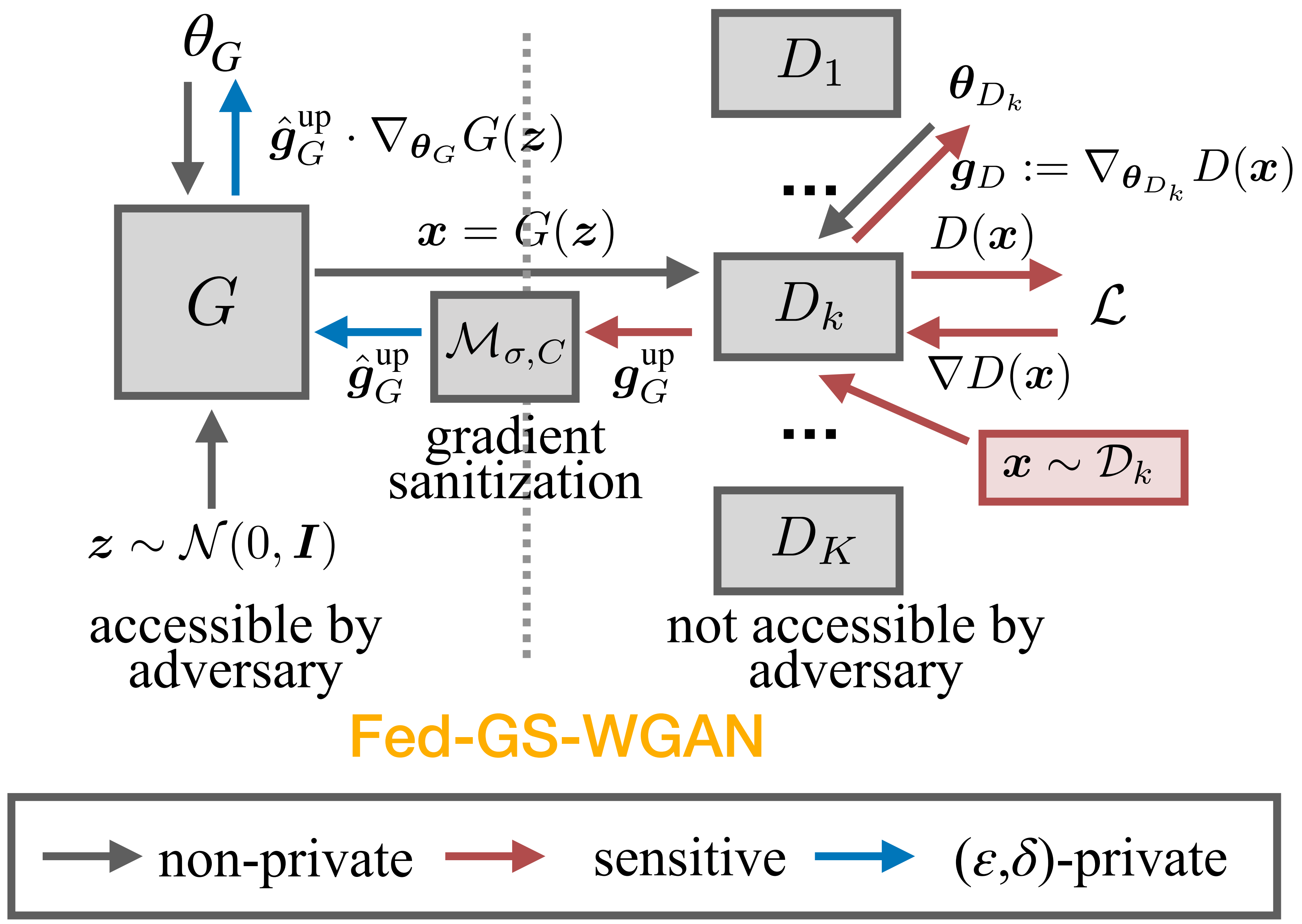


- Advantages:
  1. Maximally preserve the true gradient direction
  2. Bypass an intensive and fragile hyper-parameter search for clipping value
  3. Small clipping bias

[1] Arjovsky et al., "Wasserstein Generative Adversarial Network", ICML 2017
[2] Gulrajani et al., "Improved Training of Wasserstein GANs", NIPS 2017

Approach

- **Decentralized (Federated) setting**
  - Each user train a discriminator on its sensitive dataset locally
  - Communicate the sanitized gradient



- **Advantages**:
  - User-level DP guarantee under an *untrusted* server
  - Communication-efficient
    (gradients w.r.t. generated samples are *more compact* than gradients w.r.t model parameters[1])

[1] Augenstein et al., "Generative Models for Effective ML on Private, Decentralized Datasets", ICLR 2020

Results

- **Datasets:**
  - Images (MNIST, Fashion-MNIST, Fed-EMNIST)

- **Evaluation metrics:**
  - **Privacy**: Determined by ε with fixed δ
  - **Utility**:
    - Sample quality: realism of the generated samples
      Inception score (IS)[1,2], Frechet Inception Distance (FID)[3]
    - Usefulness for downstream tasks:
      Classification accuracy: MLP Acc, CNN Acc, Avg Acc, Calibrated Acc
      (trained on generated data and test on real data)

[1] Li et al., "Alice: Towards Understanding Adversarial Learning for Joint Distribution Matching", NIPS 2017
[2] Salimans et al., "Improved Techniques for Training GANs", NIPS 2016
[3] Heusel et al., "GANs Trained by a Two Time-scale Update Rule Converge to a Local Nash Equilibrium", NIPS 2017

Results

- **Centralized setting**

  Improves the IS by:
  - 94% on MNIST
  - 45% on Fashion-MNIST

  Improves the MLP Acc by:
  - 25% on MNIST
  - 16% on Fashion-MNIST

		IS↑	FID ↓	MLP ↑ Acc	CNN ↑ Acc	Avg ↑ Acc	Calibrated ↑ Acc
MNIST	Real	9.80	1.02	0.98	0.99	0.88	100 %
	G-PATE [1]	3.85	177.16	0.25	0.51	0.34	40%
	DP-SGD GAN	4.76	179.16	0.60	0.63	0.52	59%
	DP-Merf	2.91	247.53	0.63	0.63	0.57	66%
	DP-Merf AE	3.06	161.11	0.54	0.68	0.42	47%
	Ours	**9.23**	**61.34**	**0.79**	**0.80**	**0.60**	**69%**
Fashion-MNIST	Real	8.98	1.49	0.88	0.91	0.79	100%
	G-PATE	3.35	205.78	0.30	0.50	0.40	54%
	DP-SGD GAN	3.55	243.80	0.50	0.46	0.43	53%
	DP-Merf	2.32	267.78	0.56	0.62	0.51	65%
	DP-Merf AE	3.68	213.59	0.56	0.62	0.45	55%
	Ours	**5.32**	**131.34**	**0.65**	**0.65**	**0.53**	**67%**

Table 1: Quantitative Results on MNIST and Fashion-MNIST ($\varepsilon = 10, \delta = 10^{-5}$)

- **Decentralized (Federated) setting**

  Better *sample quality*:
  - 0.28x smaller FID

  Lower *privacy cost*:
  - 10^4× smaller epsilon

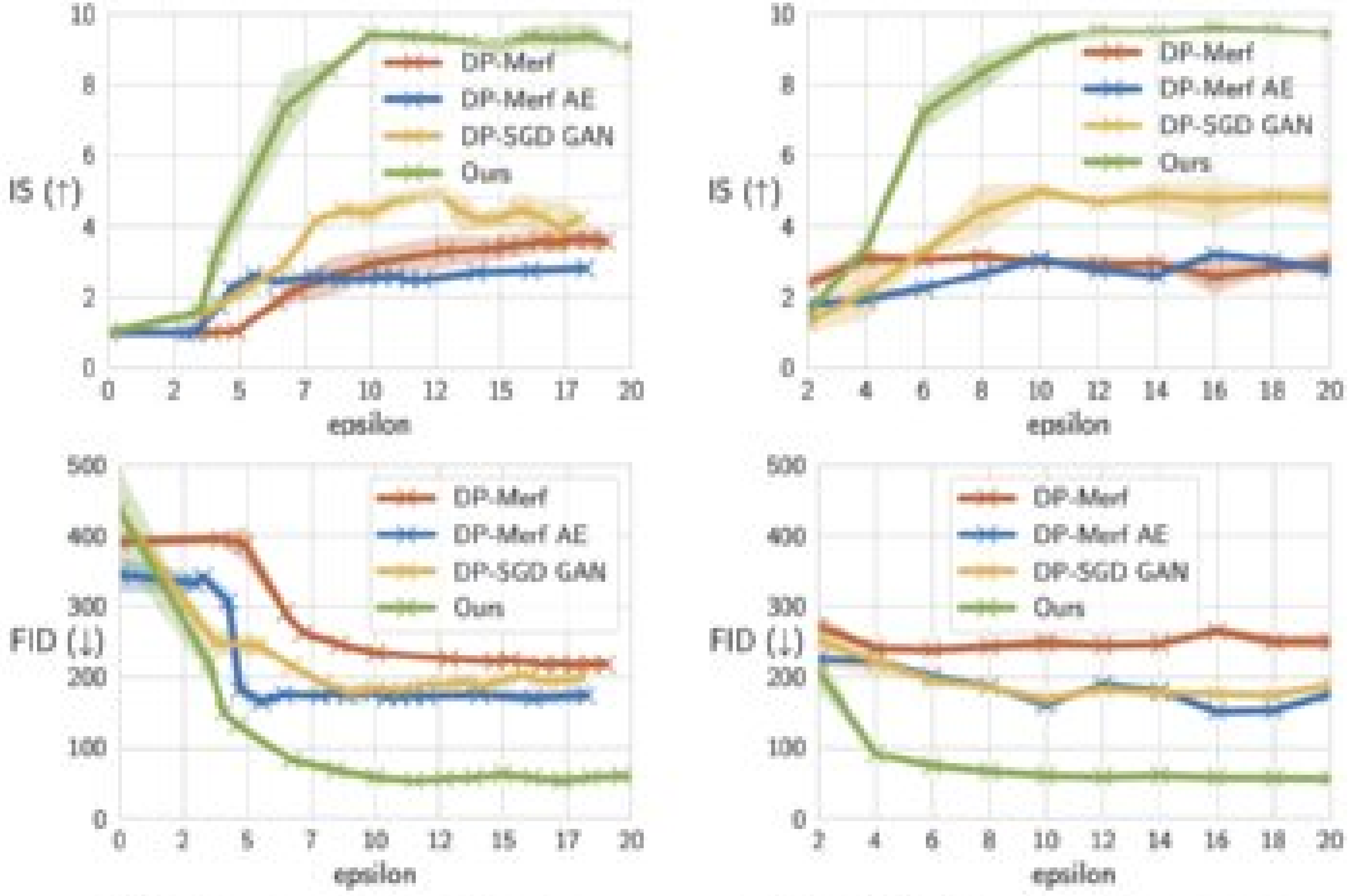
	IS ↑	FID ↓	epsilon ↓	CT (byte) ↓
Fed Avg GAN	10.88	218.24	9.99×10^6	$\sim 3.94 \times 10^7$
Ours	**11.25**	**60.76**	$\mathbf{5.99 \times 10^2}$	$\sim \mathbf{1.50 \times 10^5}$

Table 4: Quantitative Results on Federated EMNIST ($\delta = 1.15 \times 10^{-3}$)

Consistent improvement over baselines across different datasets, settings and metrics

Results

- **Privacy-utility curve** (Sample utility at different privacy level ε)



(b) Effects of Iterations (c) Effects of Noise scale

Results

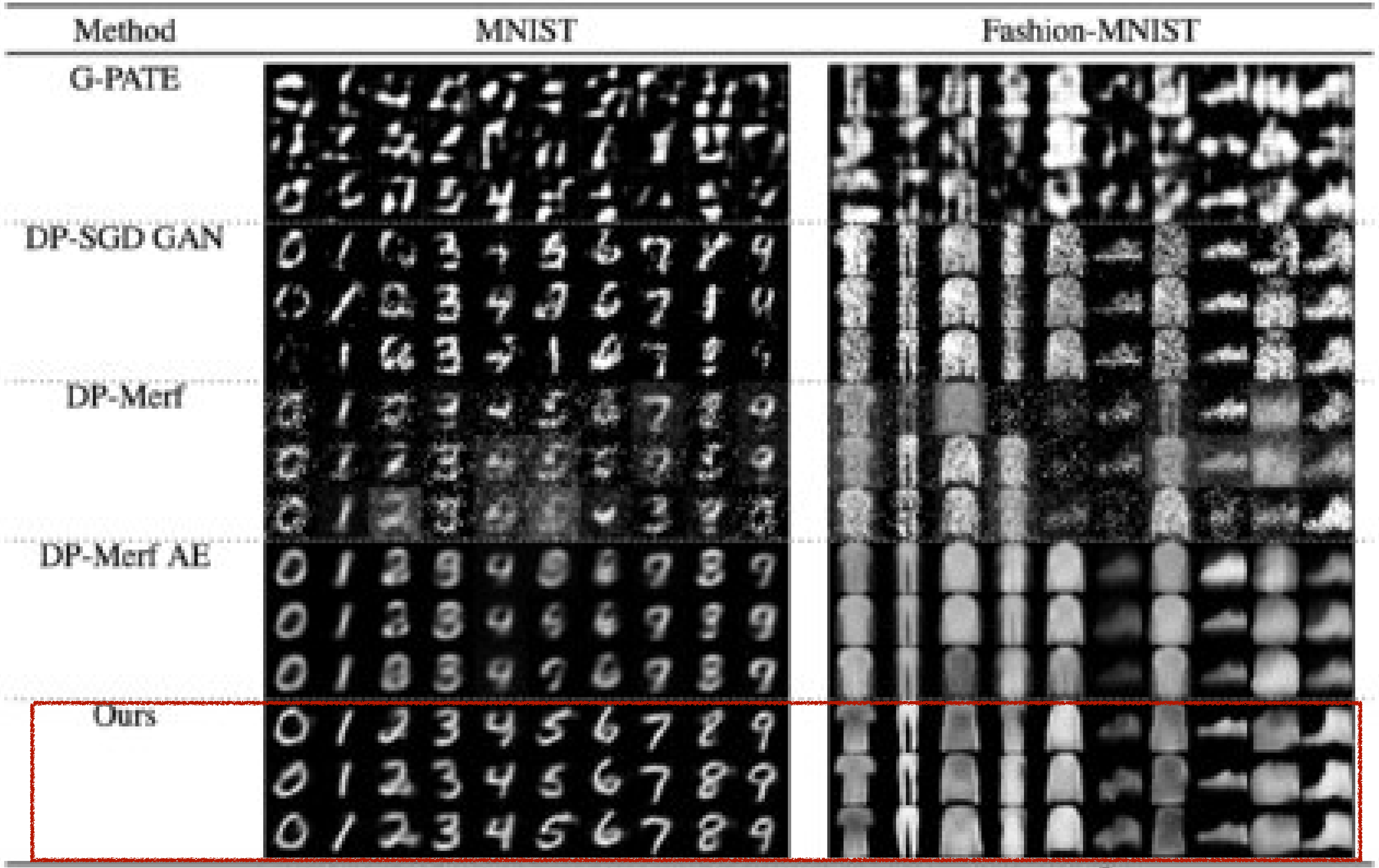


Figure 3: Generated samples with $(\varepsilon, \delta) = (10, 10^{-5})$

More details in the paper

GS-WGAN: A Gradient-Sanitized Approach for Learning Differentially Private Generators

Dingfan Chen[1] Tribhuvanesh Orekondy[2] Mario Fritz[1]

Code and Models are available on Github

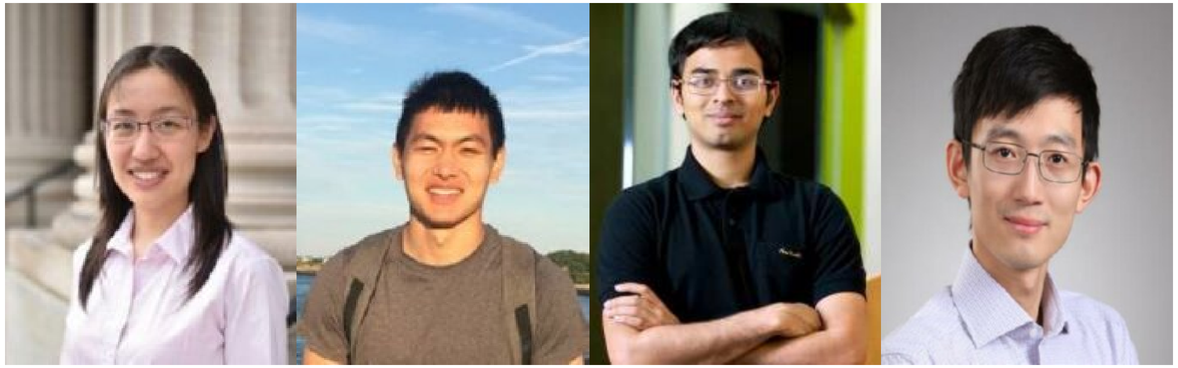
https://github.com/DingfanChen/GS-WGAN

[1] CISPA Helmholtz Center for Information Security
[2] Max Planck Institute for Informatics

Self-training Avoids Using Spurious Features Under Domain Shift
(自学习在数据分布变化时避免使用伪特征)

Yining Chen*, Colin Wei*, Ananya Kumar, Tengyu Ma (Stanford)

*equal contribution



训练和测试数据分布不同时，模型正确率下降

Unsupervised domain adaptation
无监督域适应

Labeled source distribution $\mathcal{D}_S$

Unlabeled target distribution $\mathcal{D}_T$

MNIST → SVHN

- Goal: maximize the test accuracy on the target

现有文献大多假设源域和目标域接近

- Existing theory for unsupervised domain adaptation: source and target are close [Ben-David et al. 10’, Sugiyama et al., 07’]
- Realistic domain shifts are large
  - e.g. MNIST -> SVHN
- Self-training algorithms （自学习算法）work under large domain shifts:
  - **Pseudo-labeling（伪标记）** [Lee 13’]
  - **Conditional entropy minimization （熵减）** [Grandvalet & Bengio, 05’]

域适应理论的主要难点

- Realistic assumption on the relation between $\mathcal{D}_S$ and $\mathcal{D}_T$?

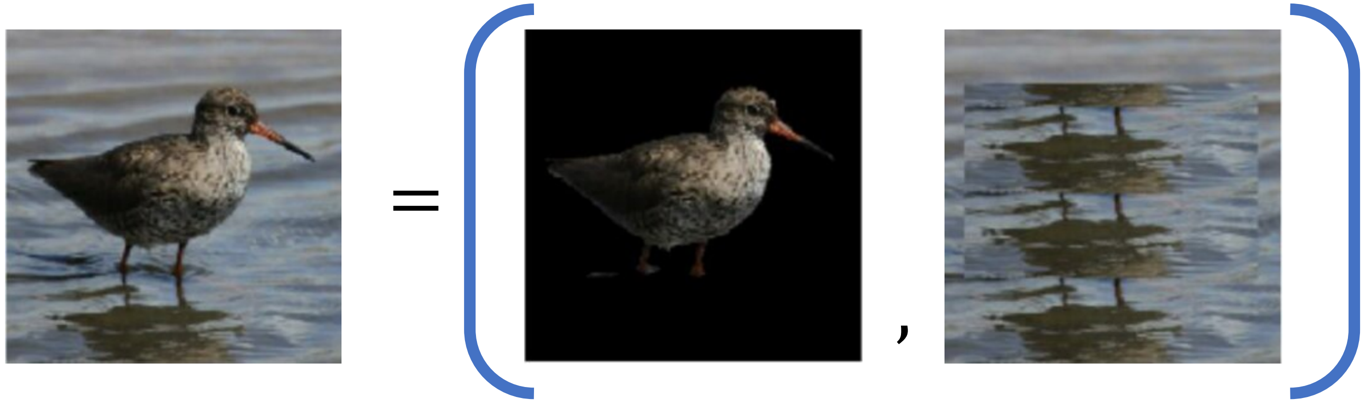
- Our work:
  - Assumption: the target is more diverse than the source
  - Self-training **provably** works

Domain shift assumption: target is more diverse
假设： 目标域更多样化

Input $x = (x_1, x_2)$

signal features
determine y in both source and target

spurious features
- correlate with y in source
- independent of y in target



我们分析的算法 (线性模型: $\hat{y} = w^\top x$)

0. Learn a classifier w_s using the source labeled data

Pseudo-labeling

1. Label $x^i \in \mathrm{D_T}$ by

$$y_{ps}^i = {w_s}^\top x^i$$

2. Train on $\left\{\left(x^i, y_{ps}^i\right)\right\}$

Entropy minimization

1. Minimize

$$H(\mathrm{Y}|\mathrm{X}) \approx \sum_{x^i \in \mathrm{D_T}} \ell_{exp}\left(\left|w^\top x^i\right|\right)$$

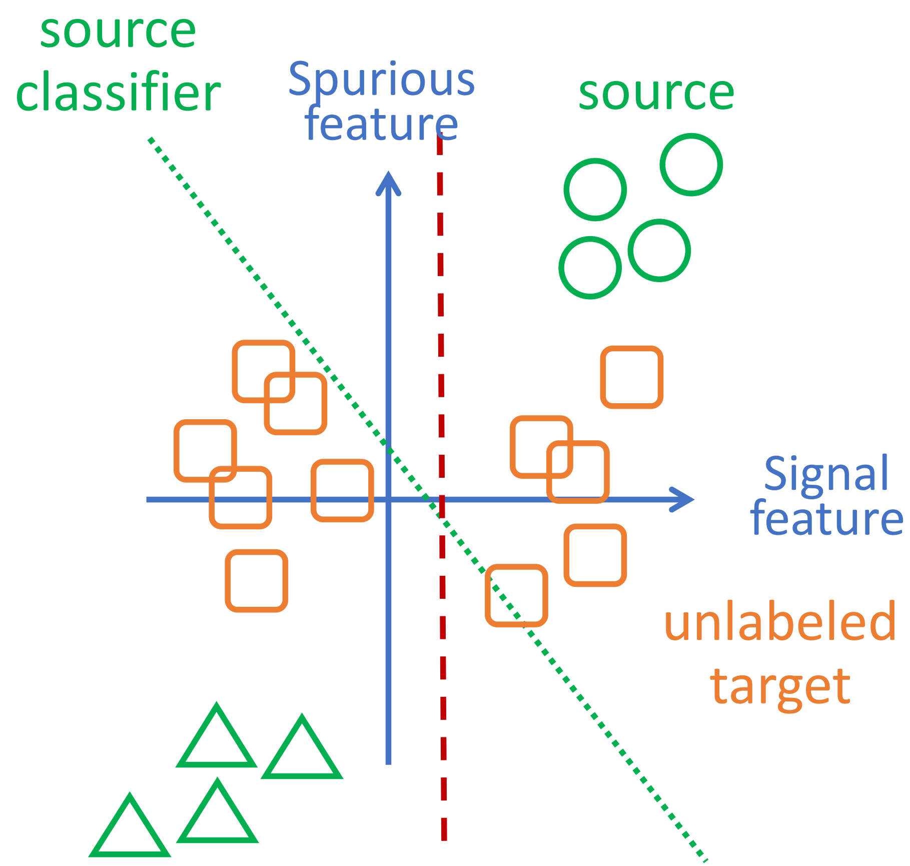
核心结论

Assume:

- Signal $x_1 \sim$ mixture of log-concave distributions
- Spurious x_2 is Gaussian

- Starting with a decent source classifier, self-training on polynomial # of unlabeled target examples converges to a solution that does not use x_2.

When x_1 is mixture of Gaussians:



证明的思路

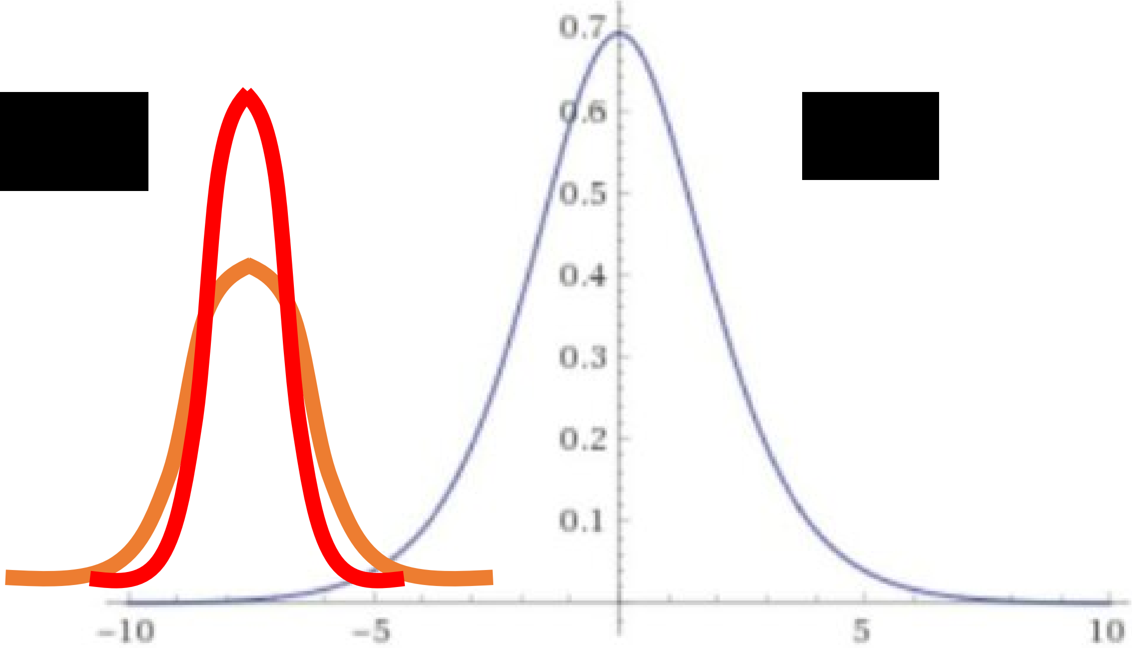
[redacted]

- Fix x_1: [redacted] is Gaussian!
  - Mean [redacted]
  - Variance [redacted]

[redacted]

- When [redacted]:

[redacted]



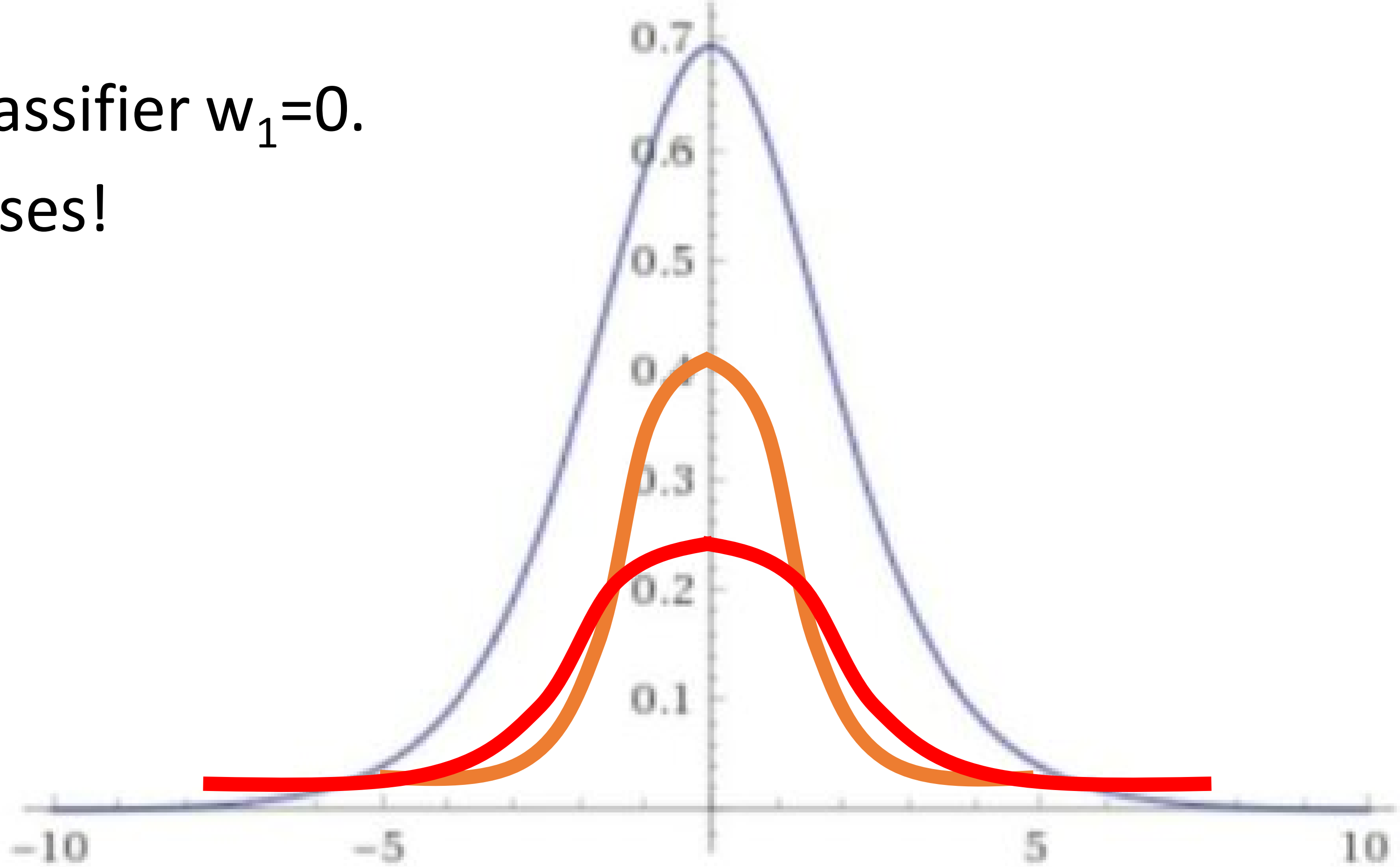
[redacted]

[redacted]

- One step of GD on L(w) decreases norm of w_2.

自学习失败例子1: Bad source classifier

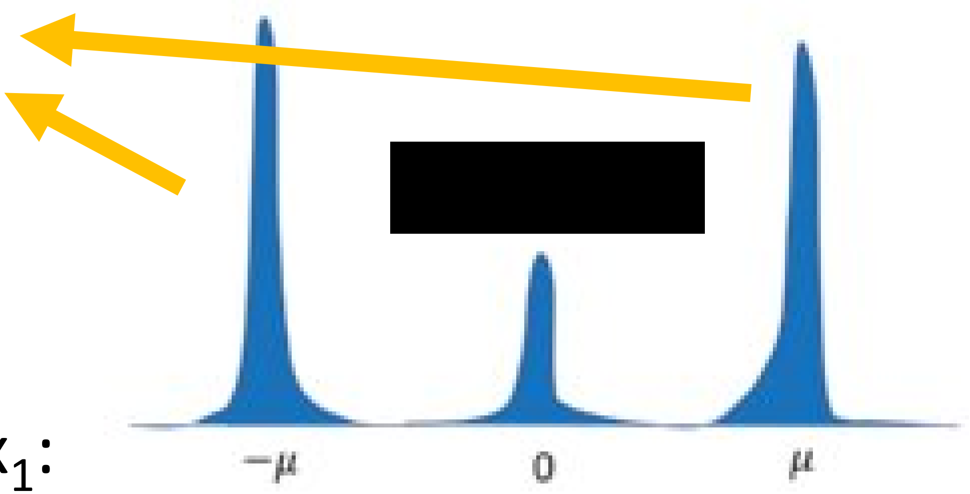
- Source classifier w_1=0.
- w_2 increases!



自学习失败例子2: Isolated clusters

- Source classifier is good, but w_2 still increases!

[redacted]

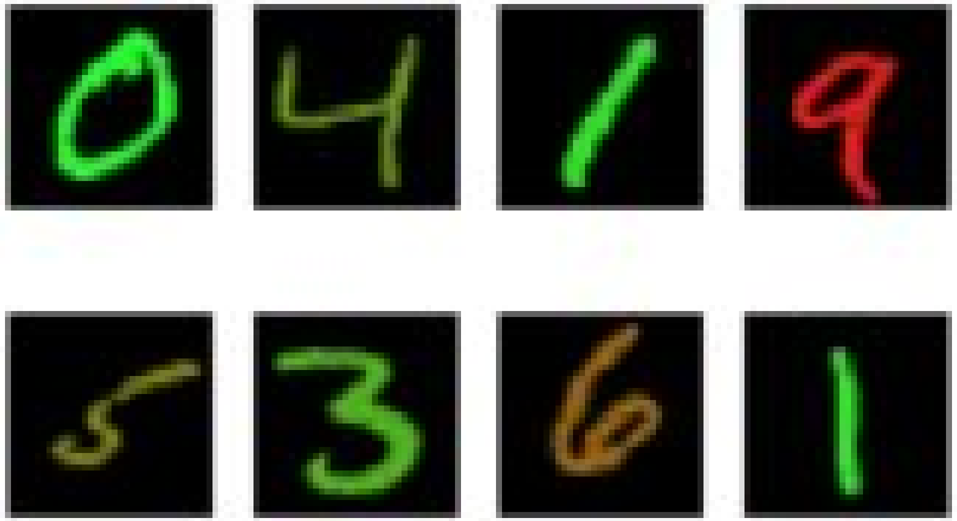


- Assumption on x_1:
  - Sliced log-concave: Each component is unimodal, not too wide.
  - Sliced log-smooth: Not too narrow.
  - Well-separated: Means far from 0.

实验1: Colored MNIST

- Signal (x_1) = Shape, Spurious feature (x_2) = Color, Target (y) = Digit

Source Domain



Target Domain

实验2: Synthetic CelebA

- Spurious feature (x_2) = Blondness, Target (y) = Gender

Source Domain

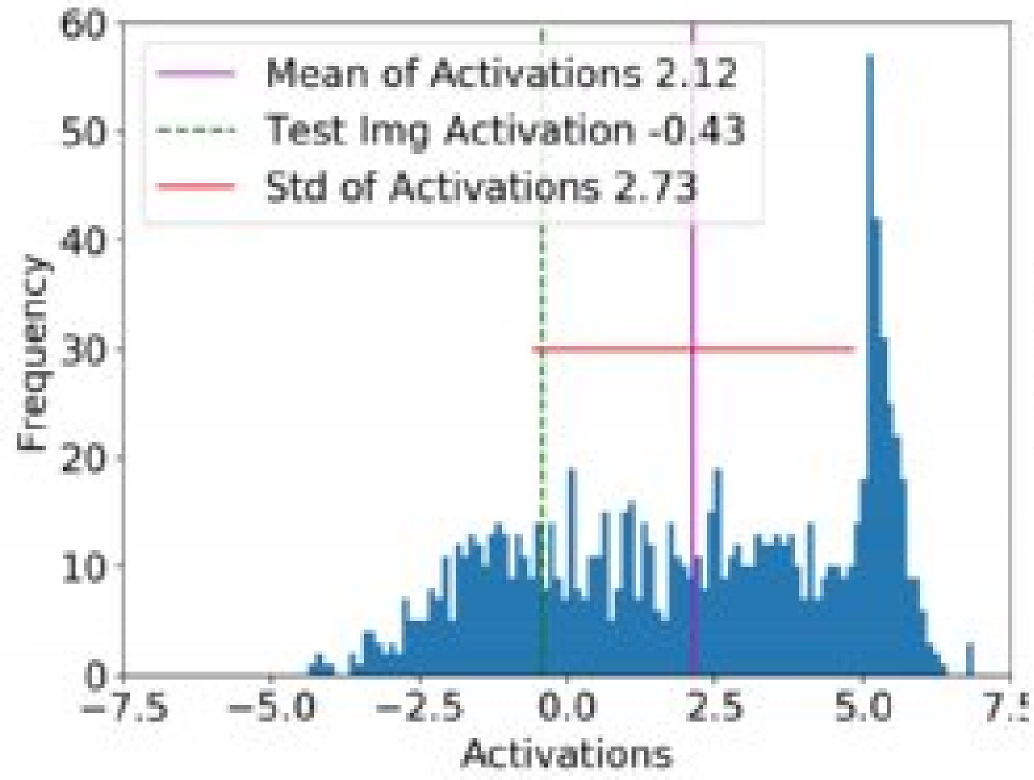
Target Domain

自学习提高目标域正确率

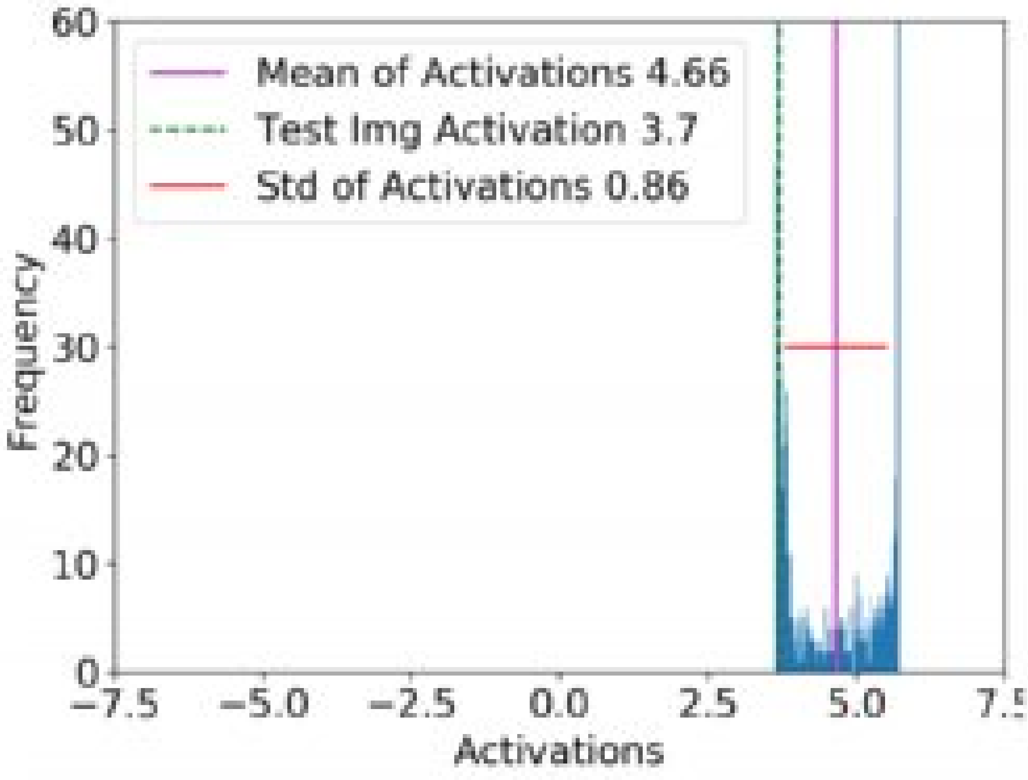
	CelebA	CMNIST10 ($p = 0.95$)	CMNIST2	CMNIST10 ($p = 0.97$)
Trained on source	81%	82%	94%	72%
After self-training	88%	91%	96%	67%

…only if the source classifier is decent

自学习减少使用伪特征



Mean of Activations 2.12
Test Img Activation -0.43
Std of Activations 2.73
Frequency
Activations



Mean of Activations 4.66
Test Img Activation 3.7
Std of Activations 0.86
Frequency
Activations

自学习减少使用伪特征

Source Domain

Target Domain

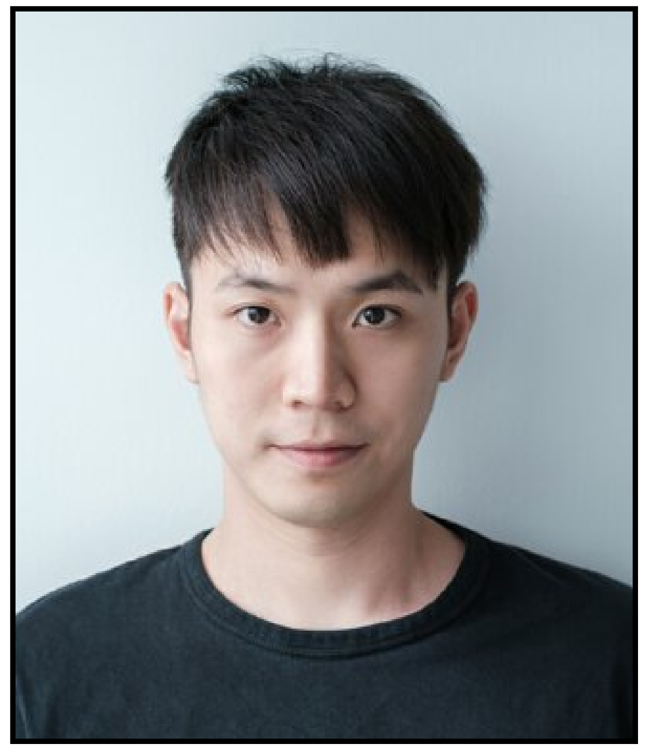
自学习更正的例子

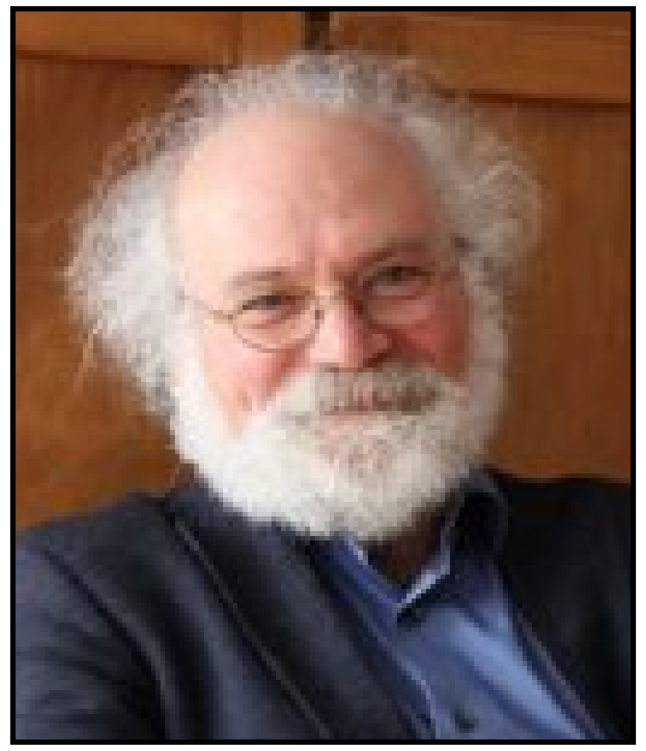
总结

谢谢大家!

- When source has spurious correlations, but the target doesn't, self-training exploits unlabeled target data to avoid relying on spurious correlations.
- Conditions for success: separation between classes, decently accurate source classifier.
- Consistent with the recent large-scale semi-supervised learning experiments, e.g. [Xie et al., 20']
  - Self-train on diverse, unlabeled data pool improves robustness.

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)

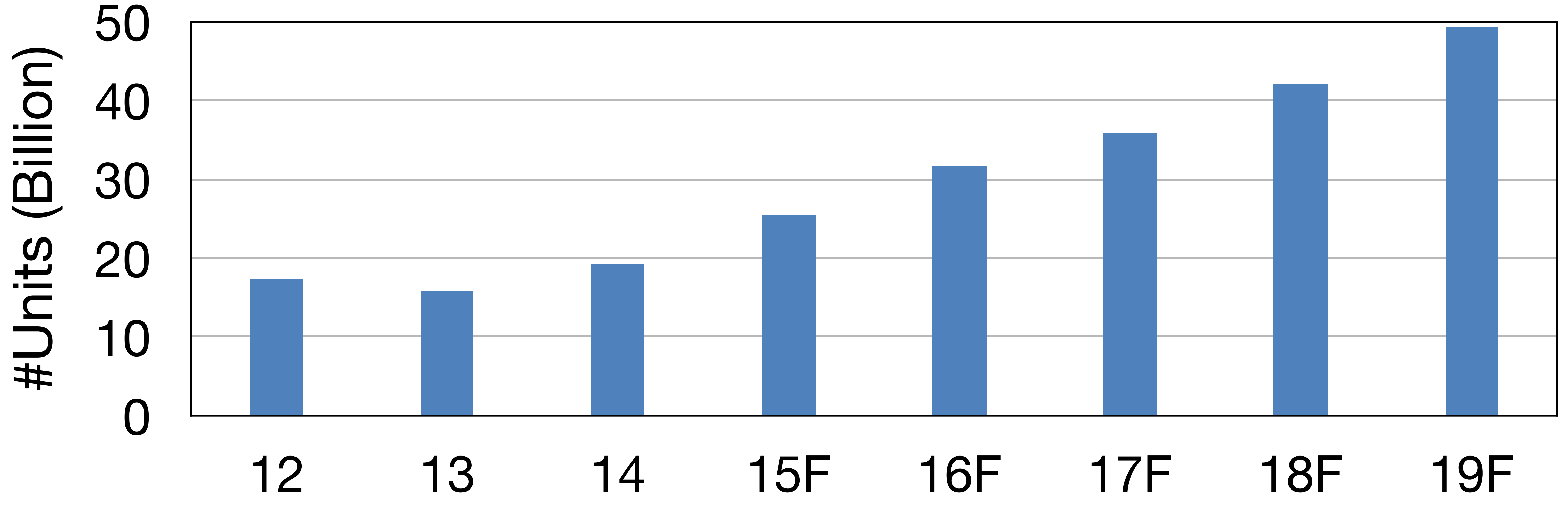
Background: The Era of AIoT on Microcontrollers (MCUs)

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

Background: The Era of AIoT on Microcontrollers (MCUs)

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

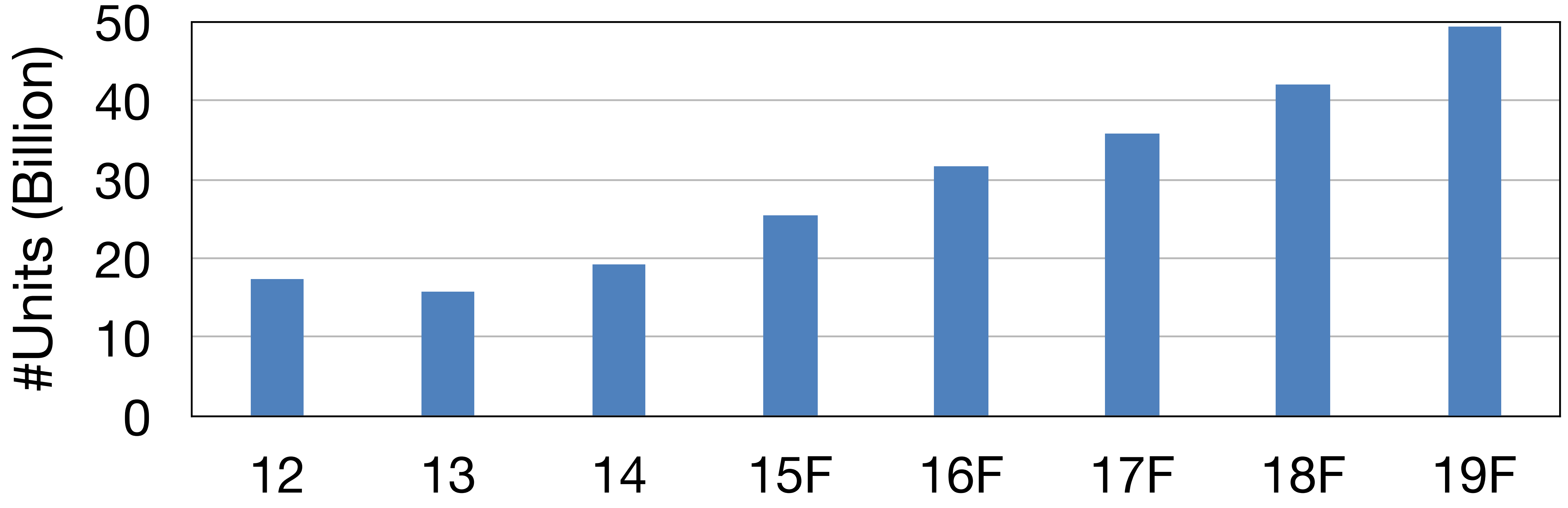
- **Rapid growth**



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

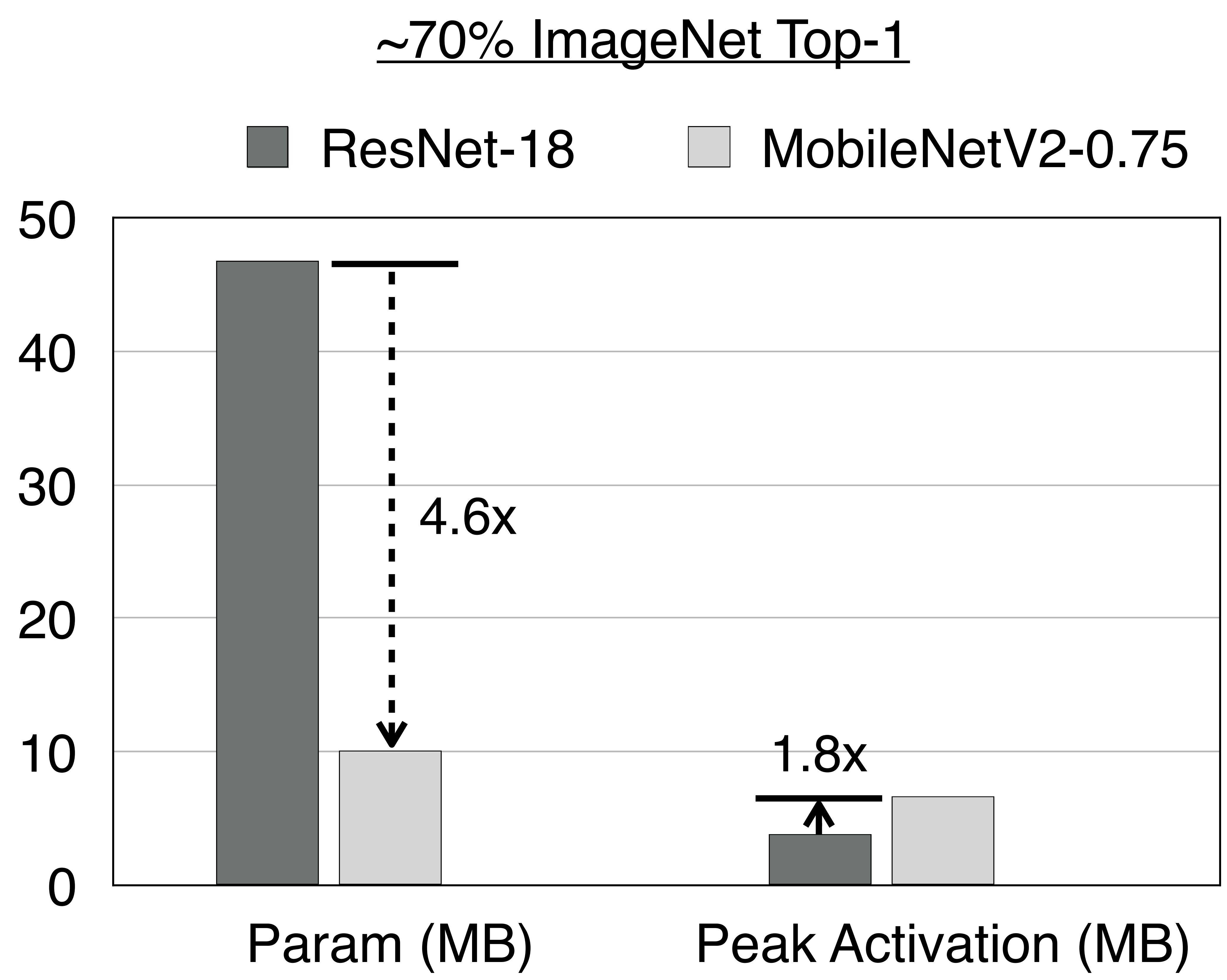
	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	16GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

13,000x smaller

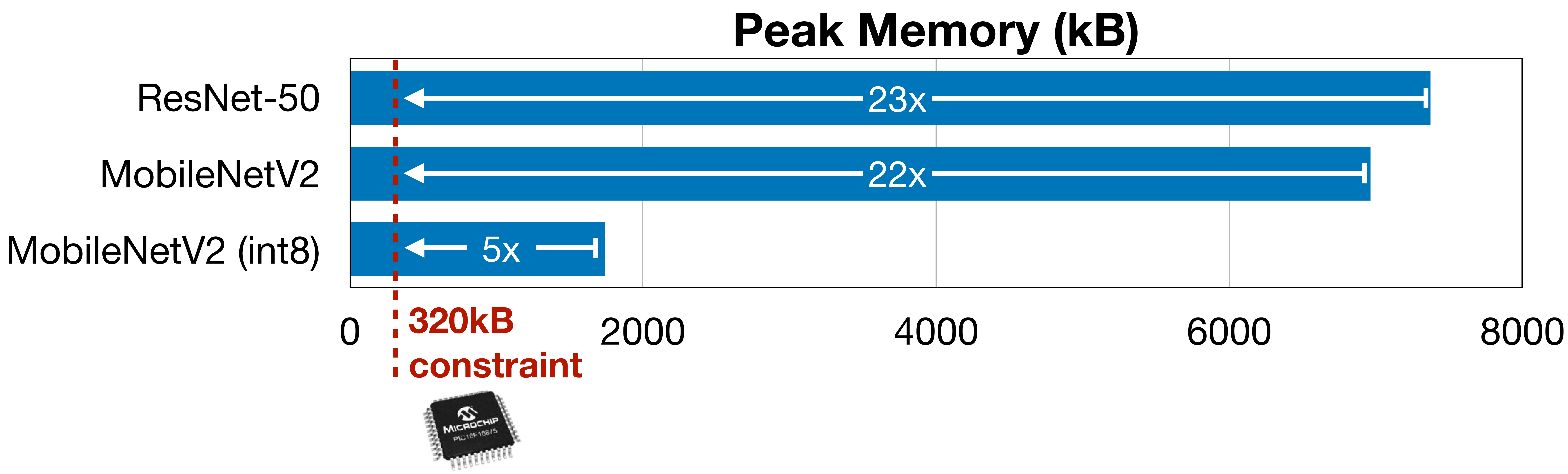
50,000x smaller

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

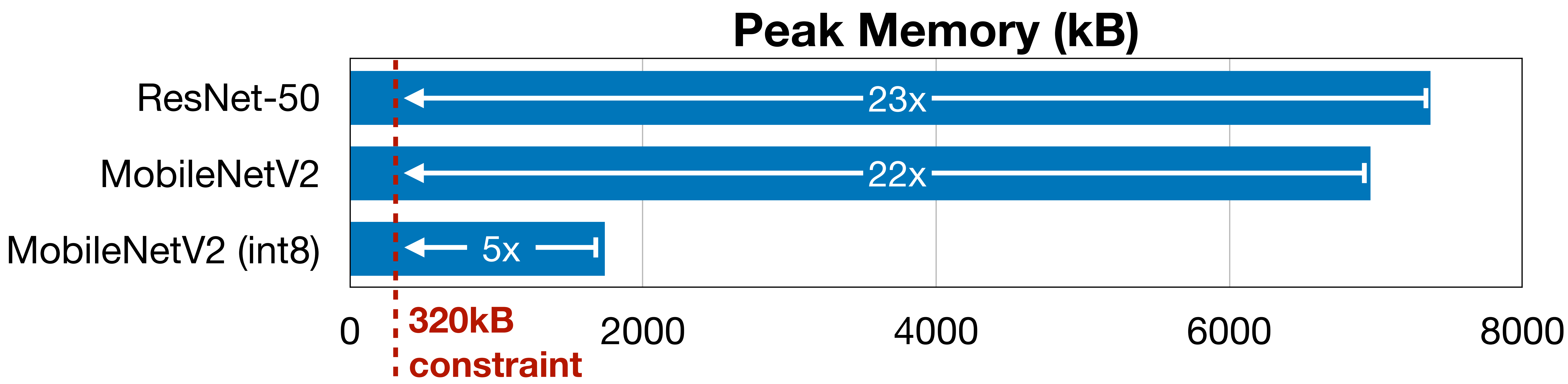
Existing efficient network only reduces model size but NOT activation size!

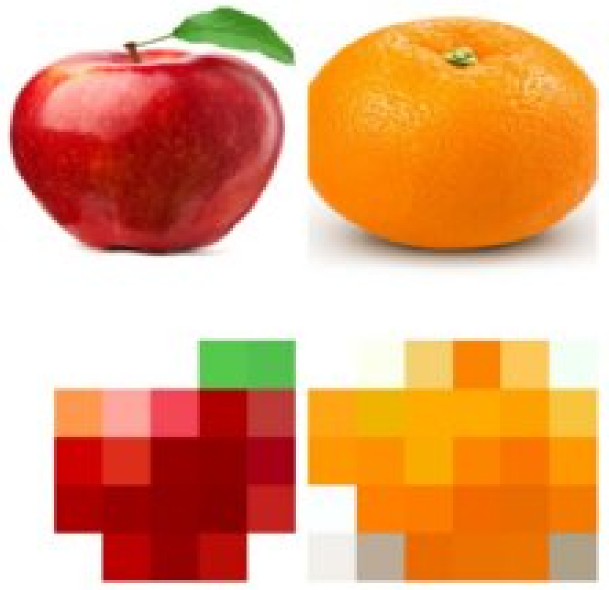


Challenge: Memory Too Small to Hold DNN

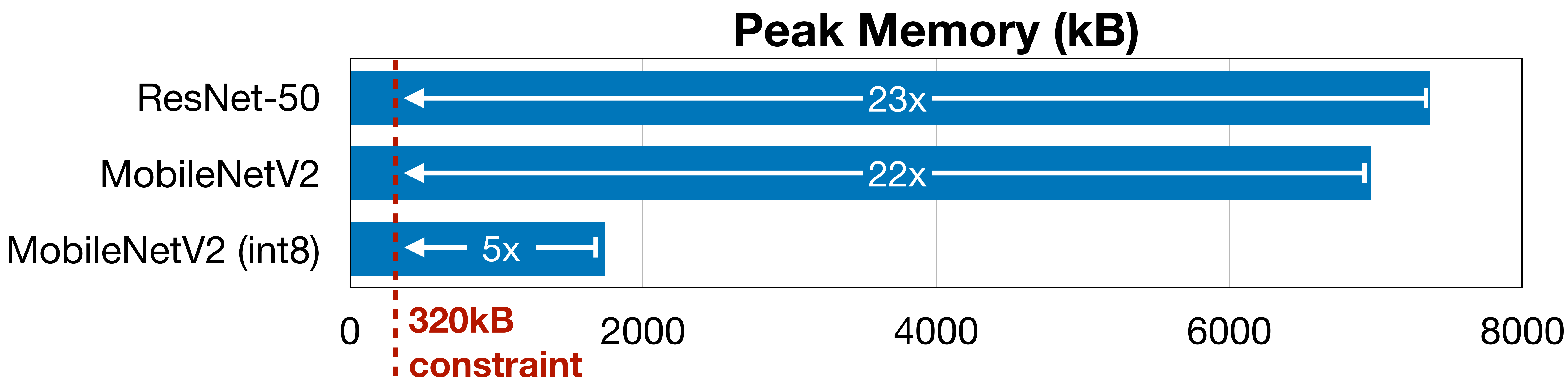


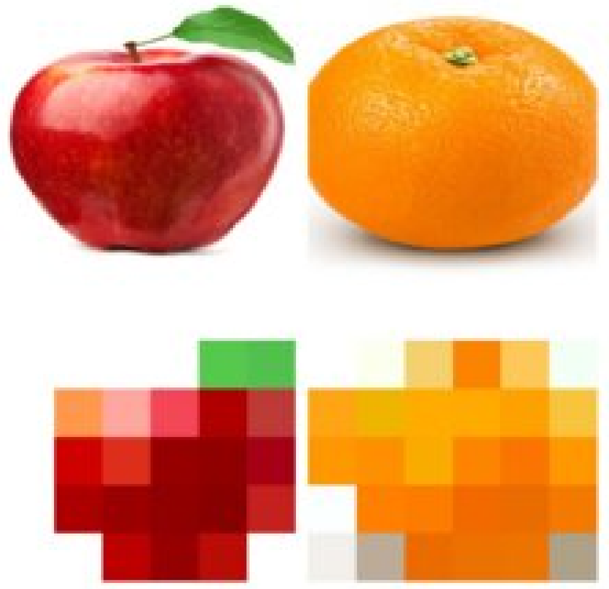
Challenge: Memory Too Small to Hold DNN





Challenge: Memory Too Small to Hold DNN

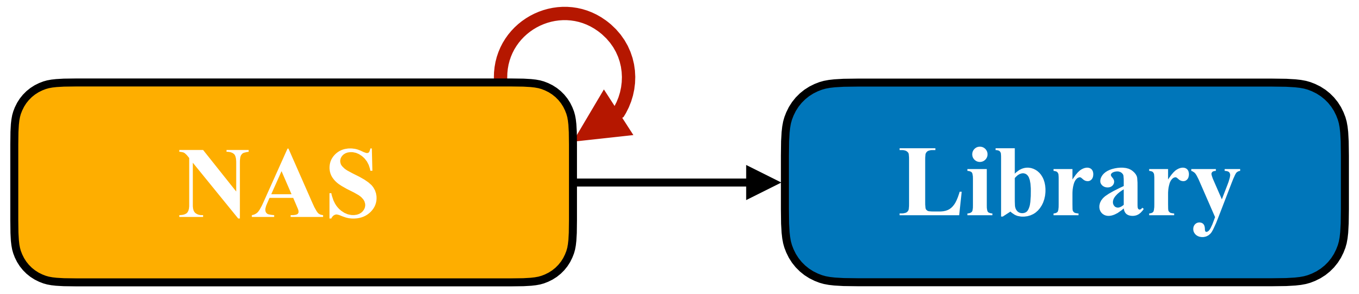




MCUNet

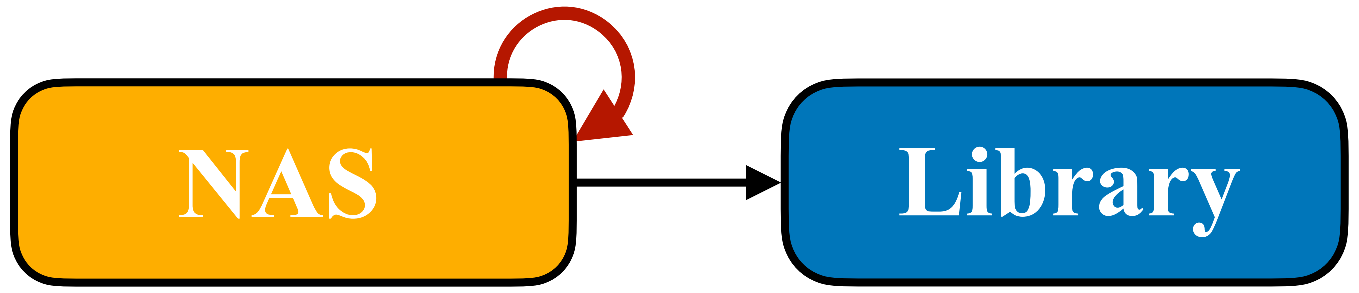
MCUNet: System-Algorithm Co-design

MCUNet: System-Algorithm Co-design

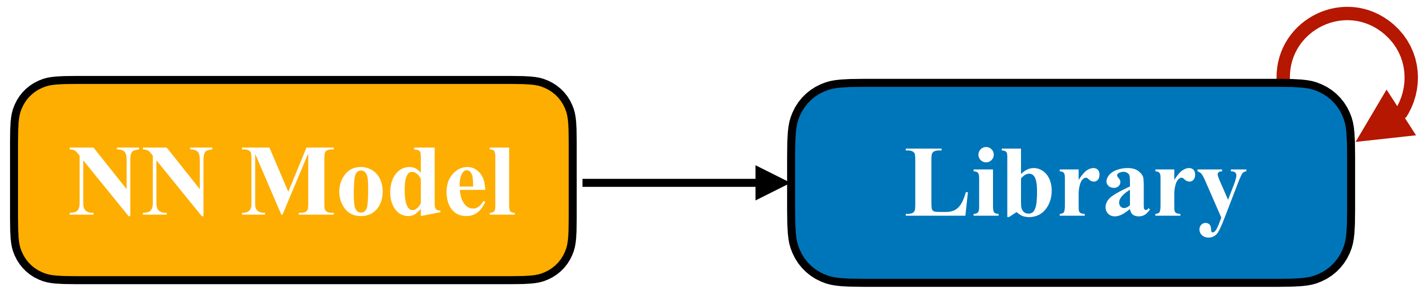


(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*

MCUNet: System-Algorithm Co-design

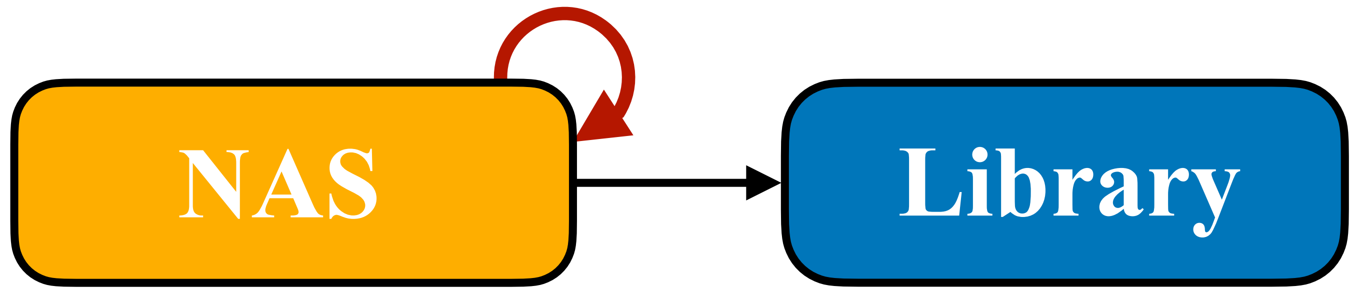


(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*

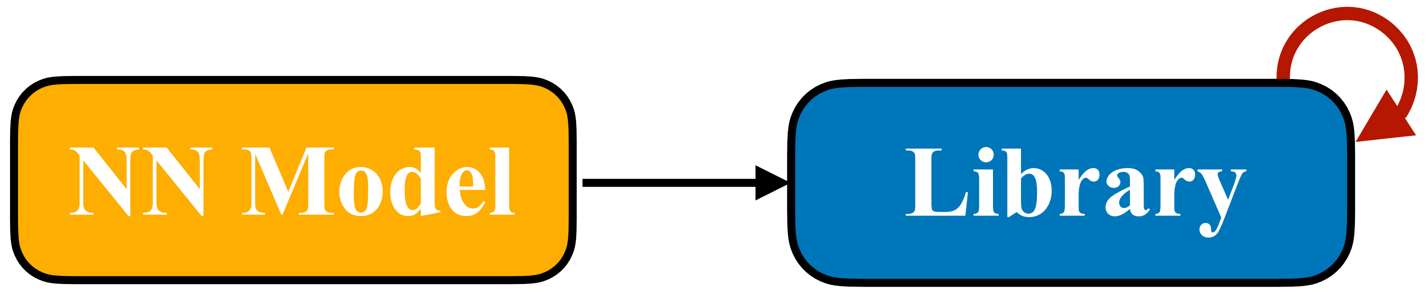


(b) Tune deep learning library given a NN model
e.g., *TVM*

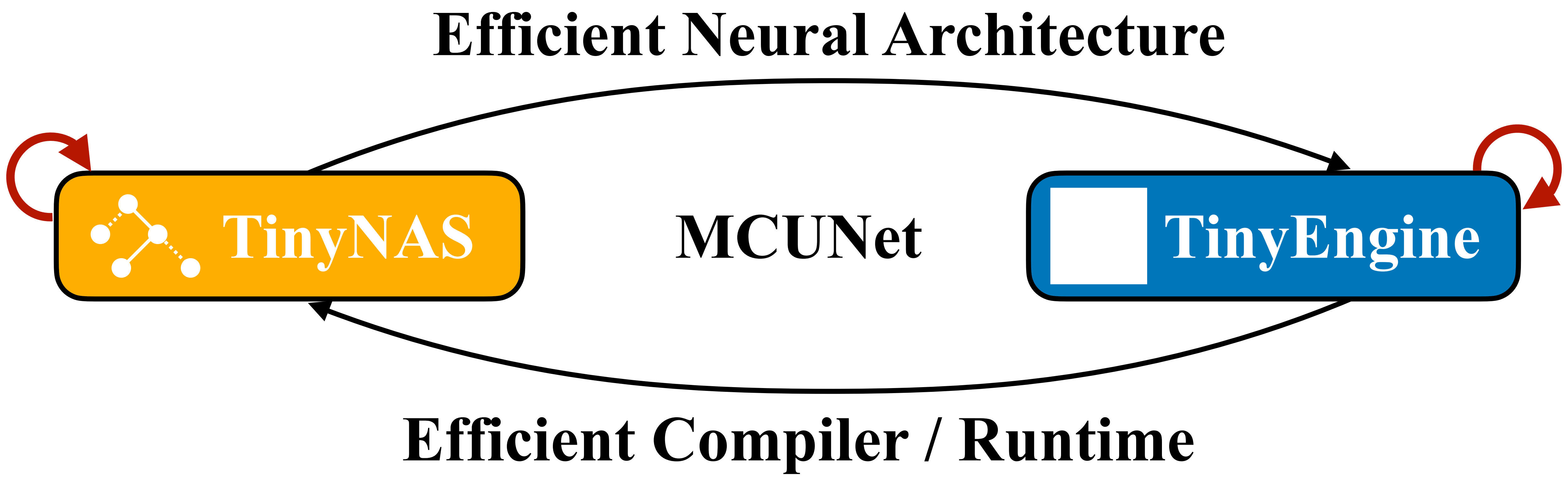
MCUNet: System-Algorithm Co-design



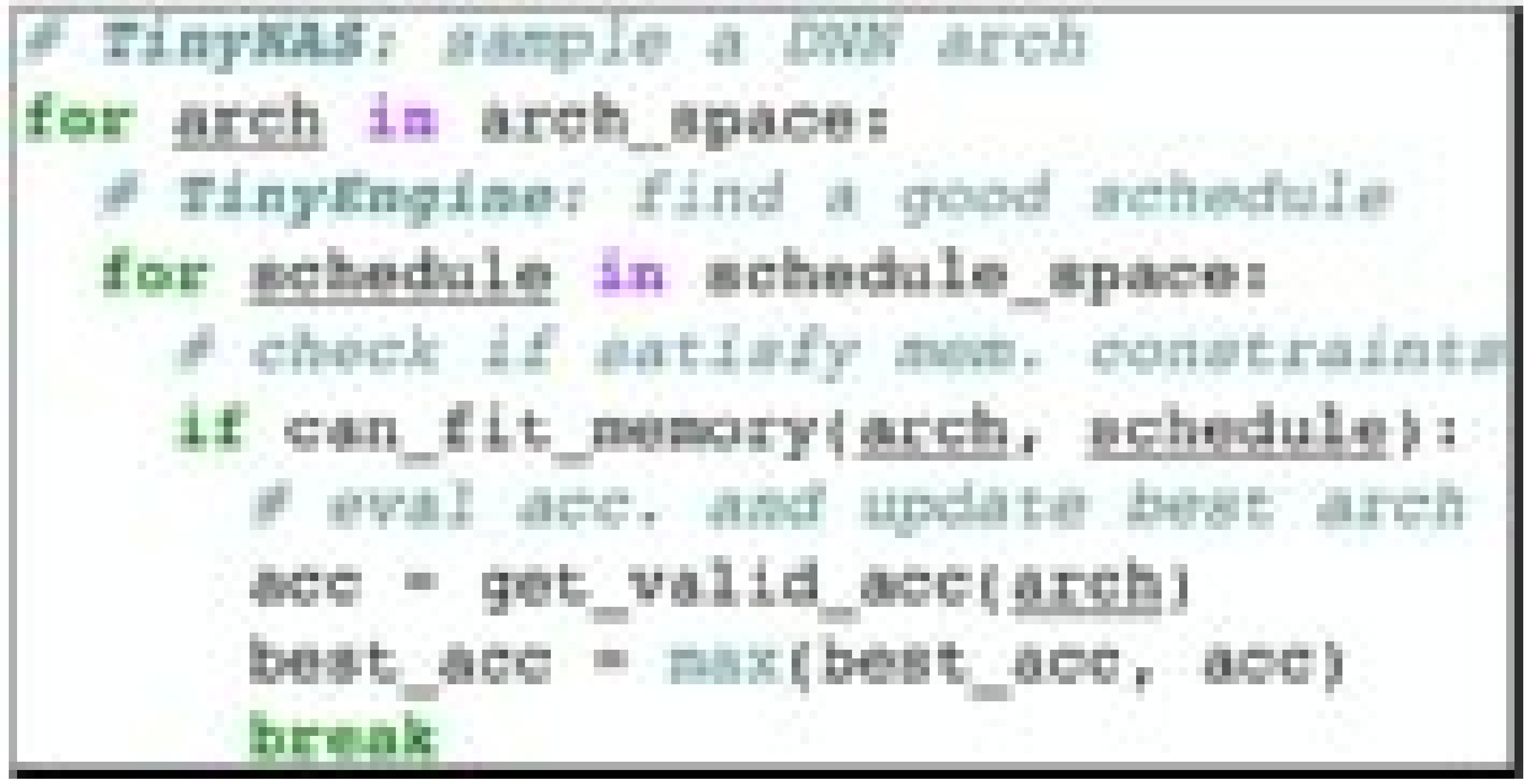
(a) Search NN model on an existing library
e.g., *ProxylessNAS, MnasNet*



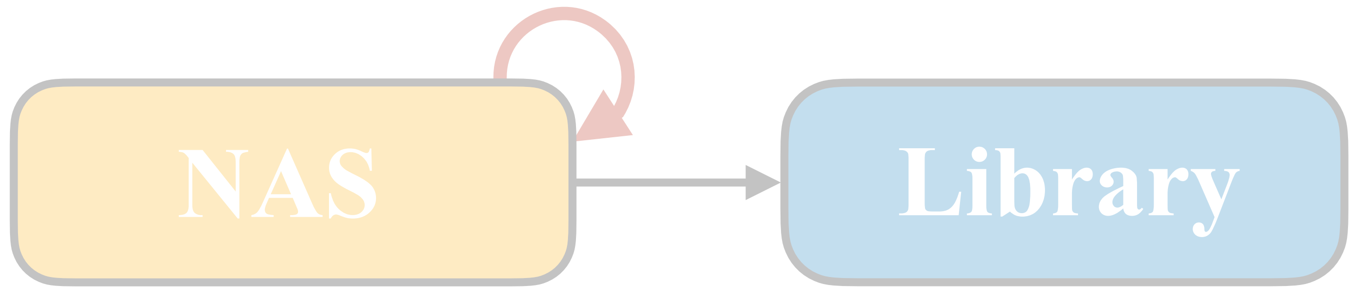
(b) Tune deep learning library given a NN model
e.g., *TVM*



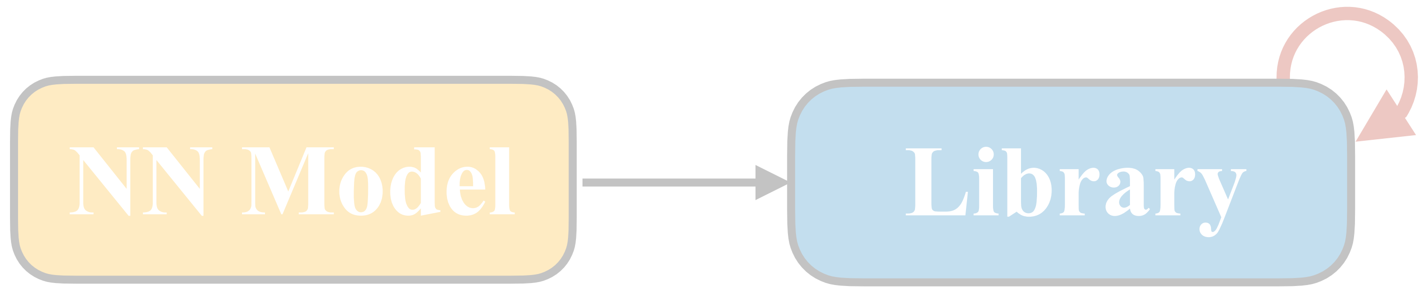
(c) *MCUNet*: system-algorithm co-design



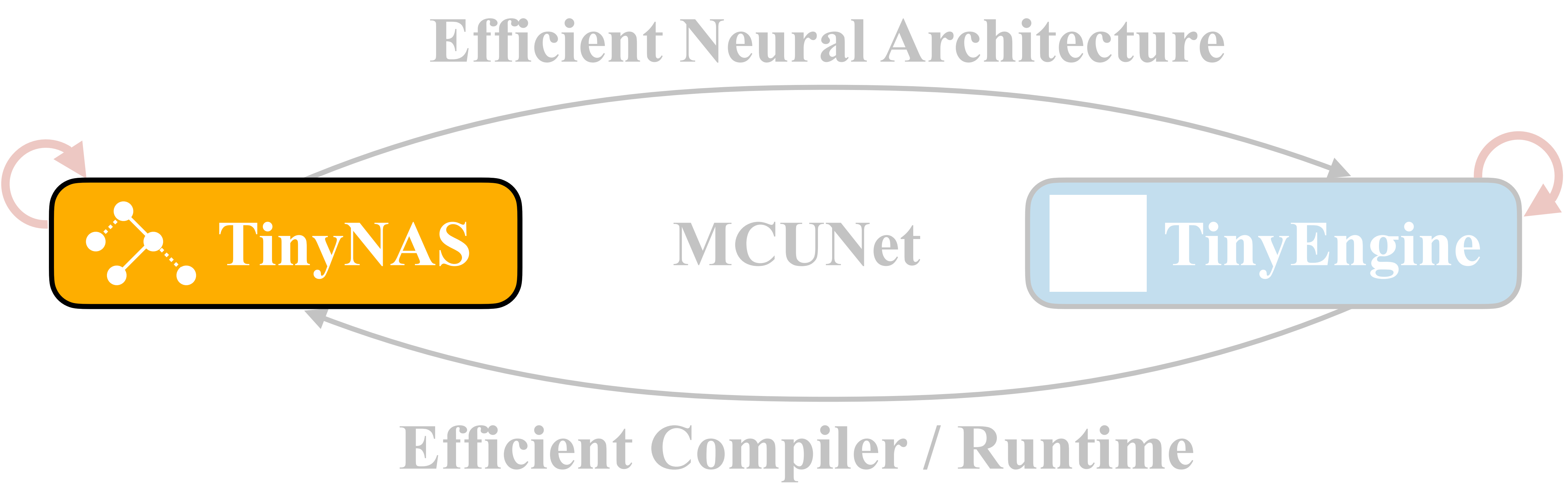
MCUNet: System-Algorithm Co-design



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



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



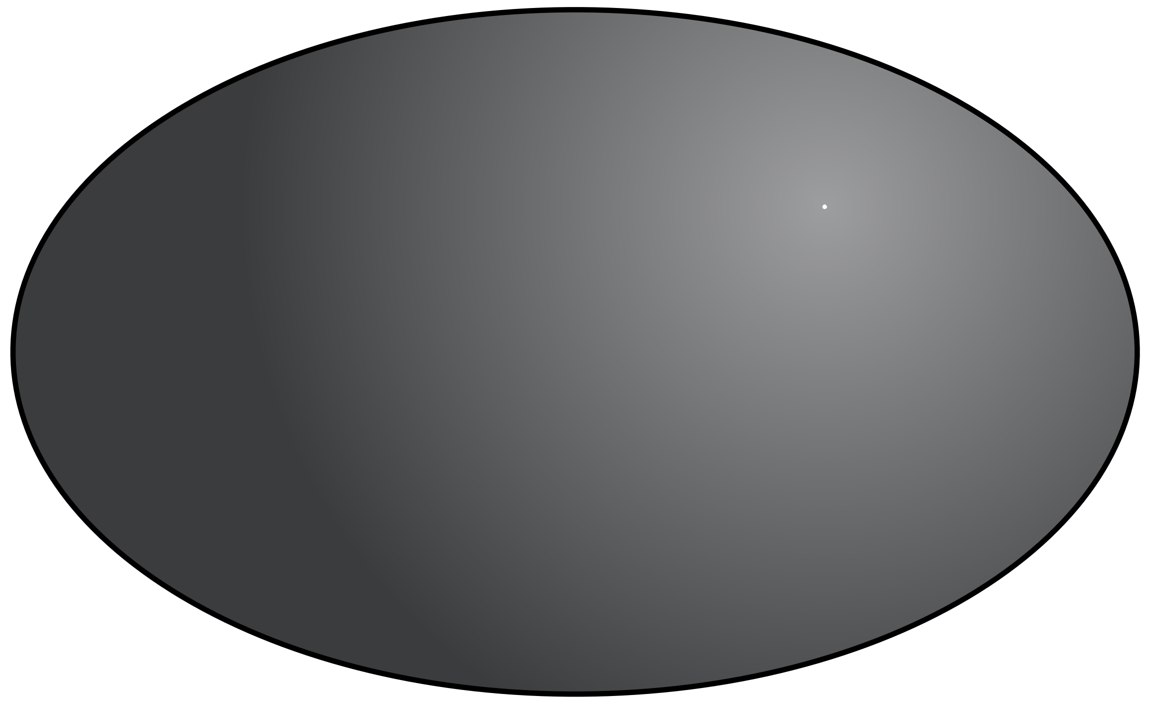
(c) *MCUNet*: system-algorithm co-design

```
# 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

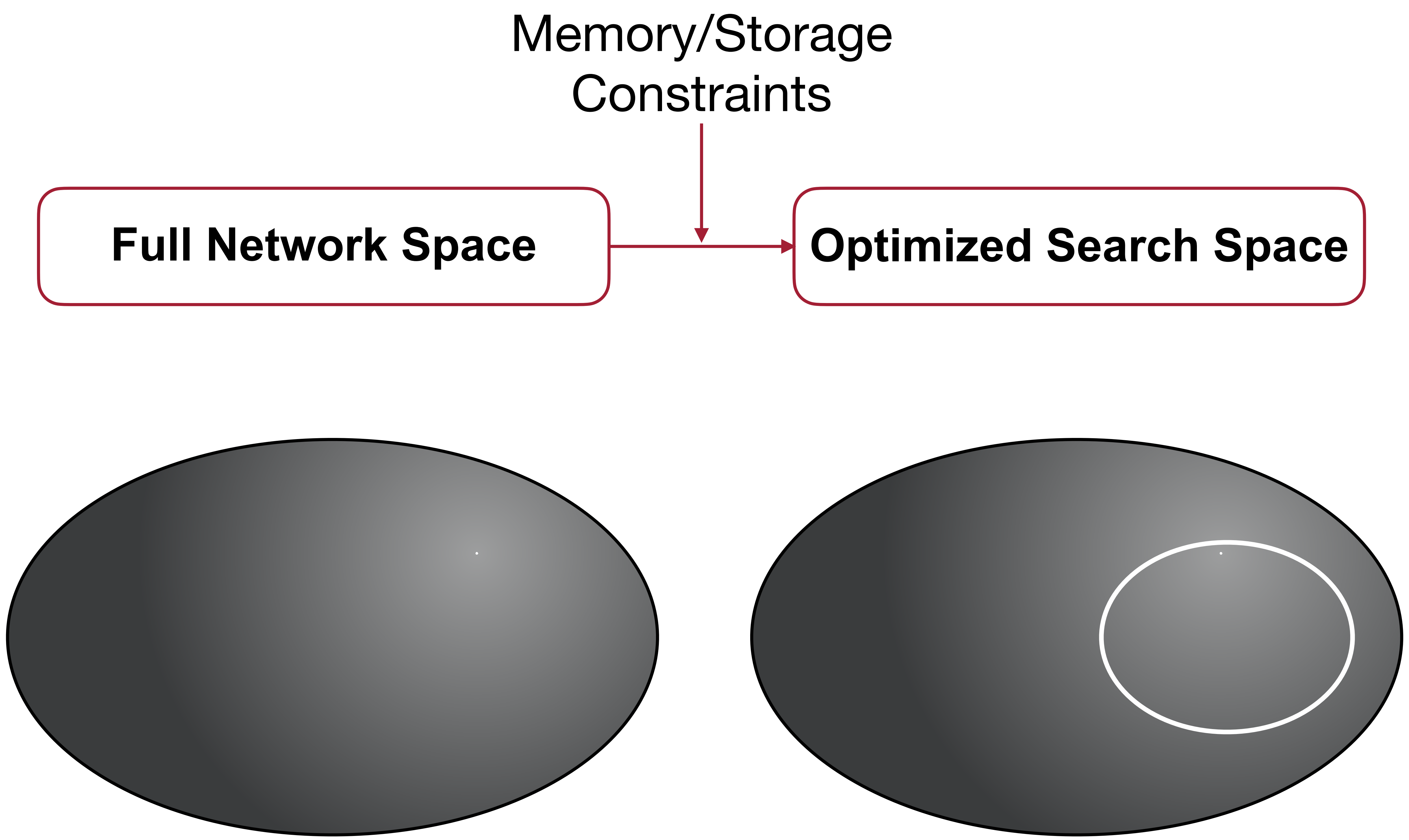
Search space design is crucial for NAS performance
There is no prior expertise on MCU model design

Full Network Space



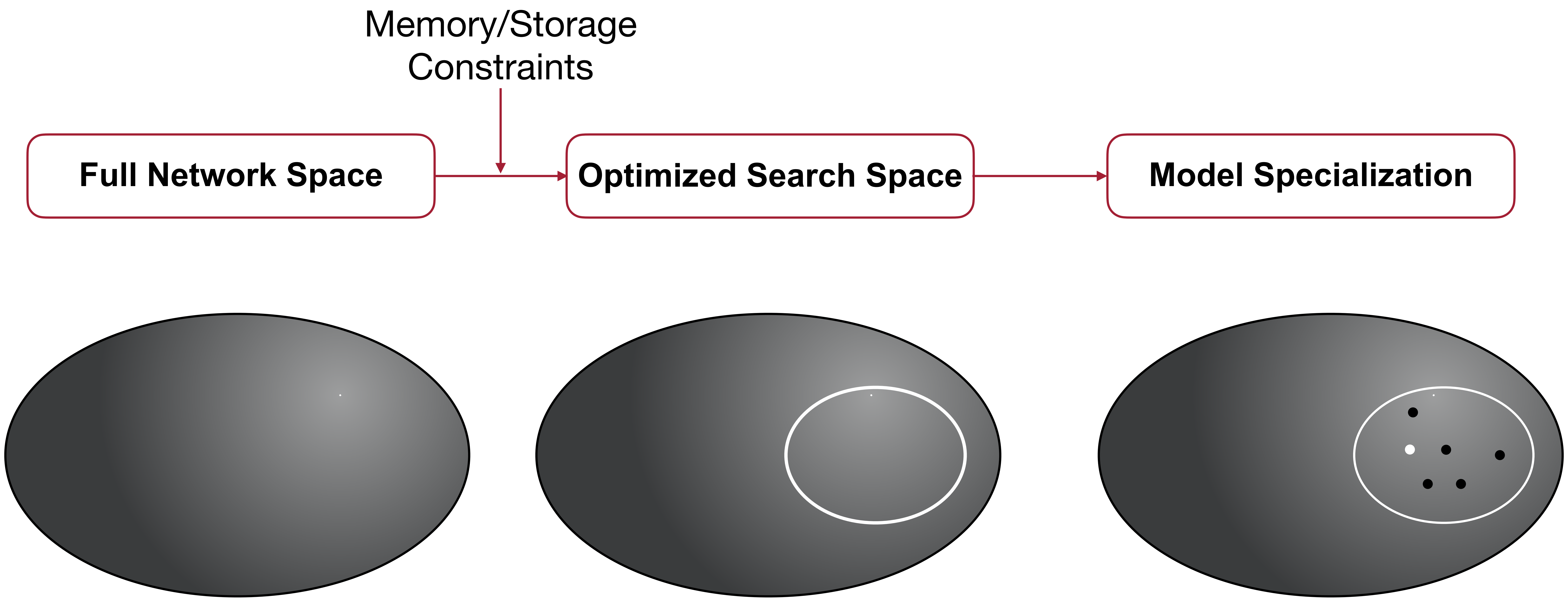
TinyNAS: Two-Stage NAS for Tiny Memory Constraints

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

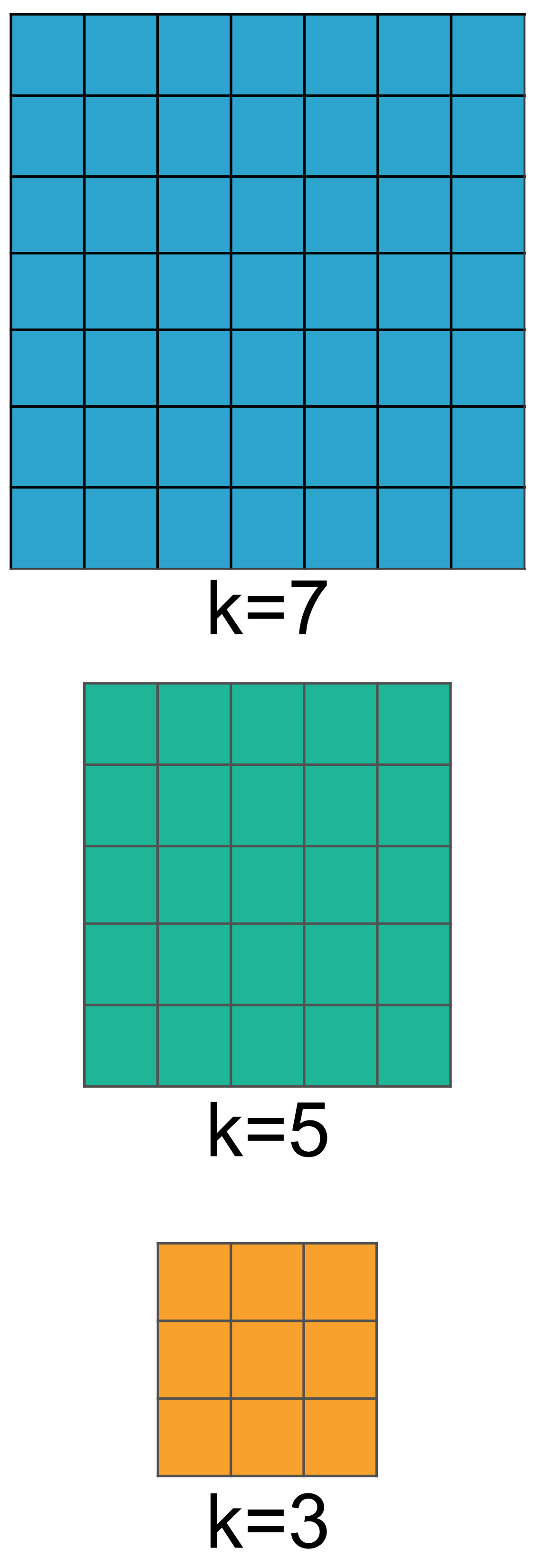
Revisit ProxylessNAS search space:

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

TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

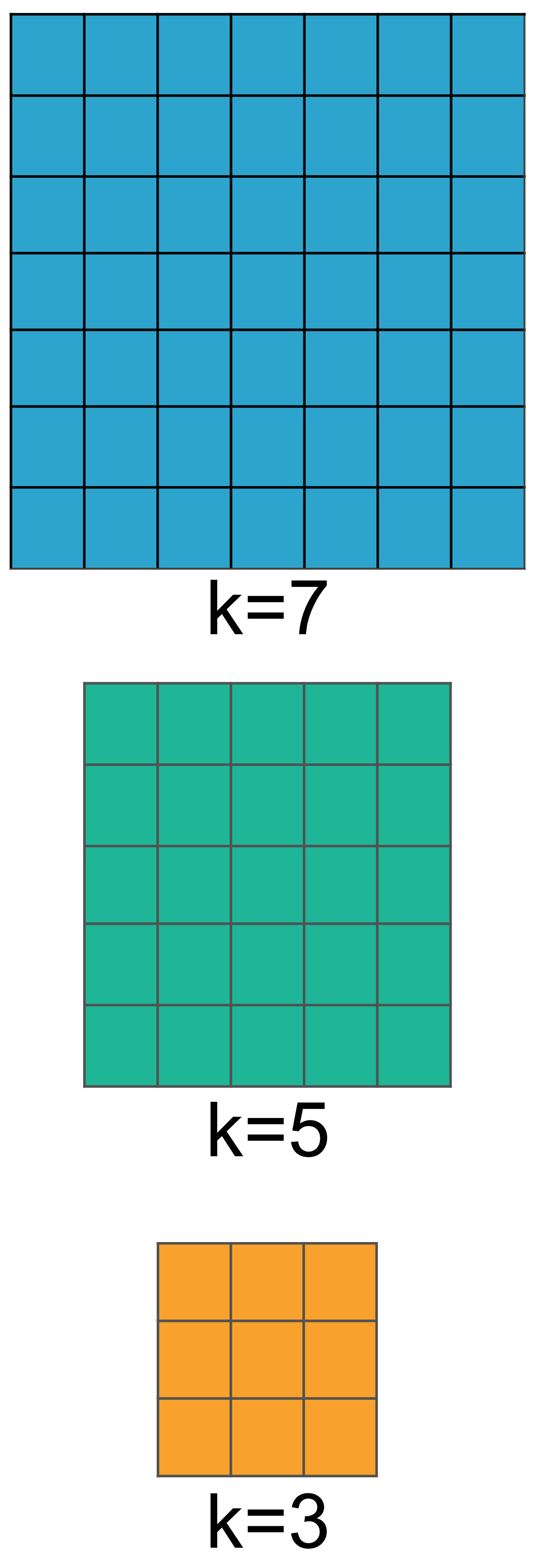
$S = \underline{kernel\ size} \times expansion\ ratio \times depth$

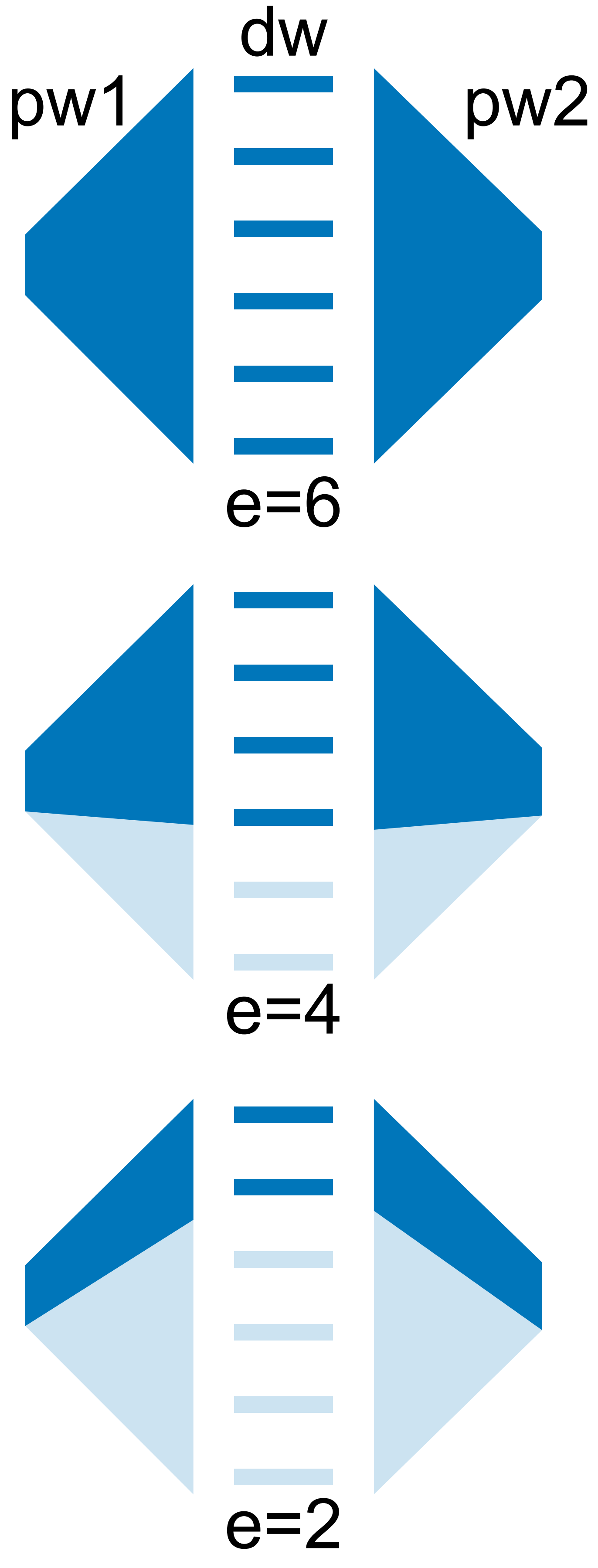


TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

S = kernel size × <u>expansion ratio</u> × depth

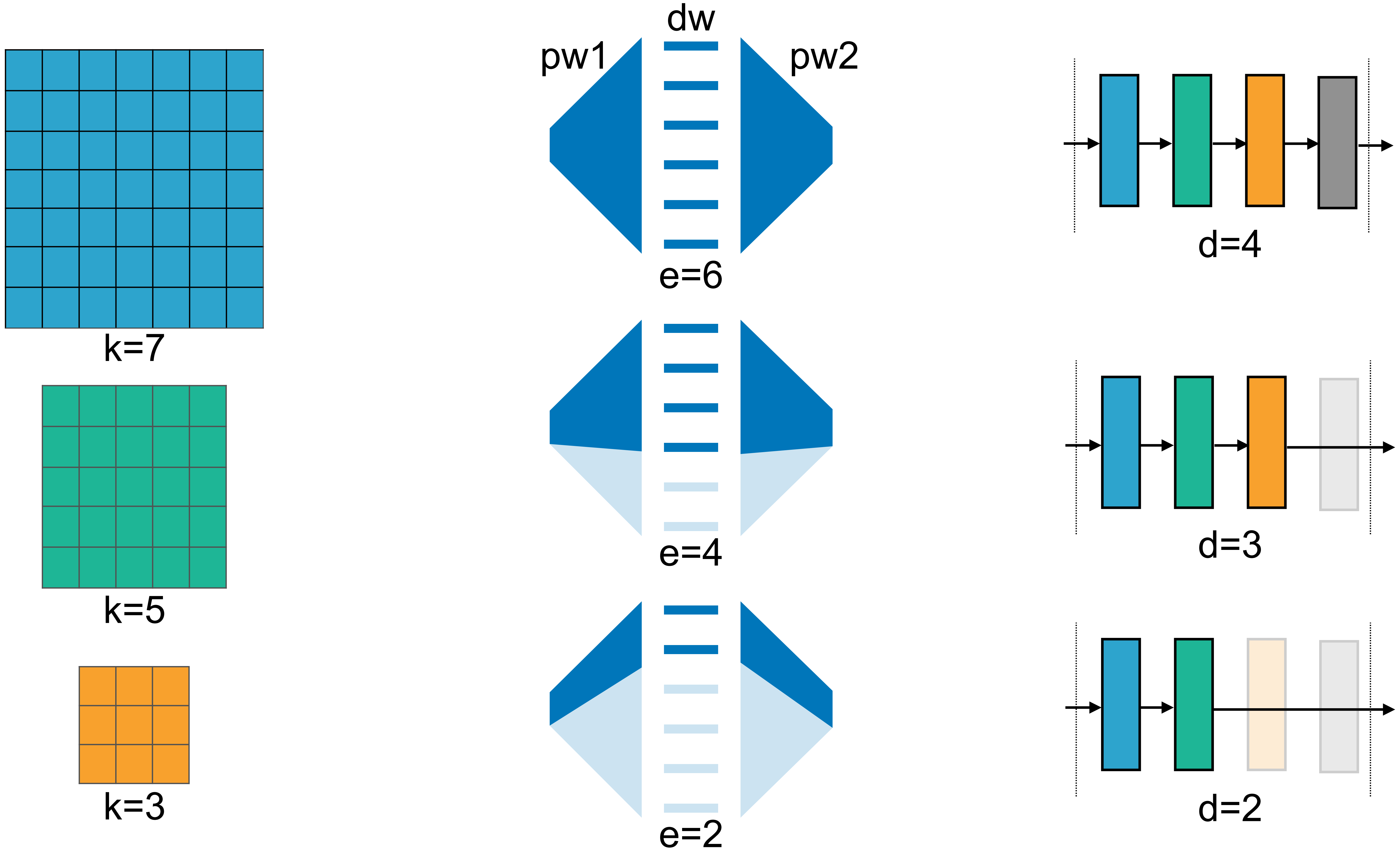




TinyNAS: (1) Automated search space optimization

Revisit ProxylessNAS search space:

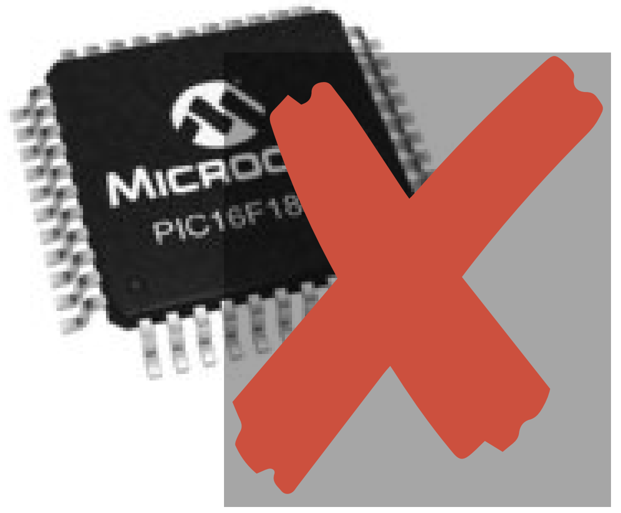
S = kernel size × expansion ratio × <u>depth</u>



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 × expansion ratio × depth × input resolution <u>R</u> × width multiplier <u>W</u>

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

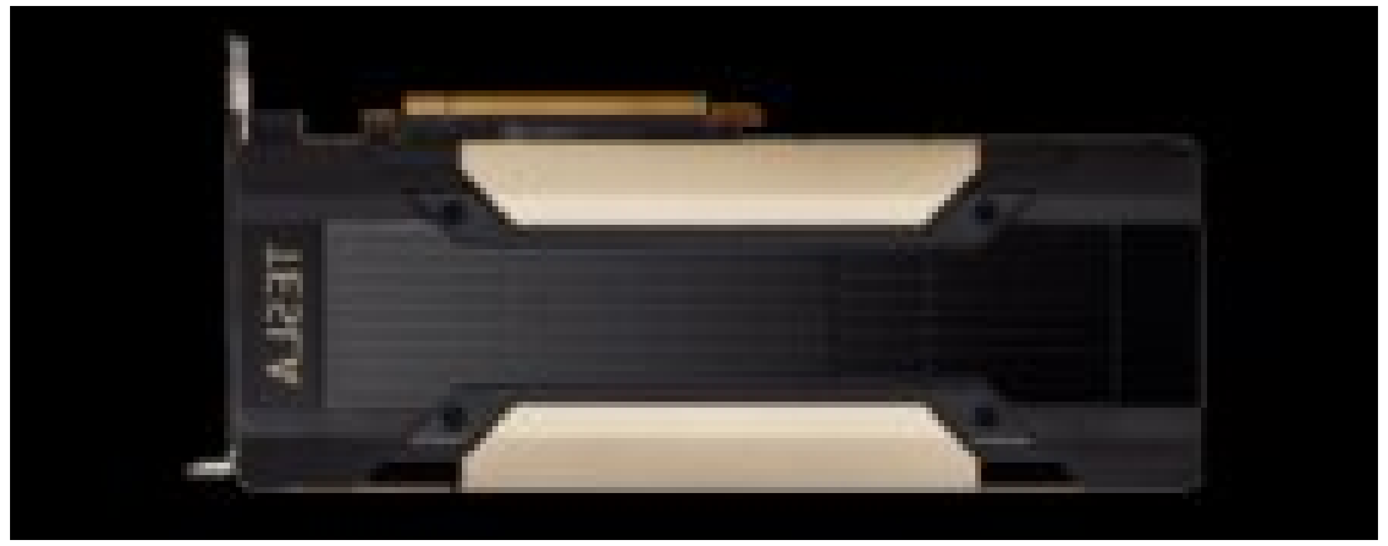
TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

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

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

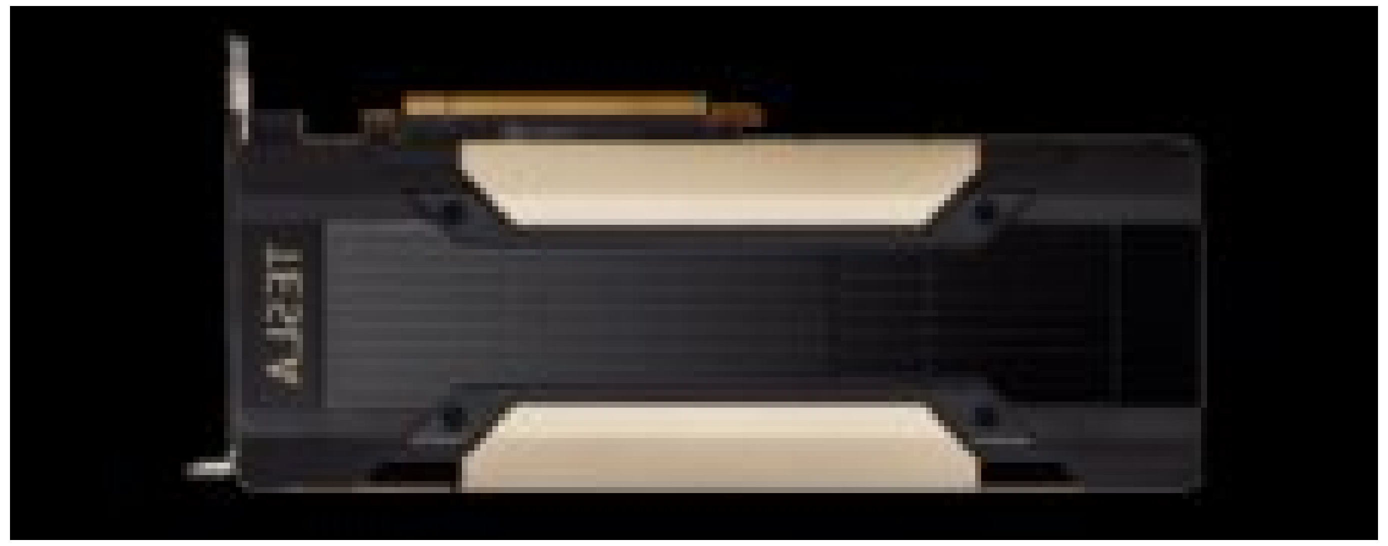
TinyNAS: (1) Automated search space optimization

Extended search space to cover wide range of hardware capacity:

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

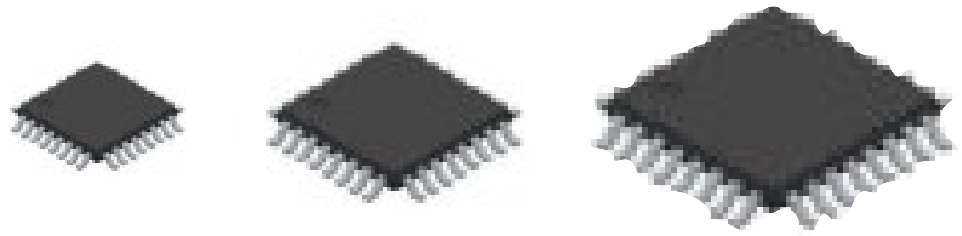
Different R and W for different hardware capacity
(i.e., different optimized sub-space)

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

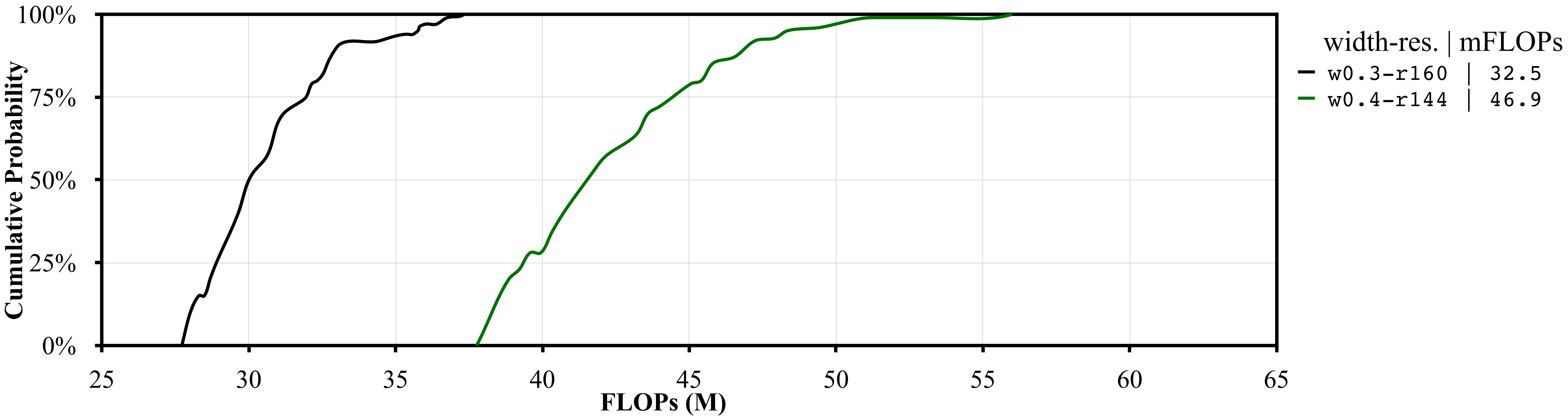
Analyzing **FLOPs distribution** of satisfying models in each search space:
Larger FLOPs -> Larger model capacity -> More likely to give higher accuracy

320kB?

width-res.
w0.3-r160
w0.4-r144

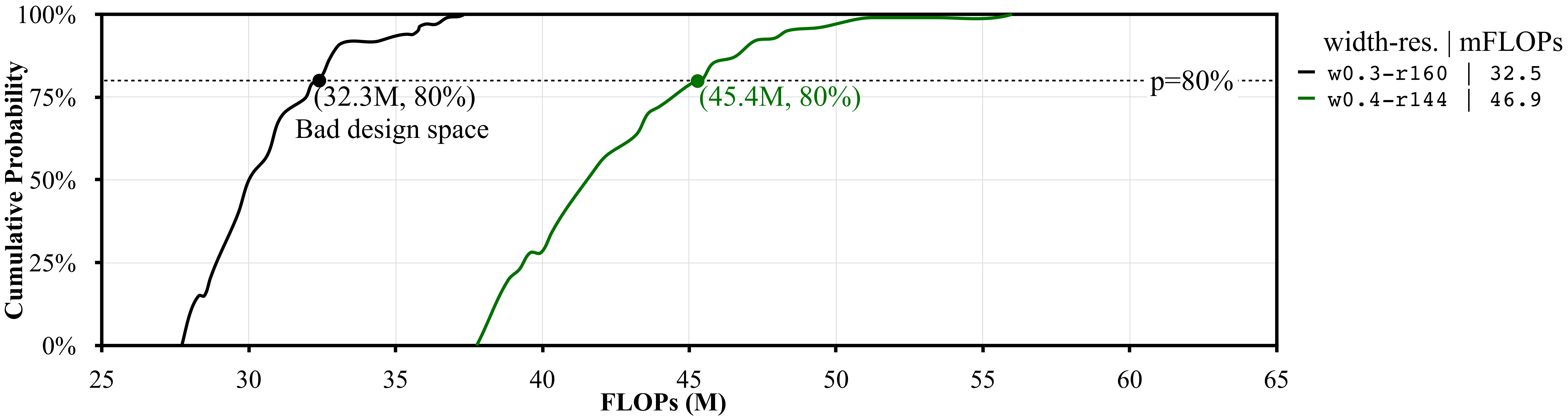
TinyNAS: (1) Automated search space optimization

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



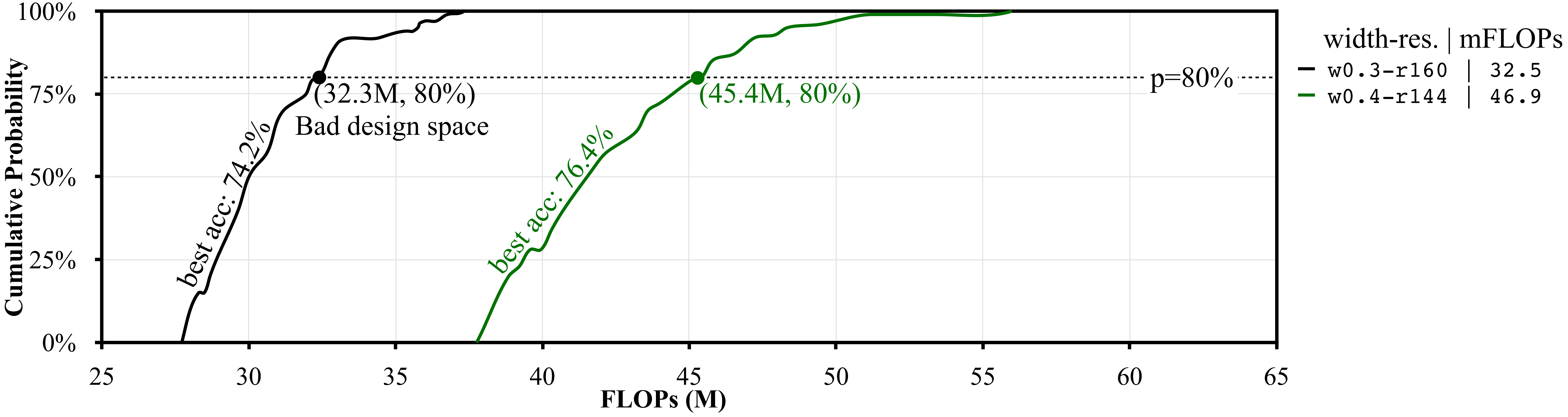
TinyNAS: (1) Automated search space optimization

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



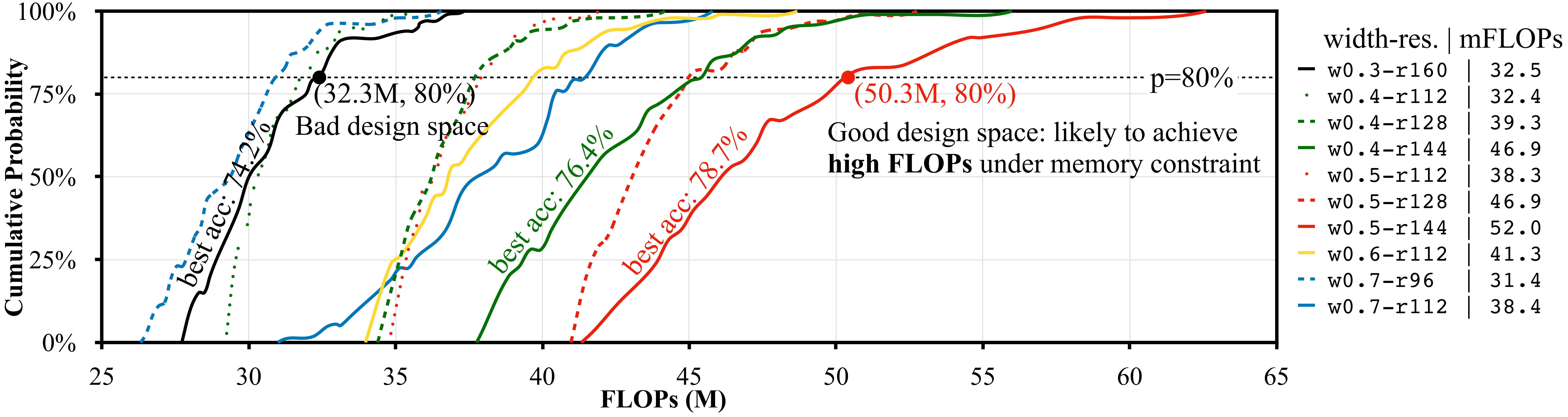
TinyNAS: (1) Automated search space optimization

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



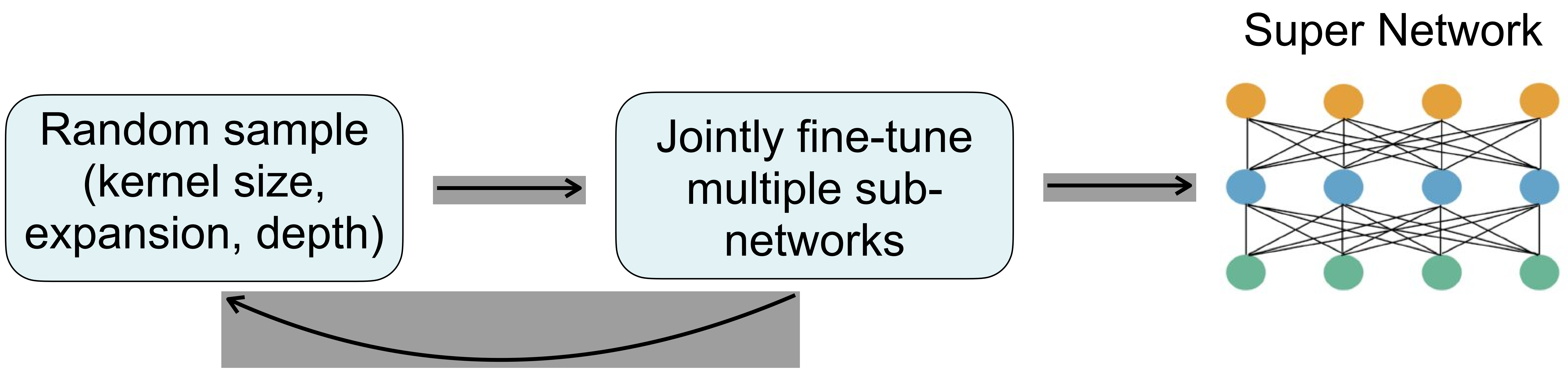
TinyNAS: (1) Automated search space optimization

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

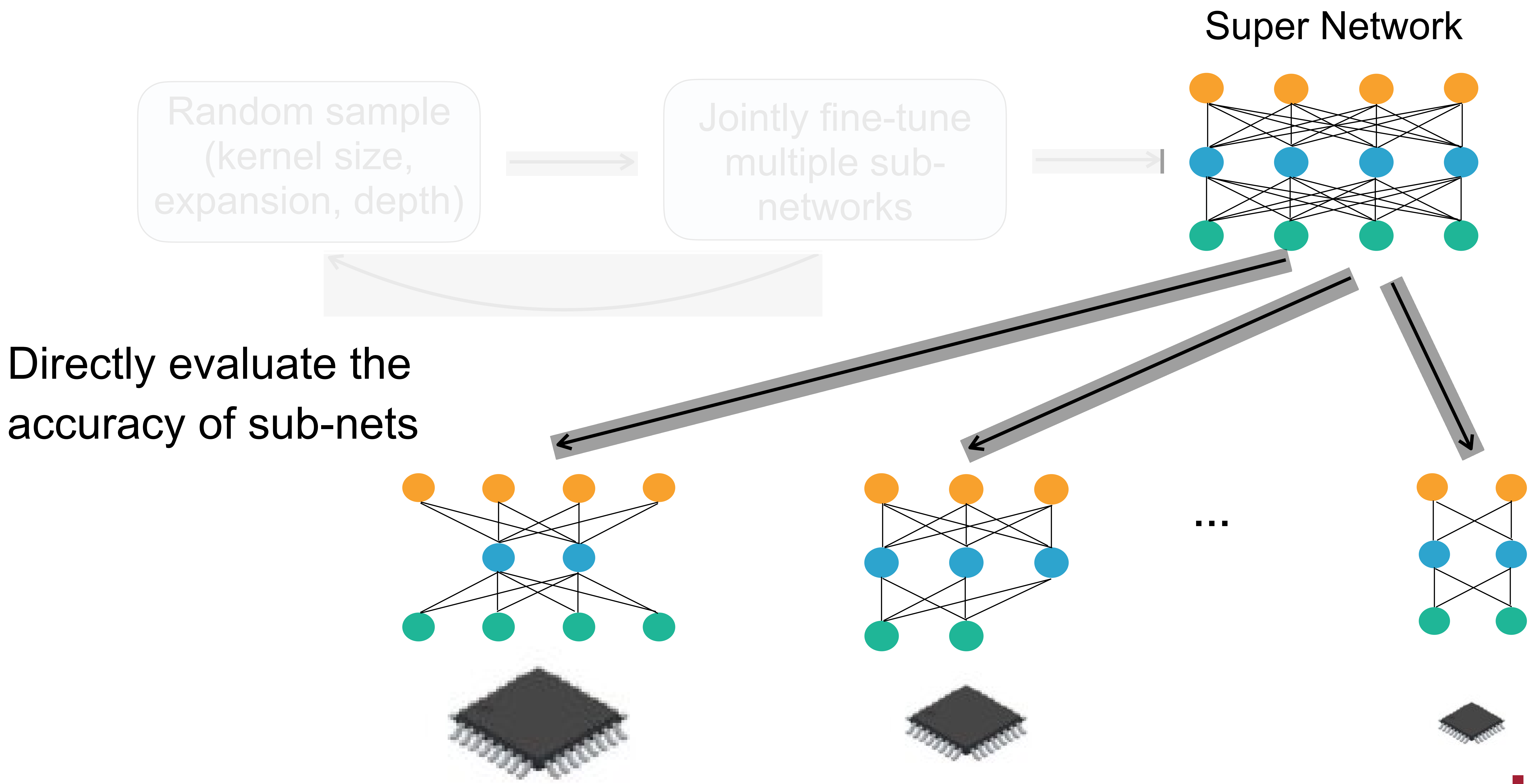
- One-shot NAS through weight sharing



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

TinyNAS: (2) Resource-constrained model specialization

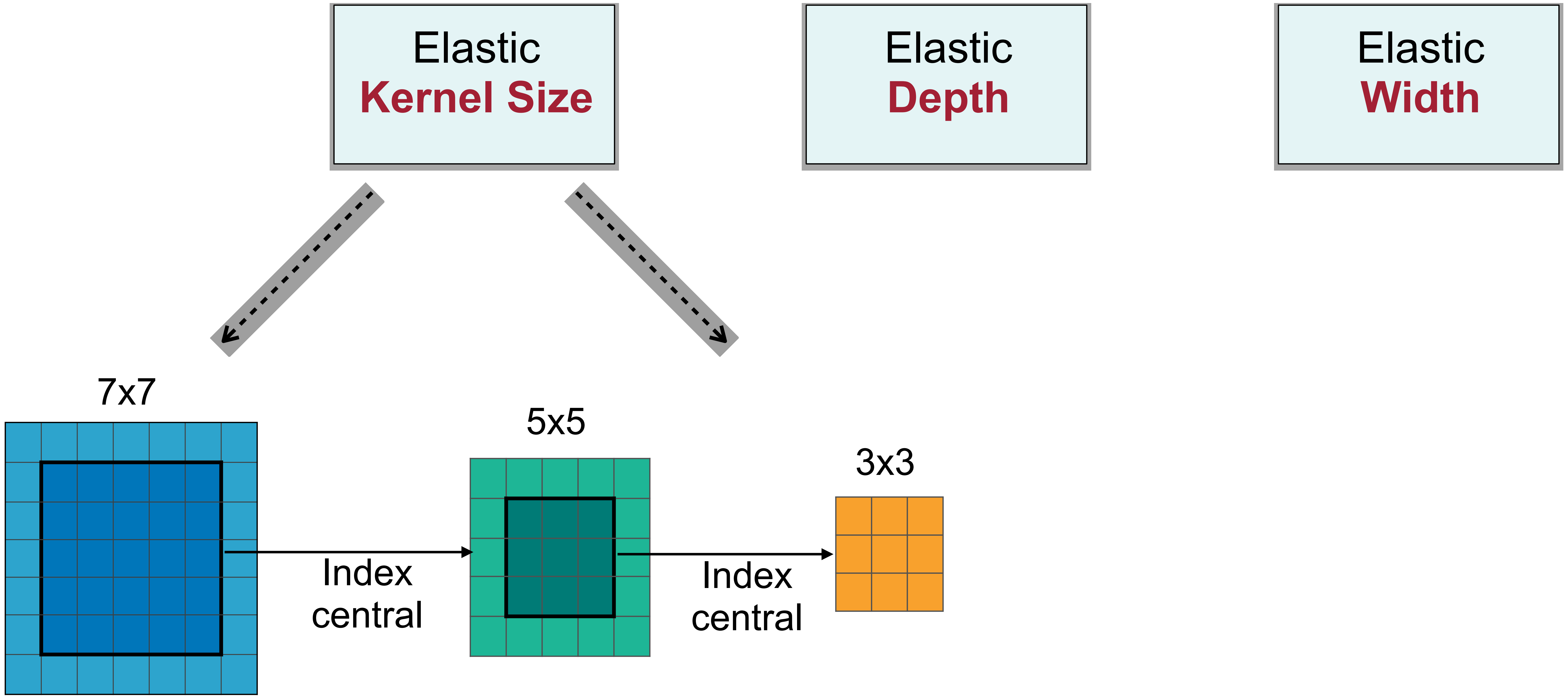
- One-shot NAS through weight sharing



TinyNAS: (2) Resource-constrained model specialization

Elastic **Kernel Size**	Elastic **Depth**	Elastic **Width**

TinyNAS: (2) Resource-constrained model specialization



Start with **full** kernel size

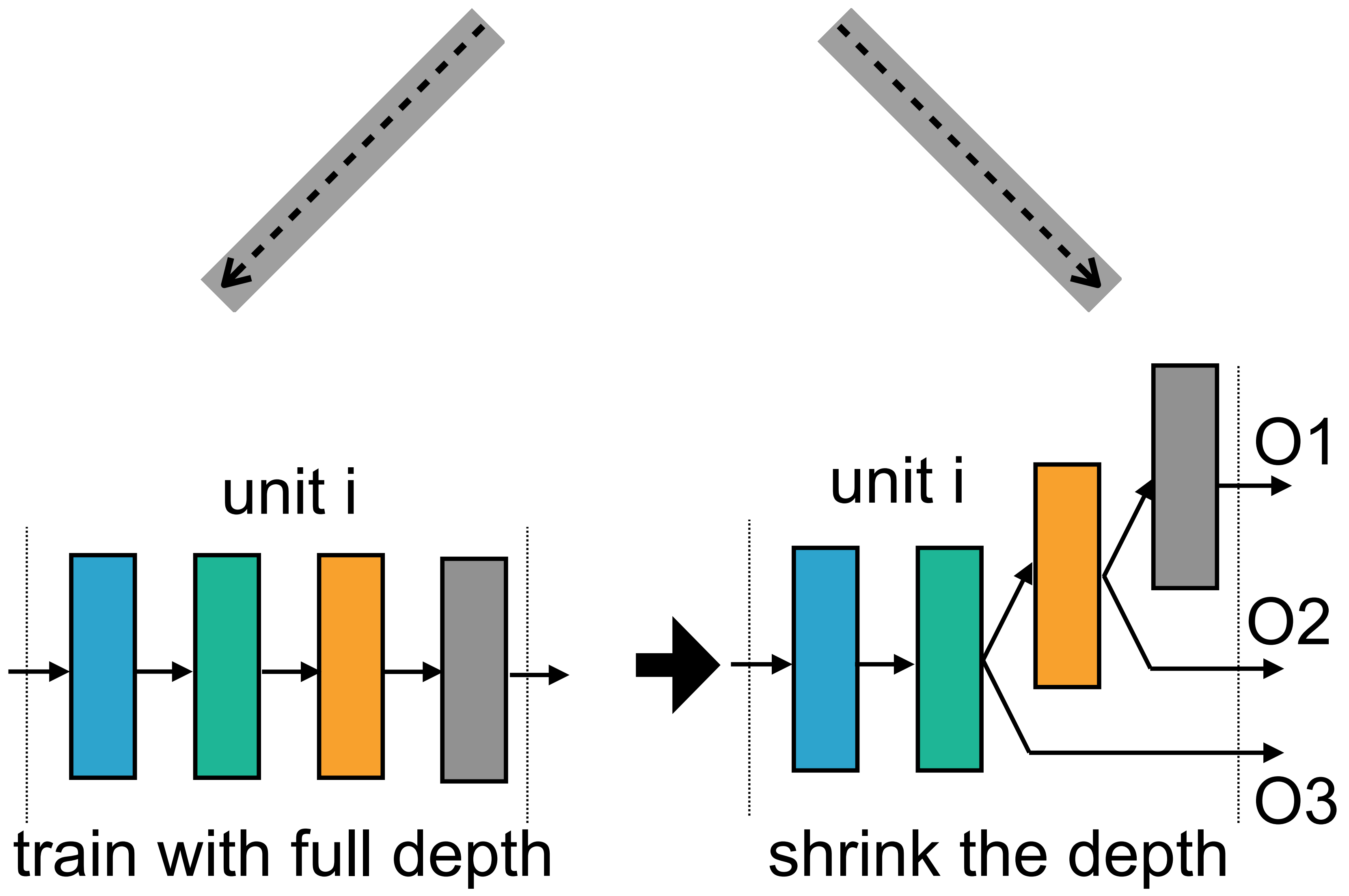
Smaller kernel takes centered weights

TinyNAS: (2) Resource-constrained model specialization

Elastic **Kernel Size**

Elastic **Depth**

Elastic **Width**



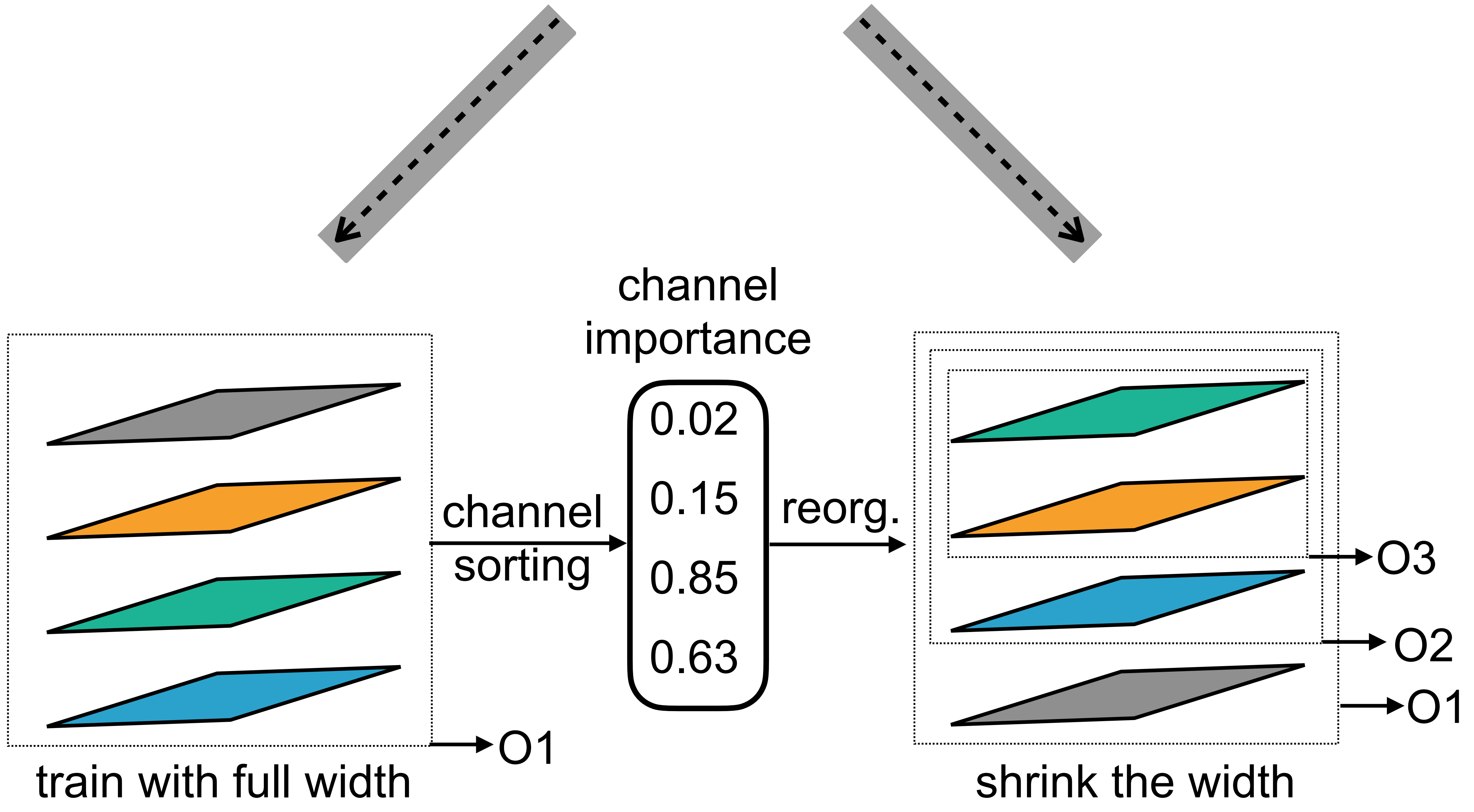
Allow **later** layers in each unit
to be skipped to reduce the depth

TinyNAS: (2) Resource-constrained model specialization

Elastic
Kernel Size

Elastic
Depth

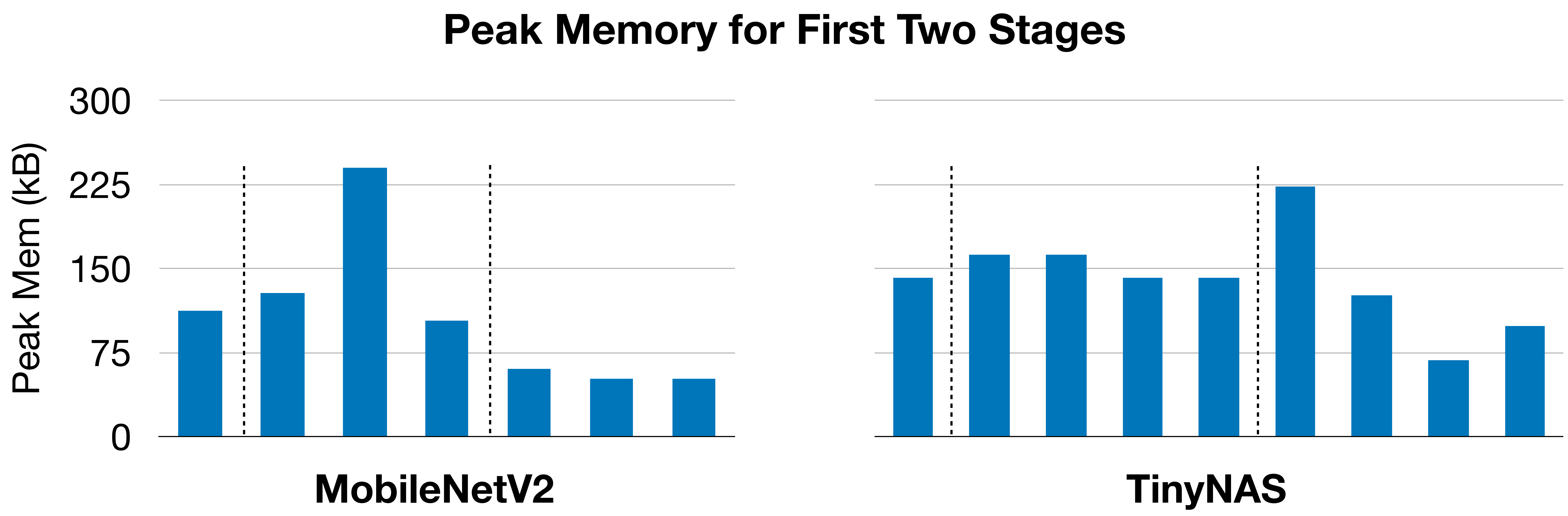
Elastic
Width



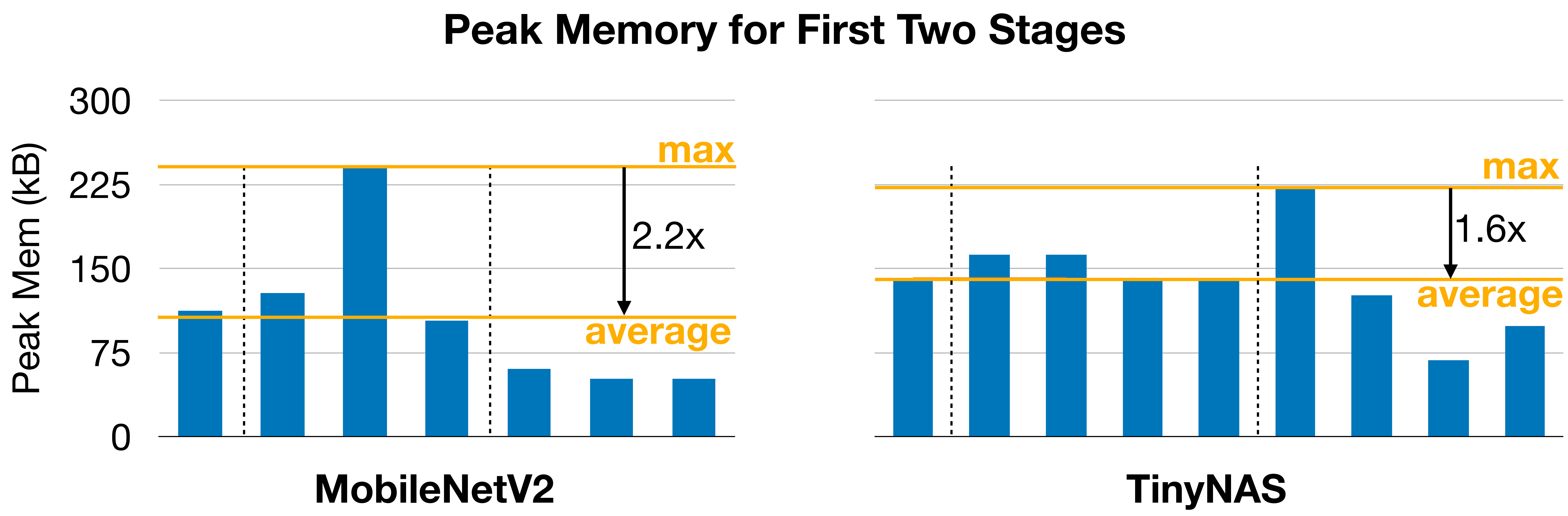
Shrink the width

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

TinyNAS Better Utilizes the Memory



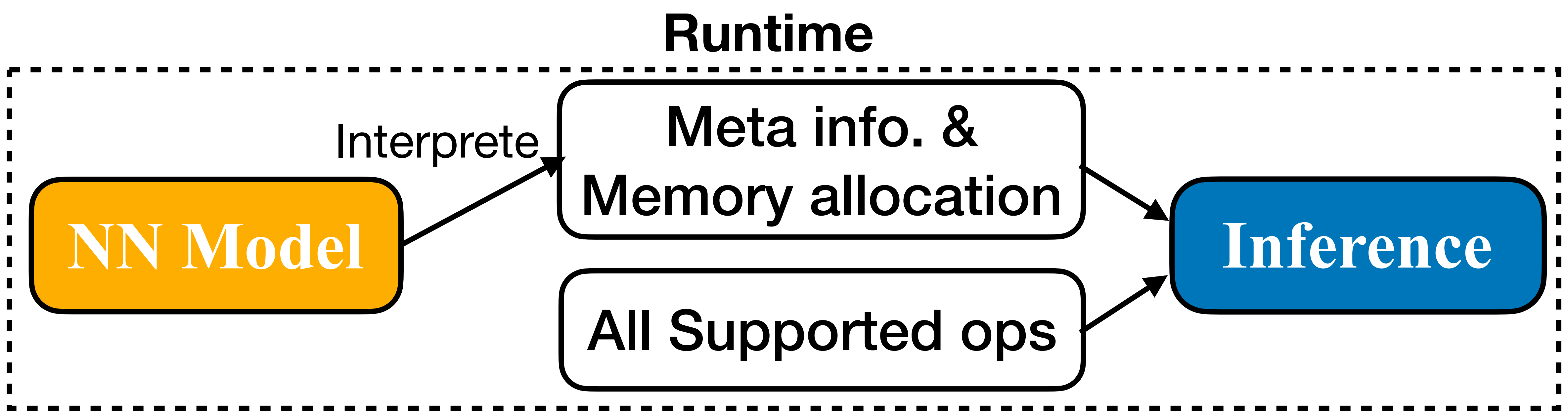
TinyNAS Better Utilizes the Memory



- 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

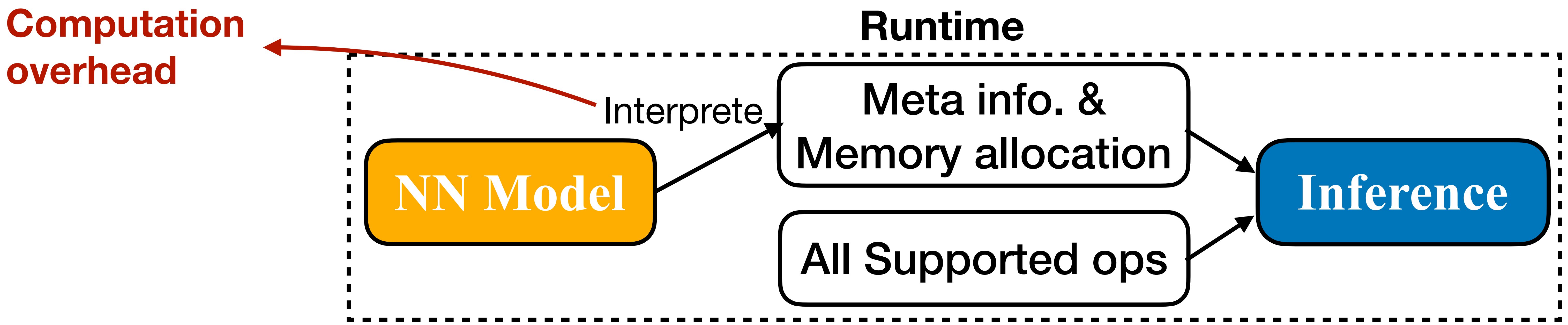
1. Reducing overhead with separated compilation & runtime



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

TinyEngine: A Memory-Efficient Inference Library

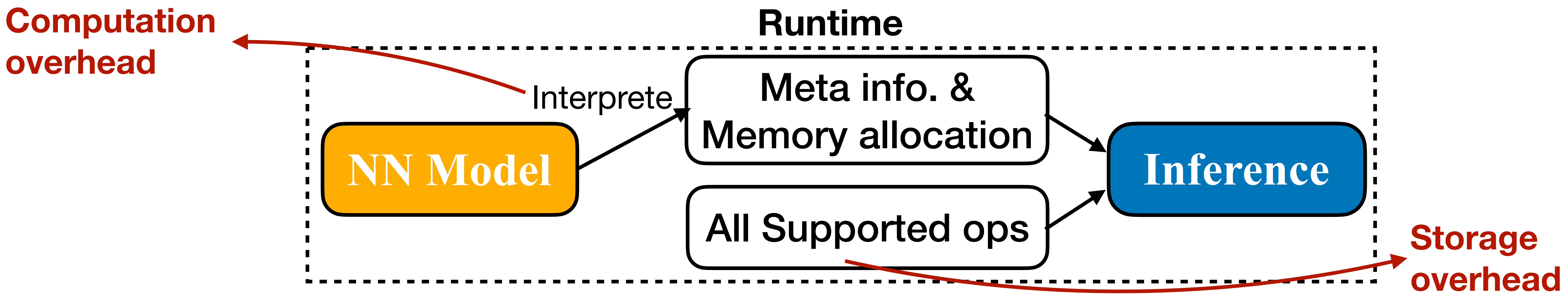
1. Reducing overhead with separated compilation & runtime



(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

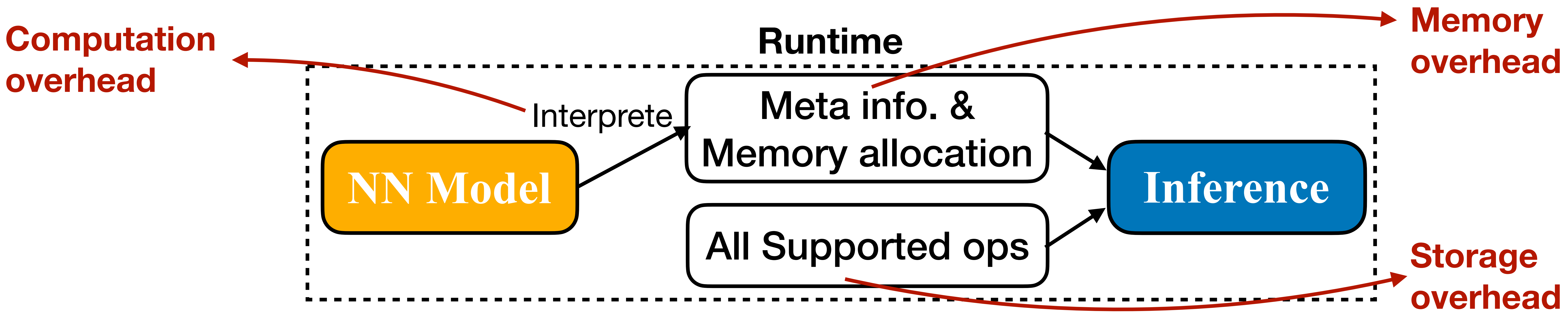


(a) Existing libraries based on **runtime interpretation**

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

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime

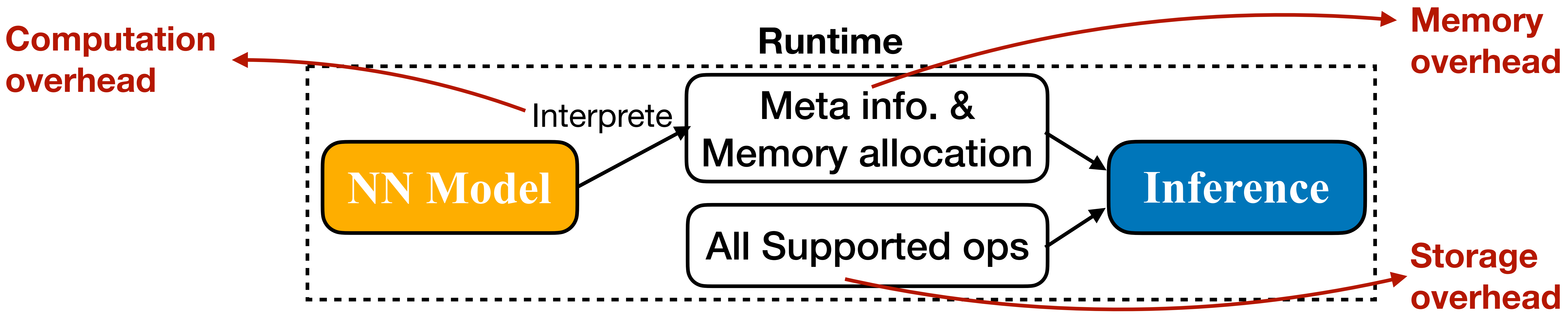


(a) Existing libraries based on **runtime interpretation**

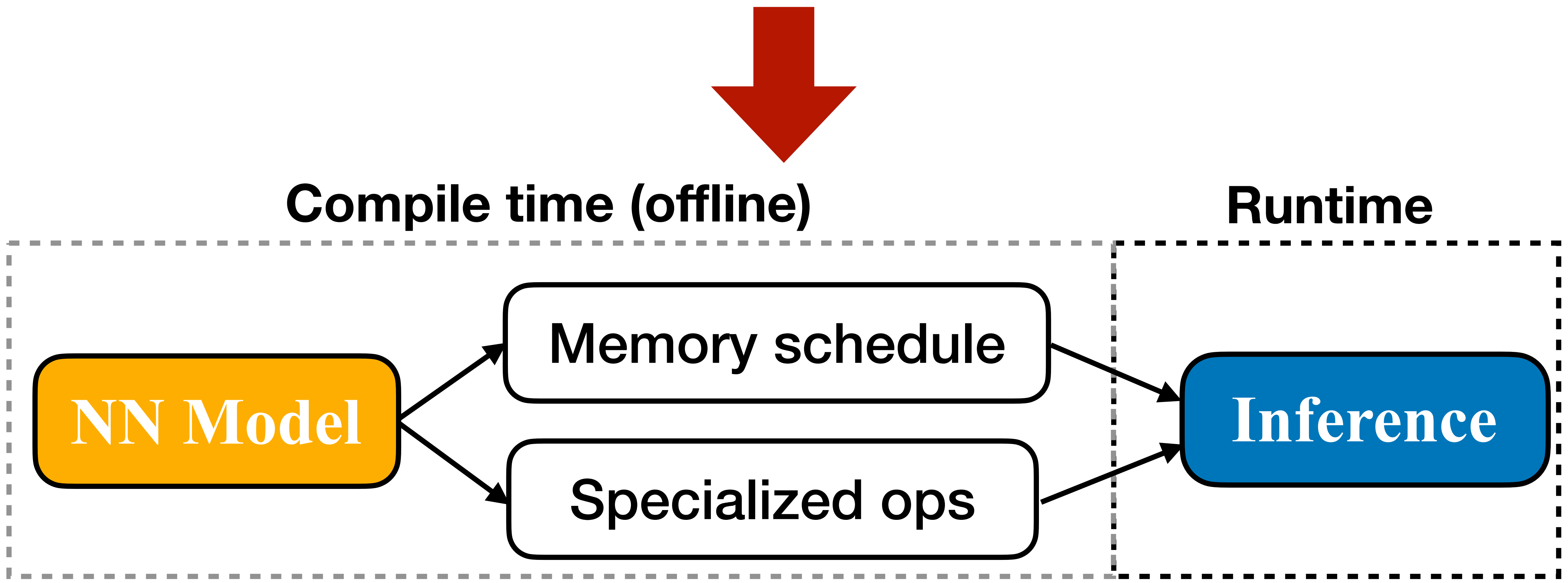
e.g., *TF-Lite Micro, CMSIS-NN*

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime



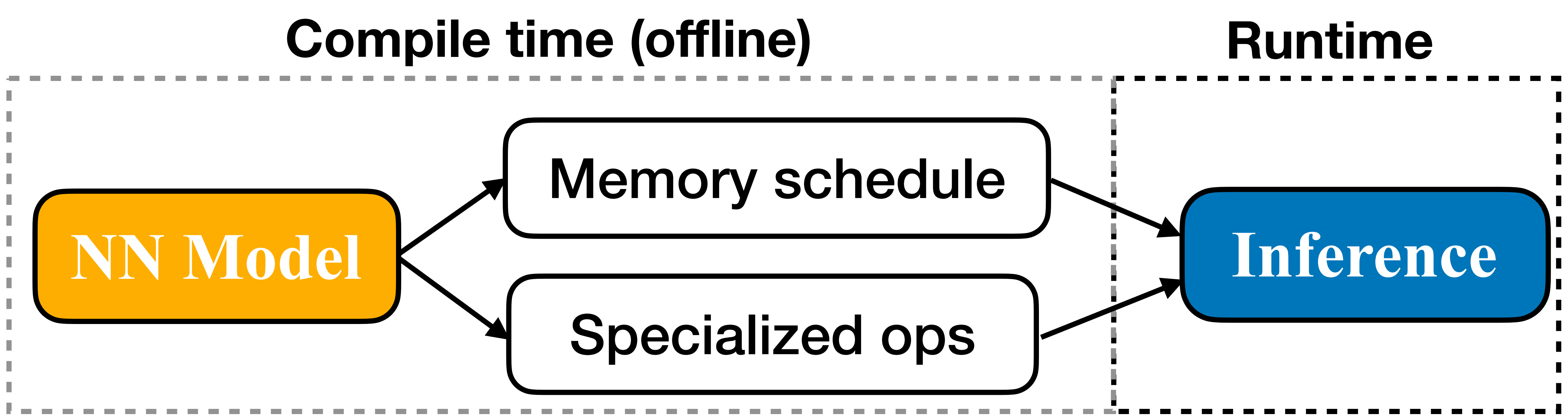
(a) Existing libraries based on **runtime interpretation**
e.g., *TF-Lite Micro, CMSIS-NN*



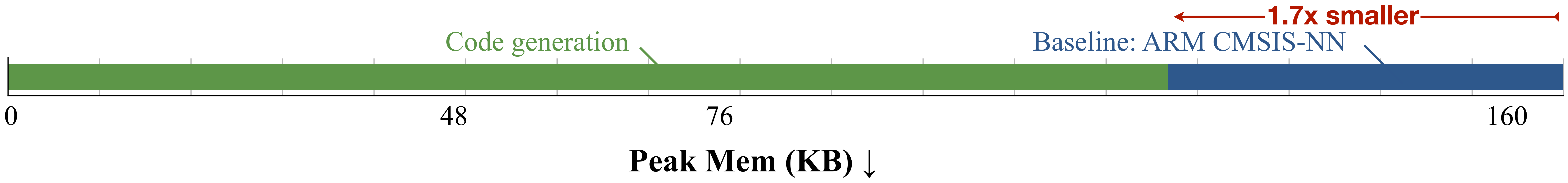
(b) TinyEngine: **Model-adaptive** code generation.

TinyEngine: A Memory-Efficient Inference Library

1. Reducing overhead with separated compilation & runtime



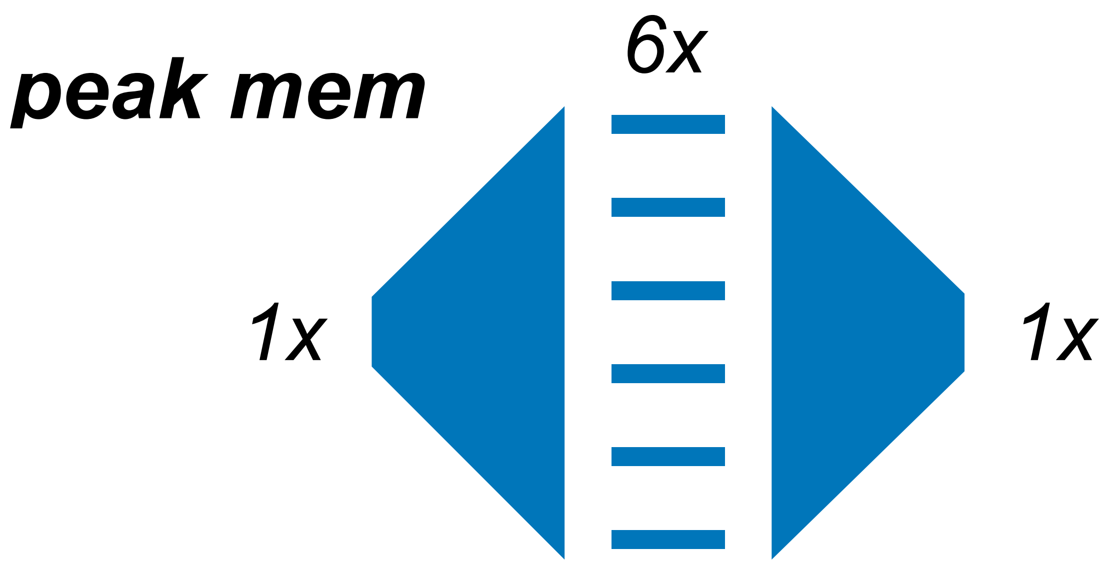
(b) TinyEngine: **Model-adaptive** code generation.



TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

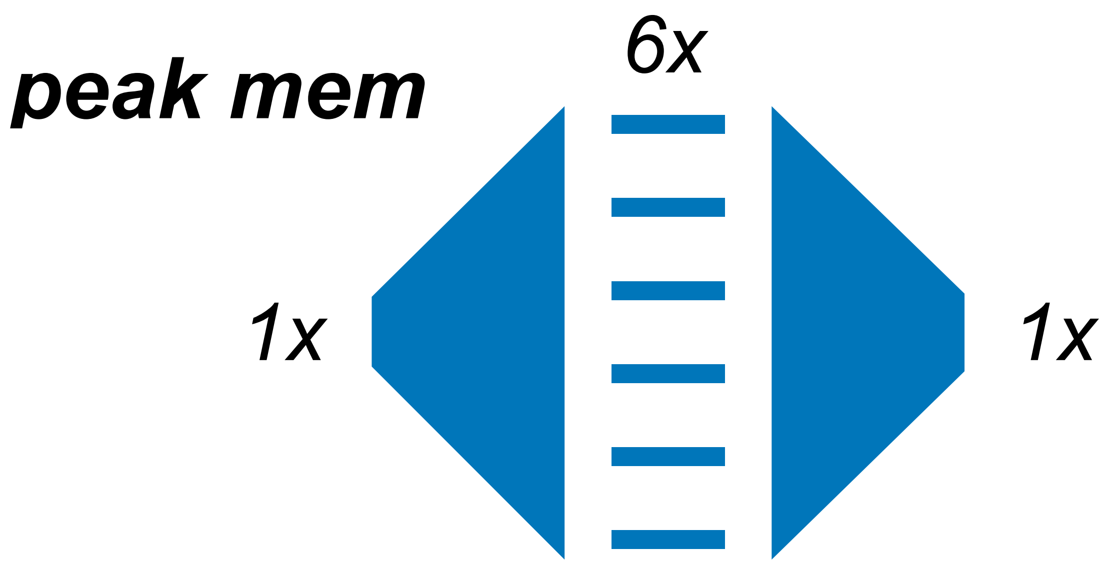
The dilemma of *inverted bottleneck*

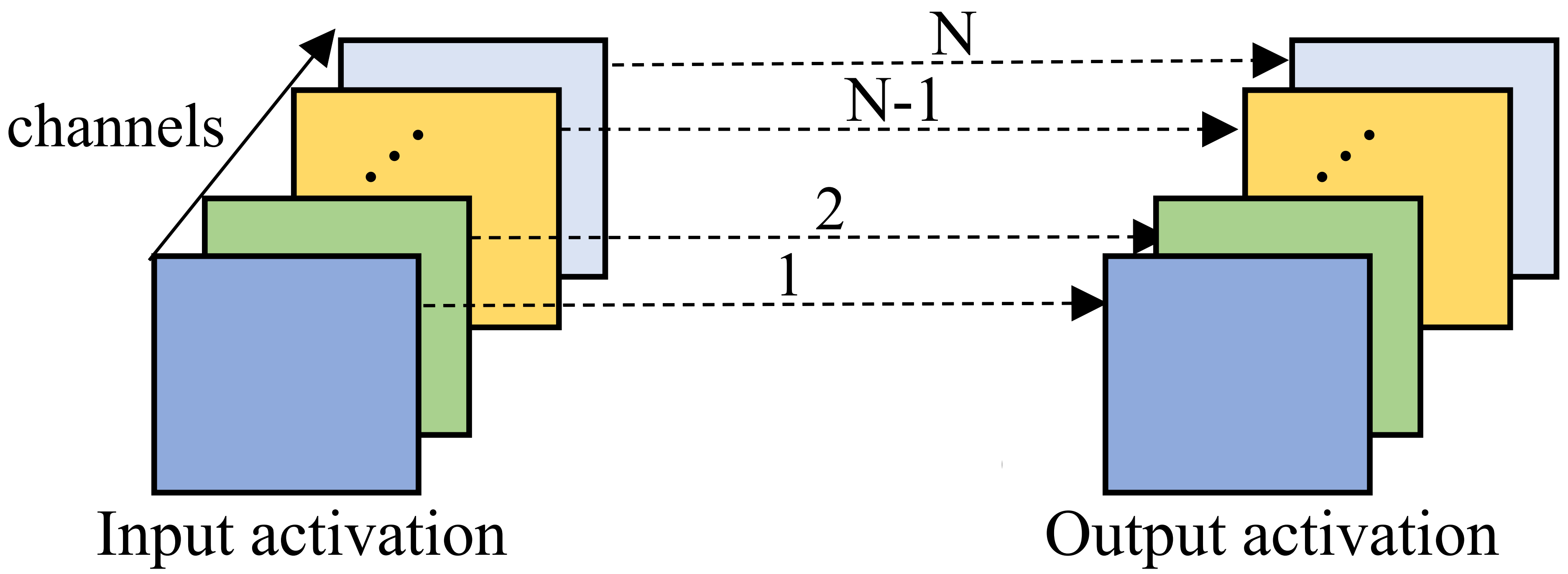


TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

The dilemma of *inverted bottleneck*





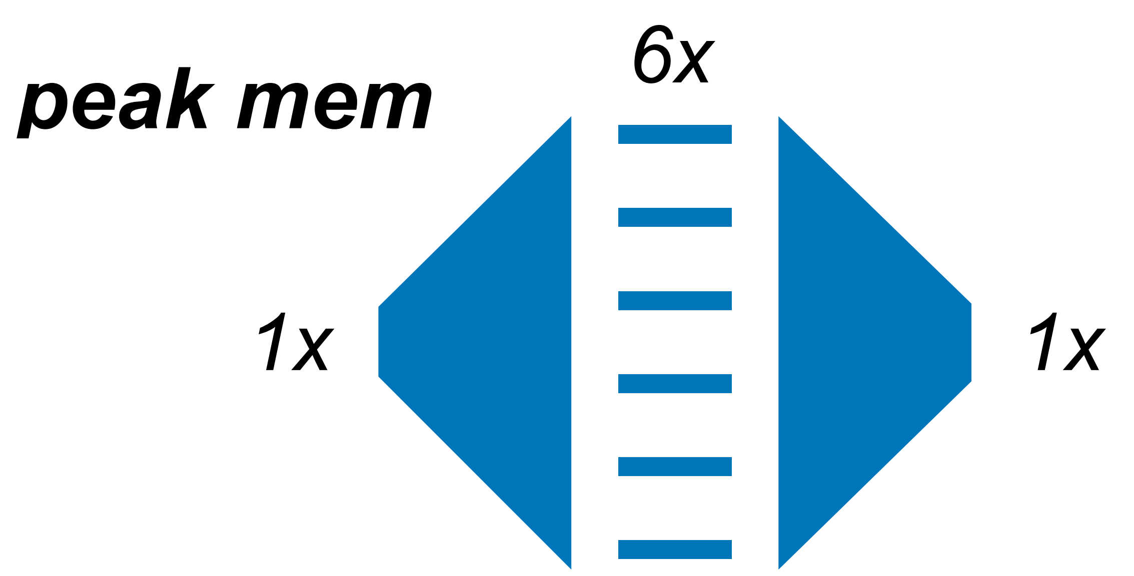
(a) Depth-wise convolution

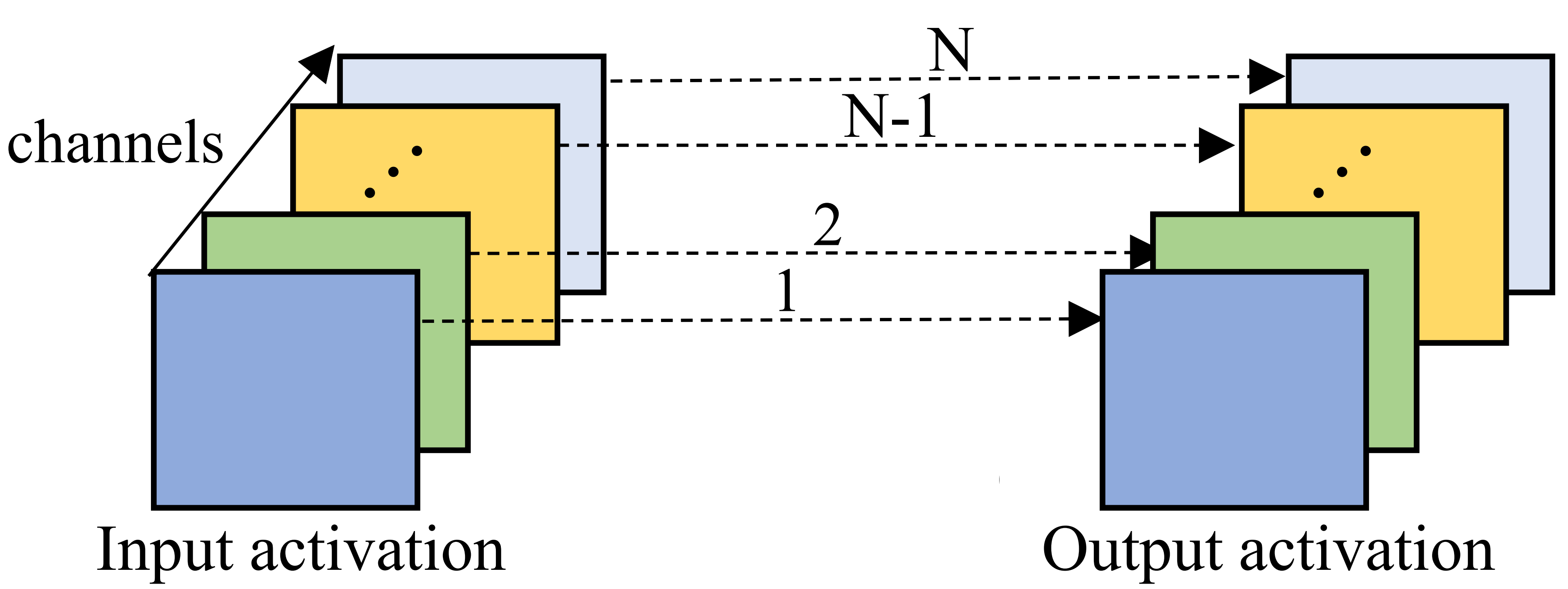
Peak Memory: 2N

TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

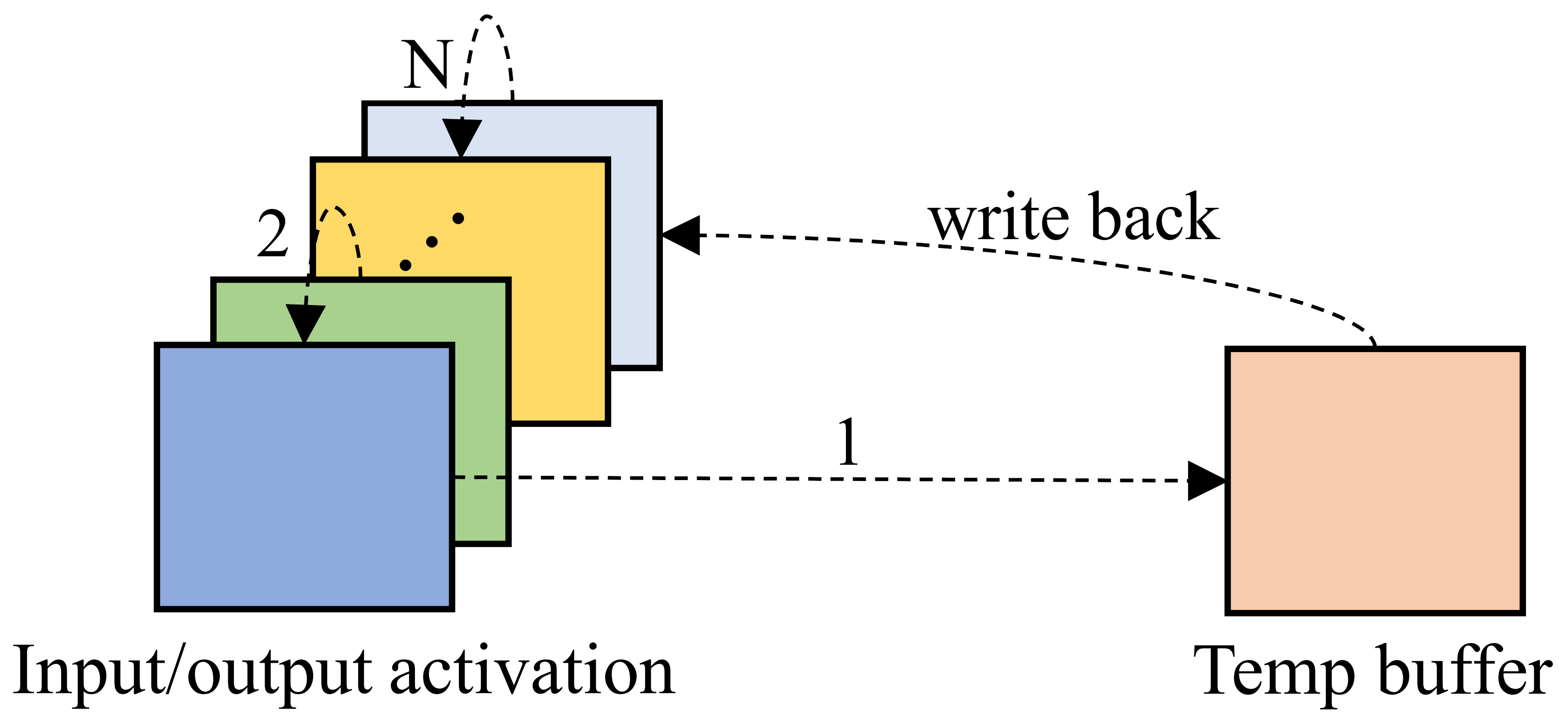
The dilemma of *inverted bottleneck*





(a) Depth-wise convolution

Peak Memory: 2N



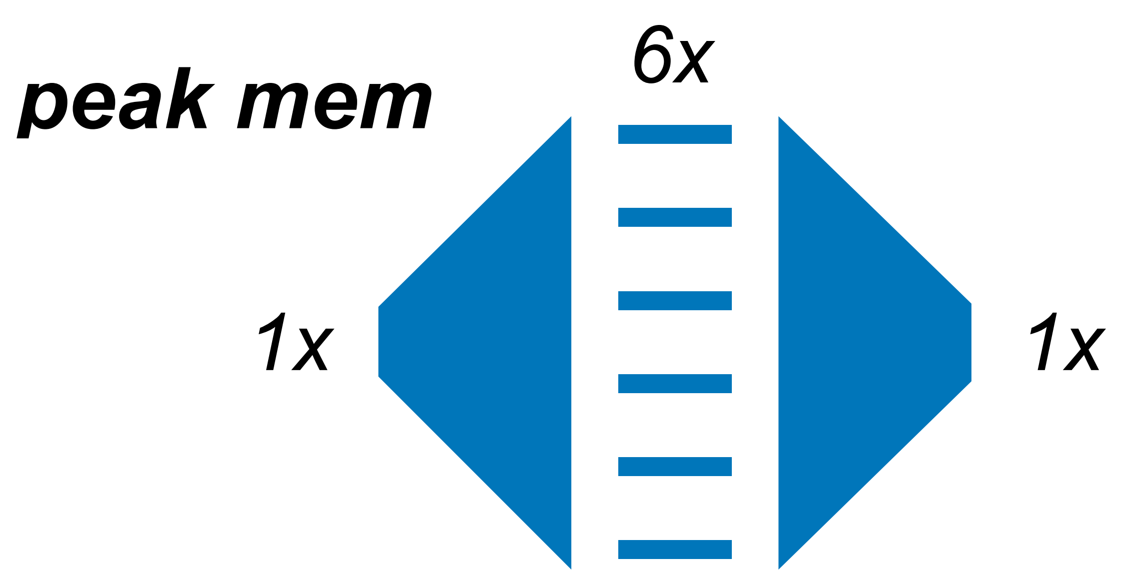
(b) In-place depth-wise convolution

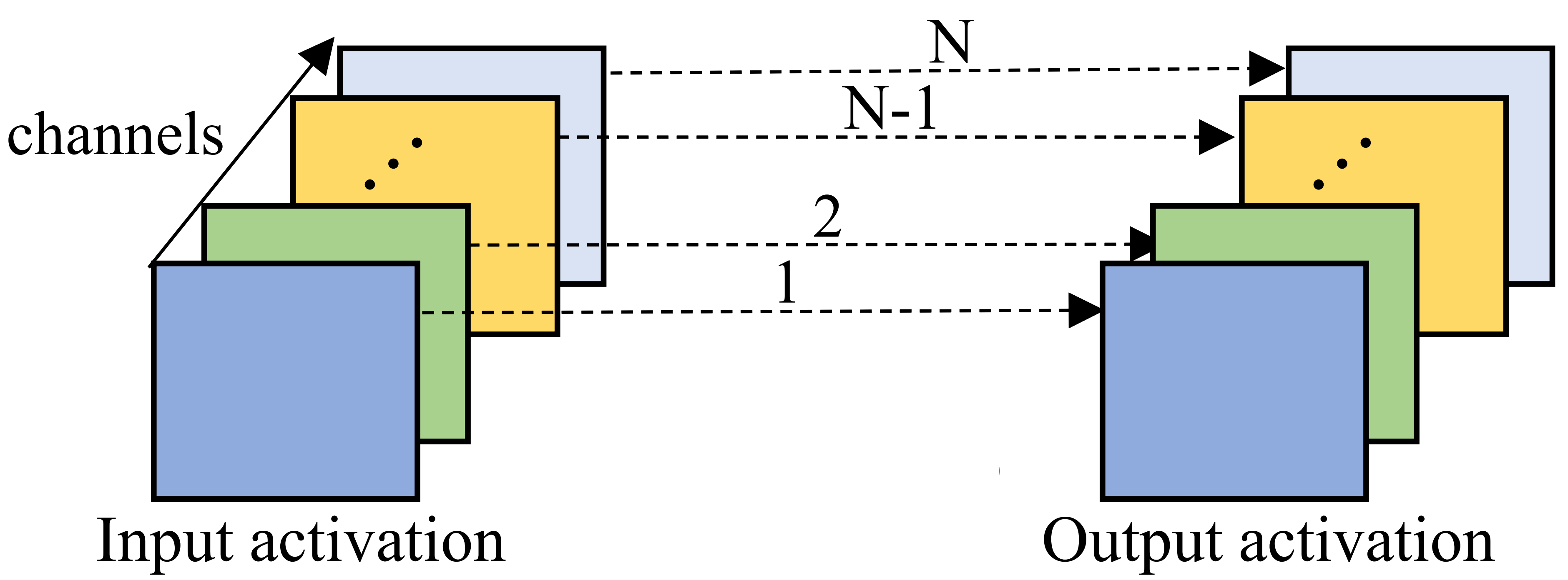
Peak Memory: N+1

TinyEngine: A Memory-Efficient Inference Library

2. In-place depth-wise convolution

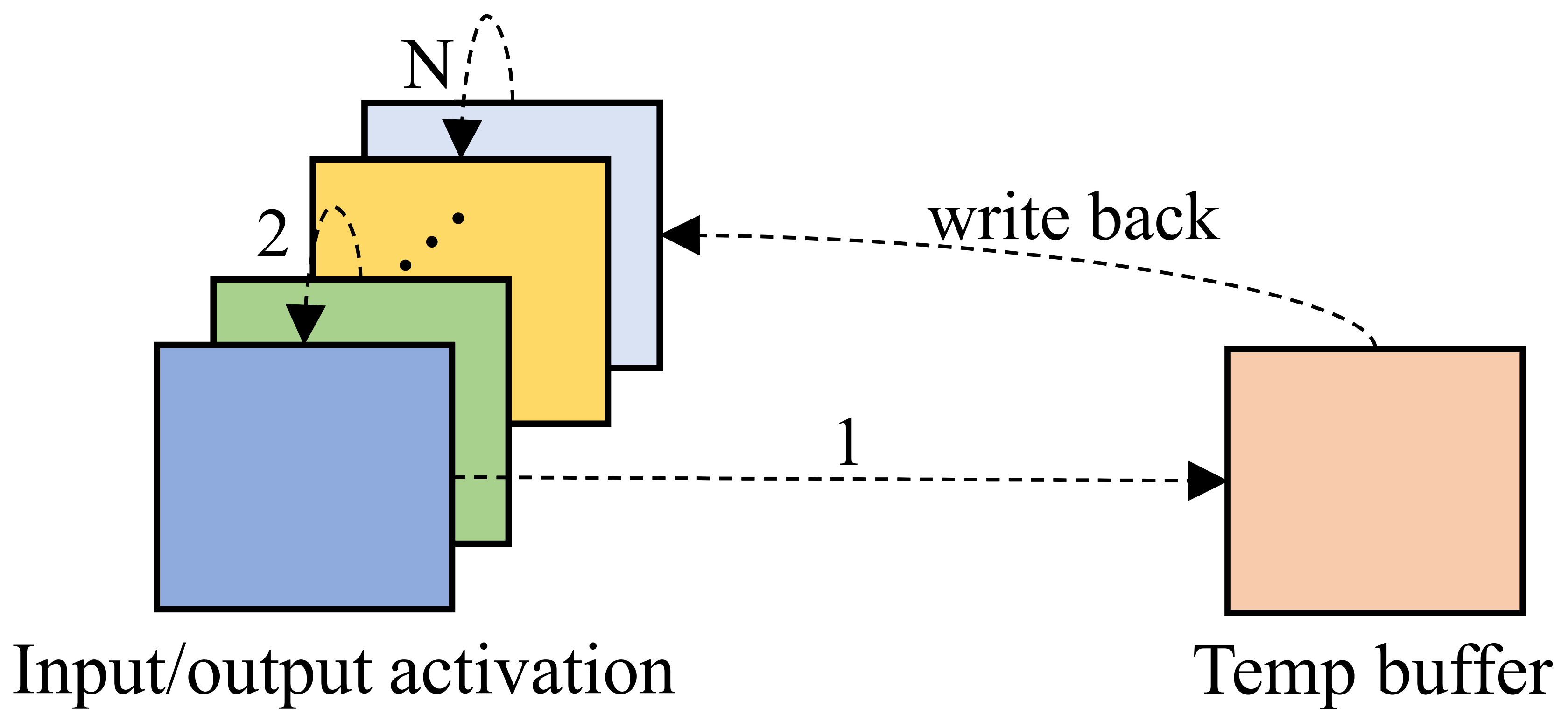
The dilemma of *inverted bottleneck*





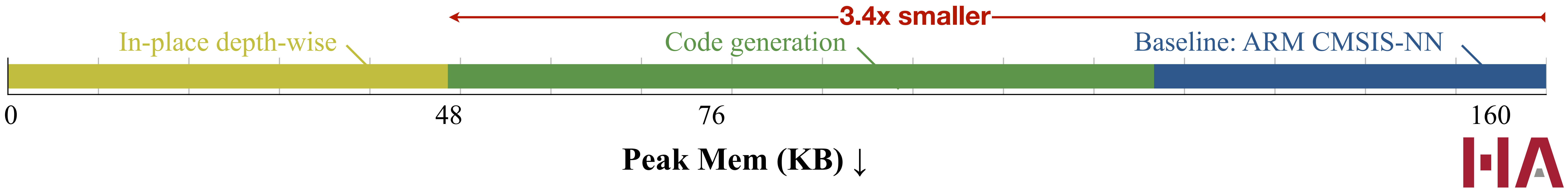
(a) Depth-wise convolution

Peak Memory: 2N



(b) In-place depth-wise convolution

Peak Memory: N+1



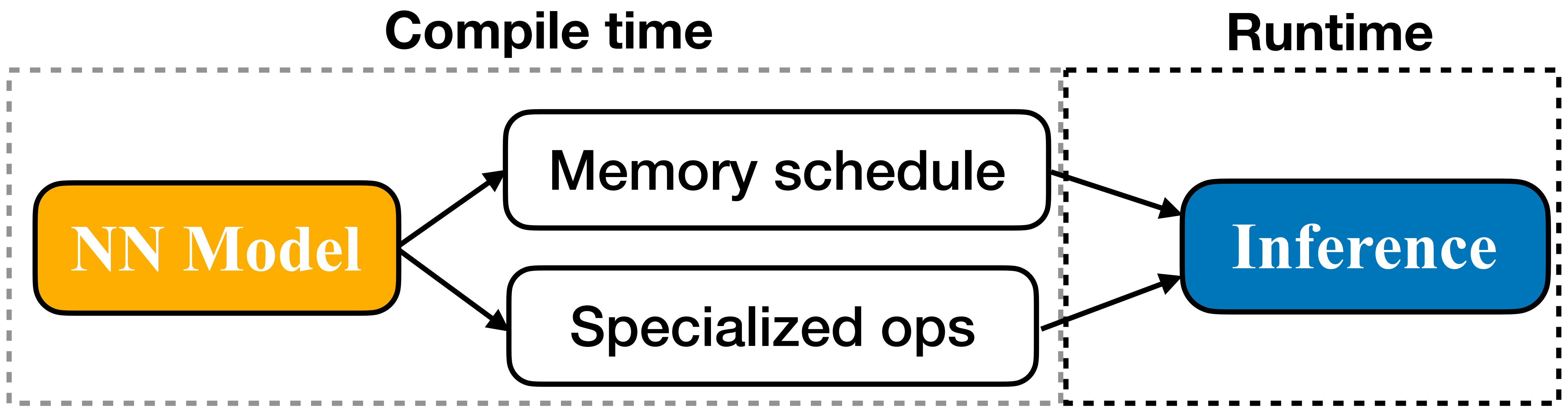
TinyEngine: Faster Inference Speed

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

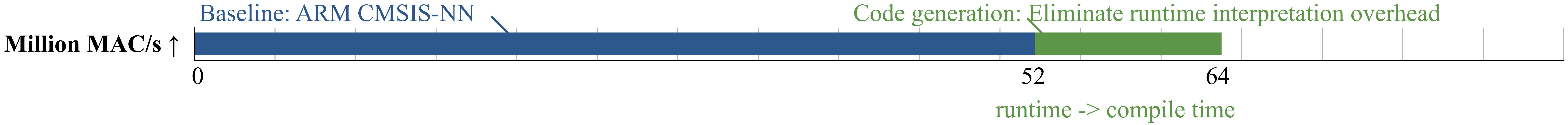
TinyEngine: Faster Inference Speed

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

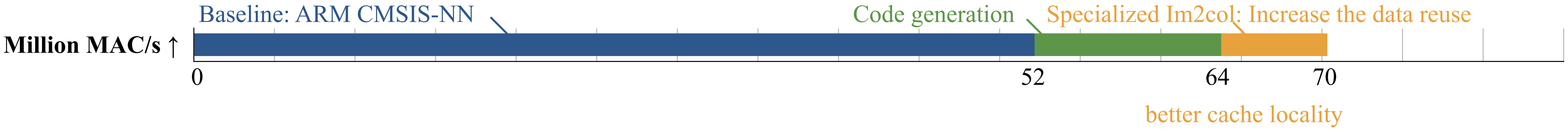
(2) Model-adaptive memory scheduling -> Increase data reuse for each layer

(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

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

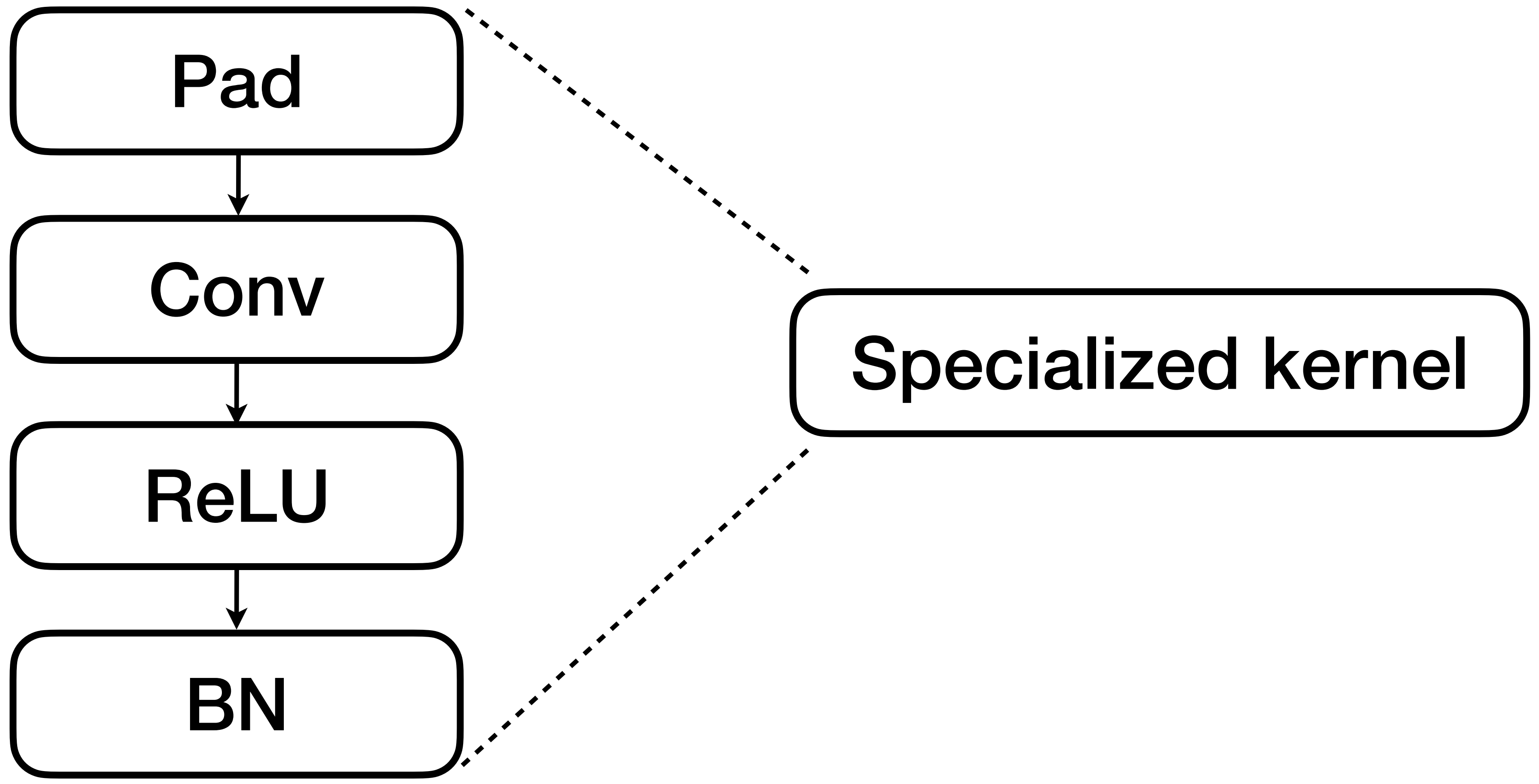


TinyEngine: Faster Inference Speed

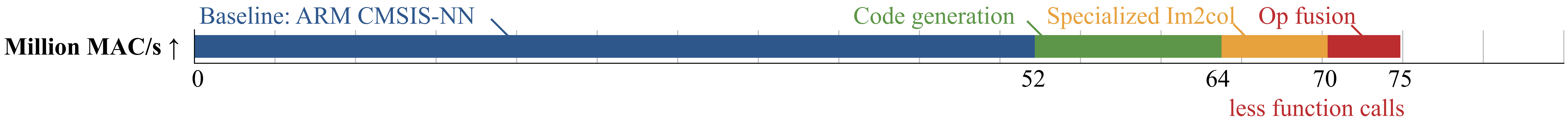
Analyzing **Million MAC/s** improved by each technique

(3) Computation Kernel Specialization: Operation fusion

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



- Minimize memory footprint
- Optimize the overall computation



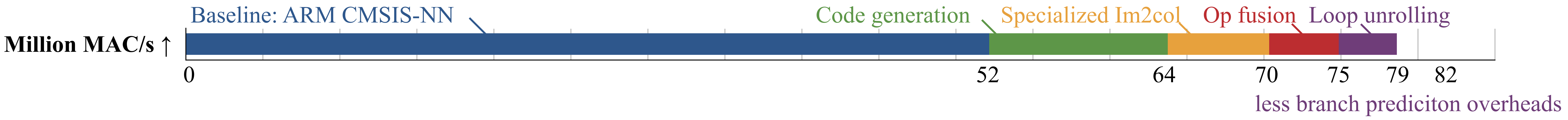
TinyEngine: Faster Inference Speed

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

(3) Computation Kernel Specialization: Loop unrolling

e.g., fully unroll for 3x3 conv

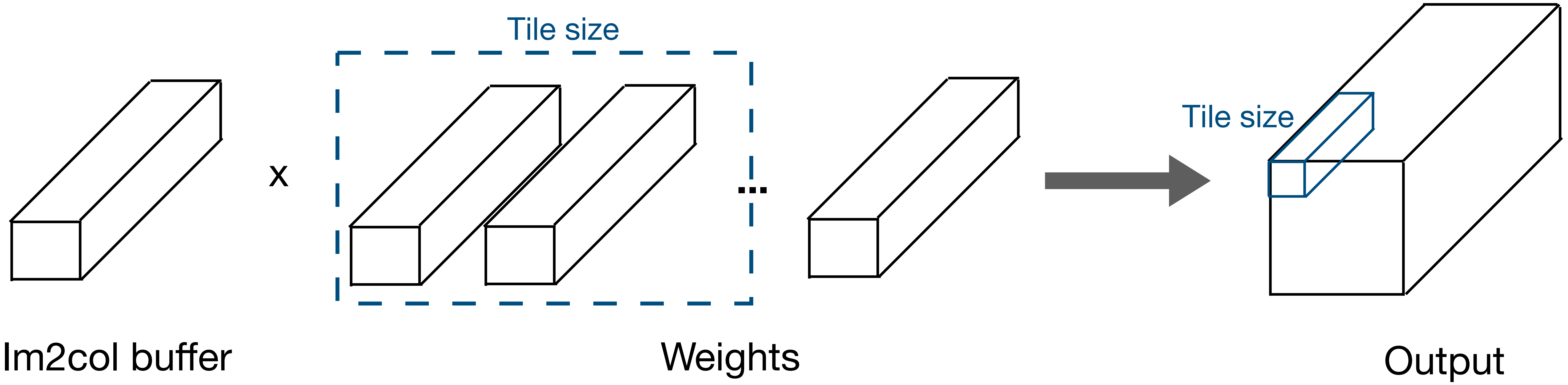
- Eliminate the branch instruction overheads of loops



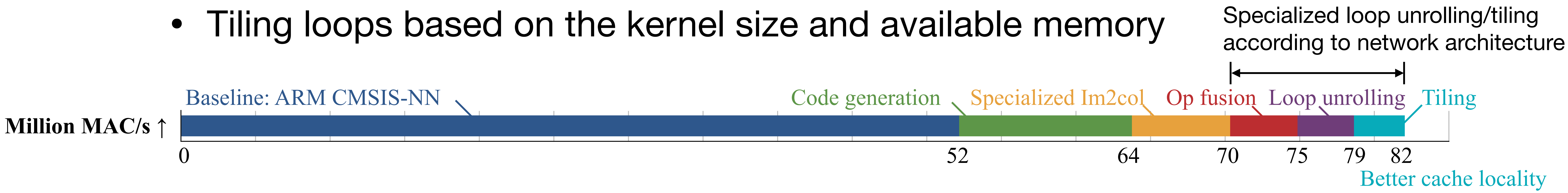
TinyEngine: Faster Inference Speed

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

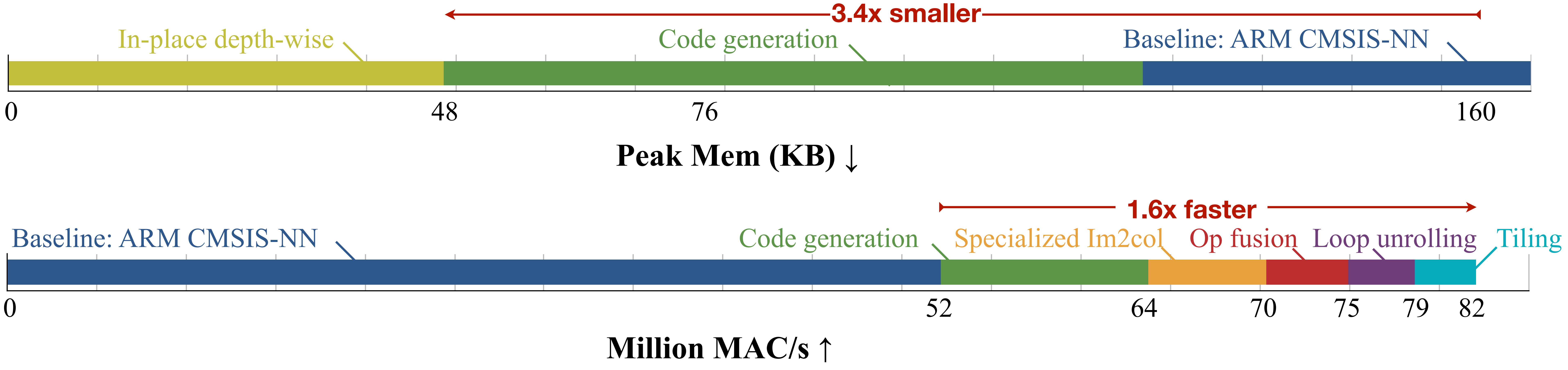
(3) Computation Kernel Specialization: Loop tiling for each layer



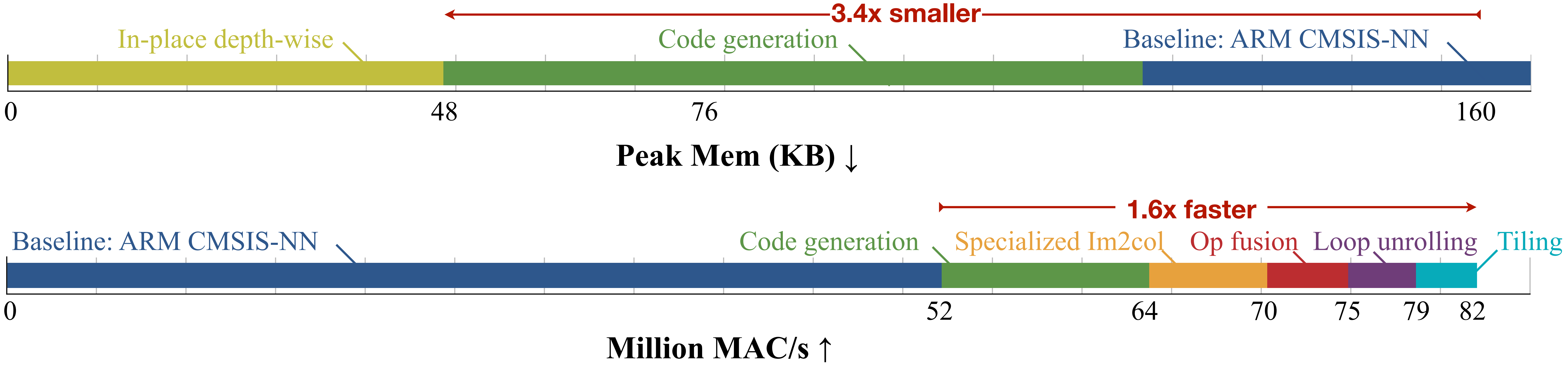
- Tiling loops based on the kernel size and available memory



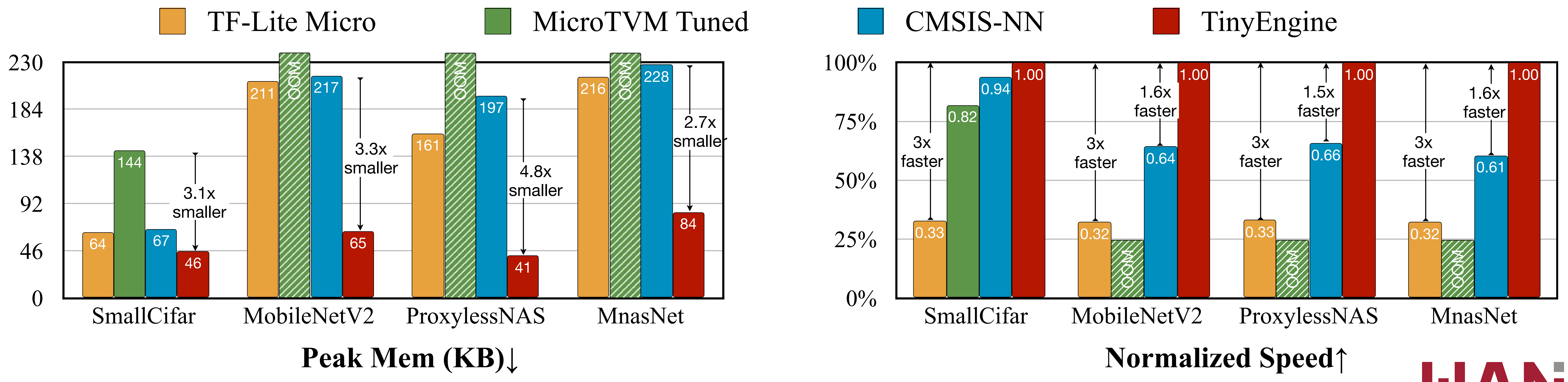
TinyEngine: A Memory-Efficient Inference Library



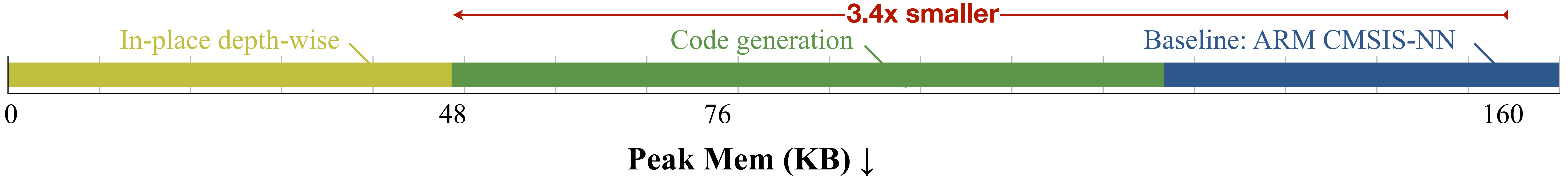
TinyEngine: A Memory-Efficient Inference Library

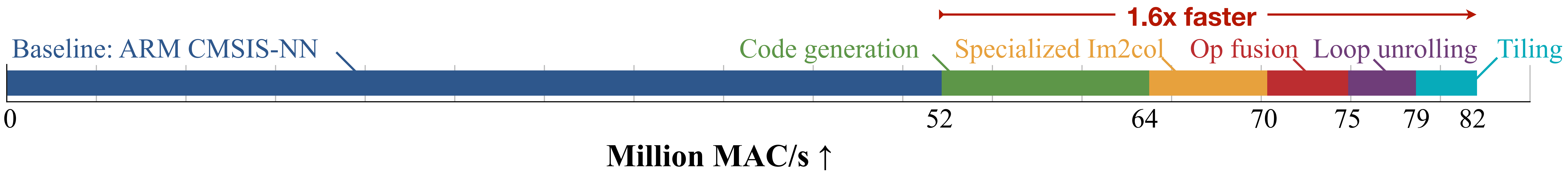


- Consistent improvement on different networks

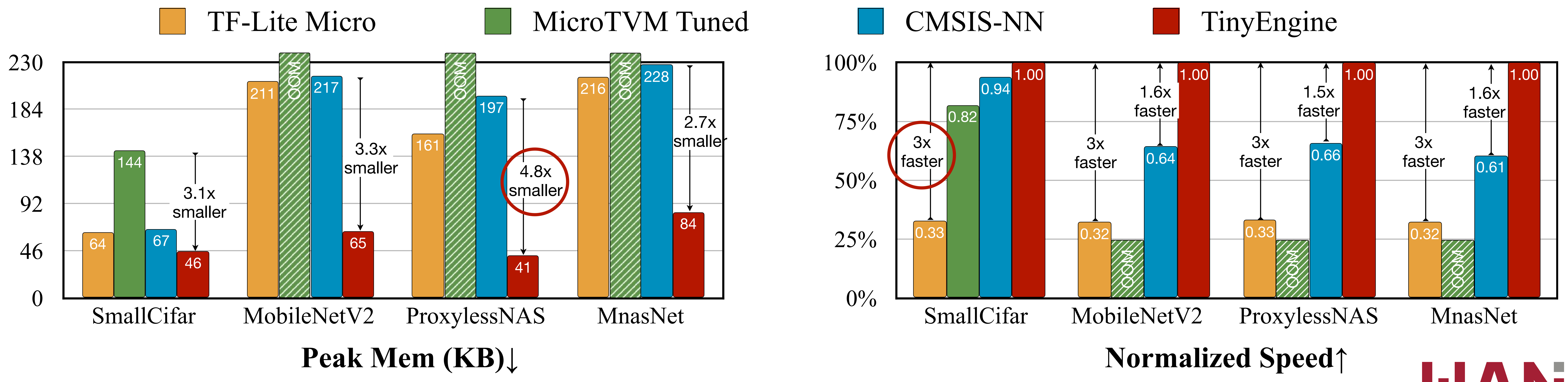


TinyEngine: A Memory-Efficient Inference Library





- Consistent improvement on different networks



Experimental Results

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

Datasets:

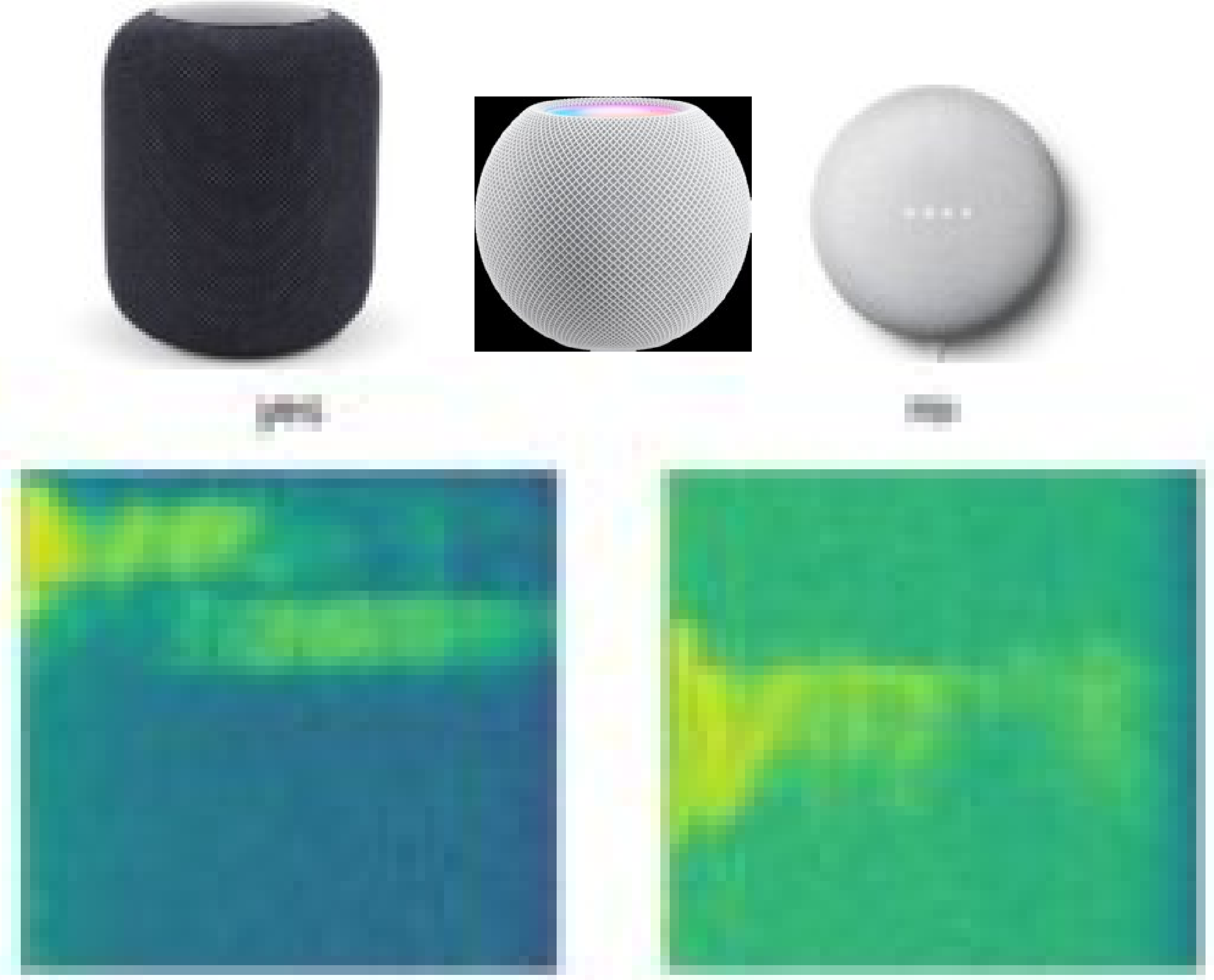
(1) ImageNet-1000

(2) Wake Words

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

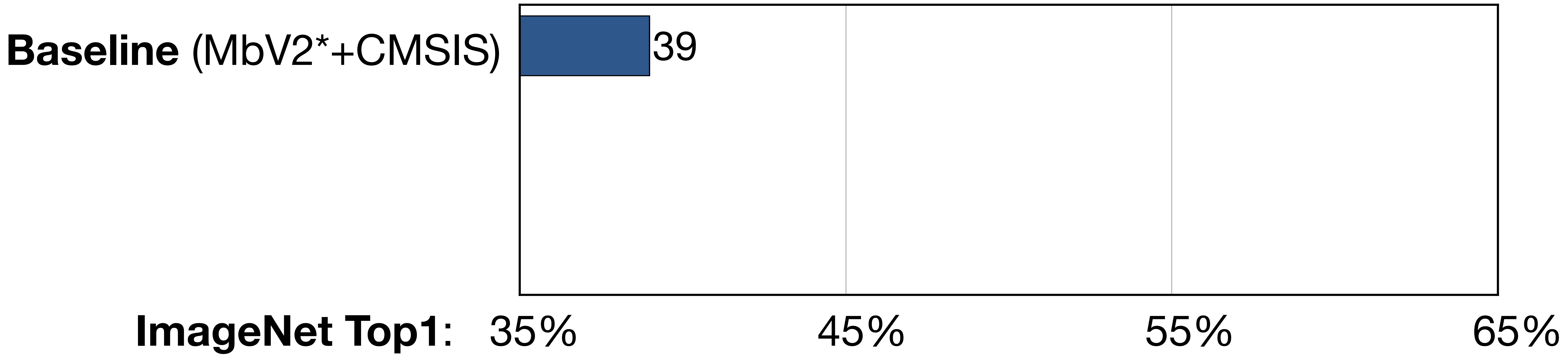
(a) 'Person'

(b) 'Not-person'



System-Algorithm Co-design Gives the Best Results

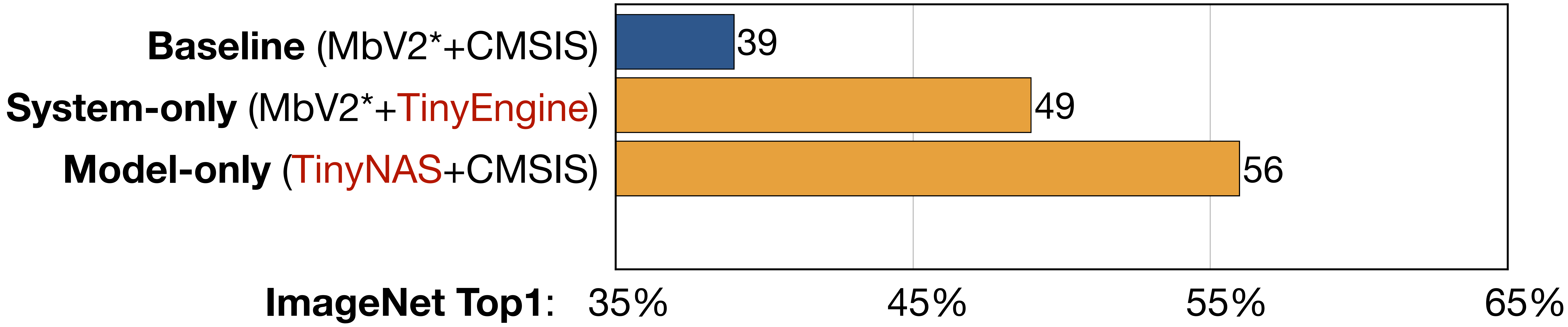
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



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

System-Algorithm Co-design Gives the Best Results

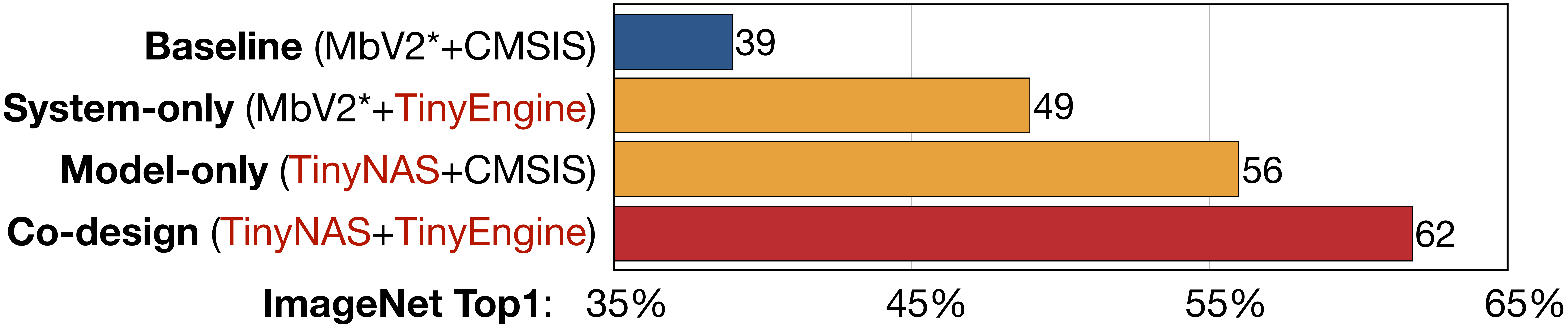
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



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

System-Algorithm Co-design Gives the Best Results

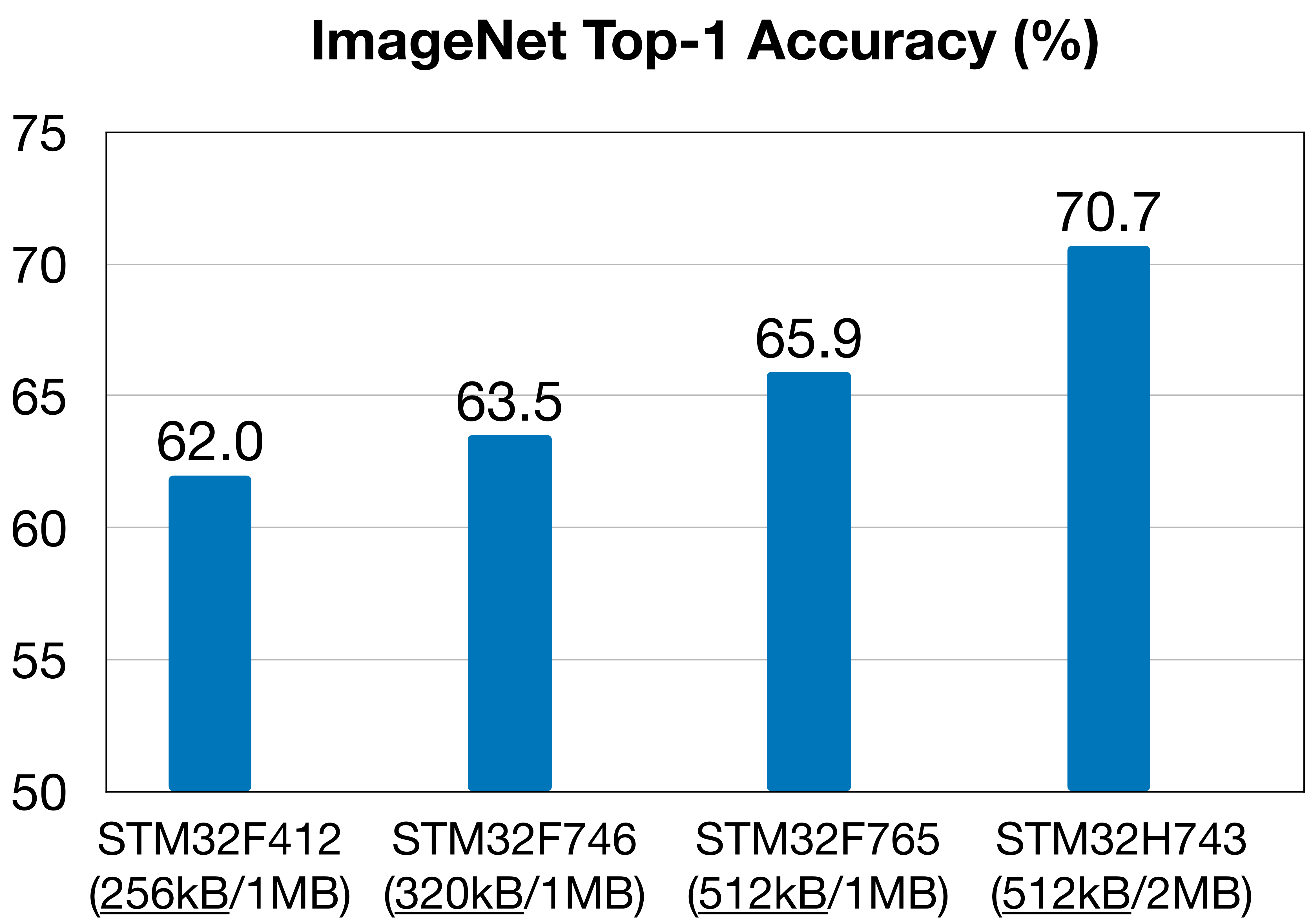
- ImageNet classification on STM32F746 MCU (**320kB SRAM**, **1MB Flash**)



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

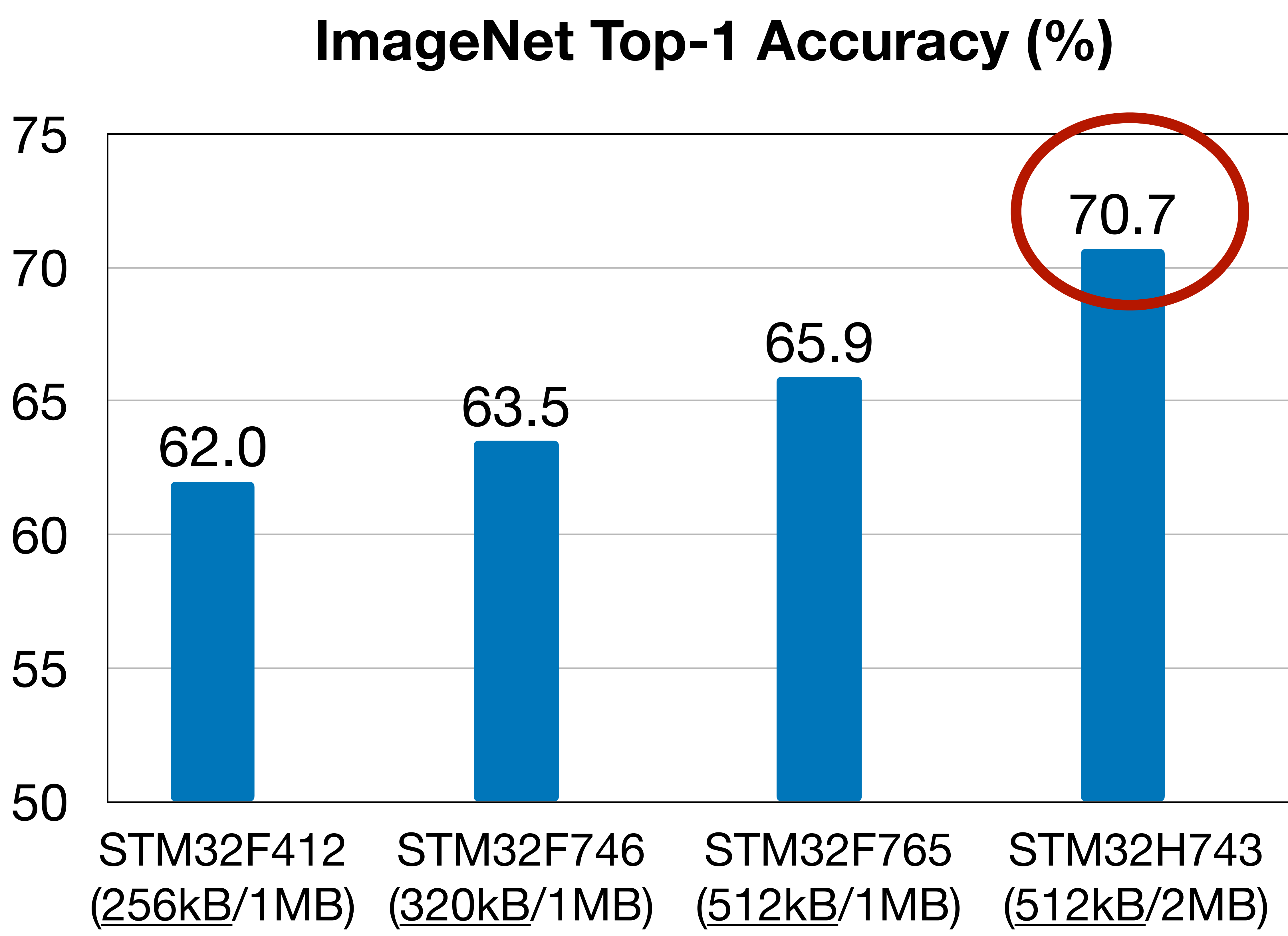
Handling Diverse Hardware

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



Handling Diverse Hardware

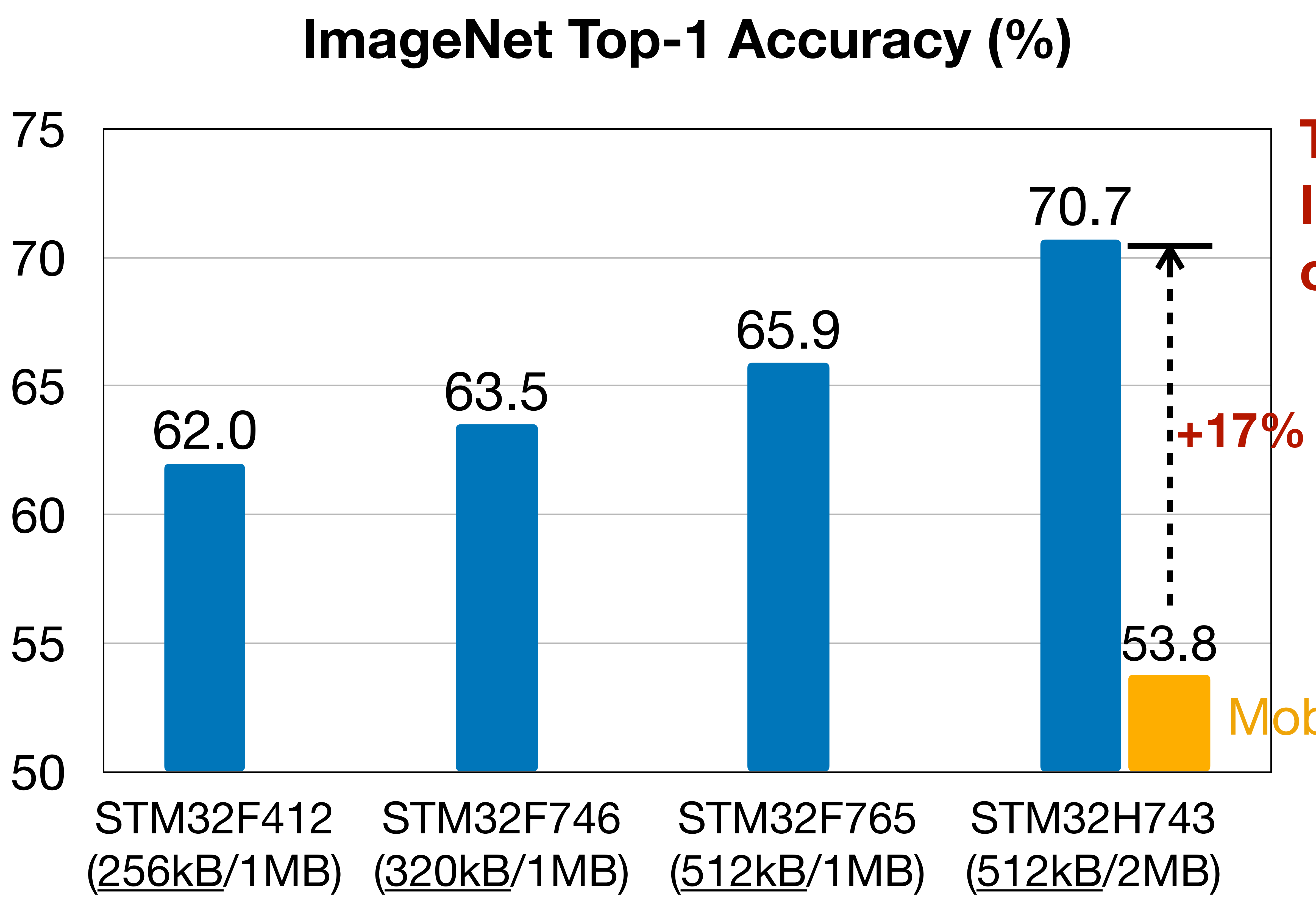
- Specializing models (int4) for different MCUs (SRAM/Flash)



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

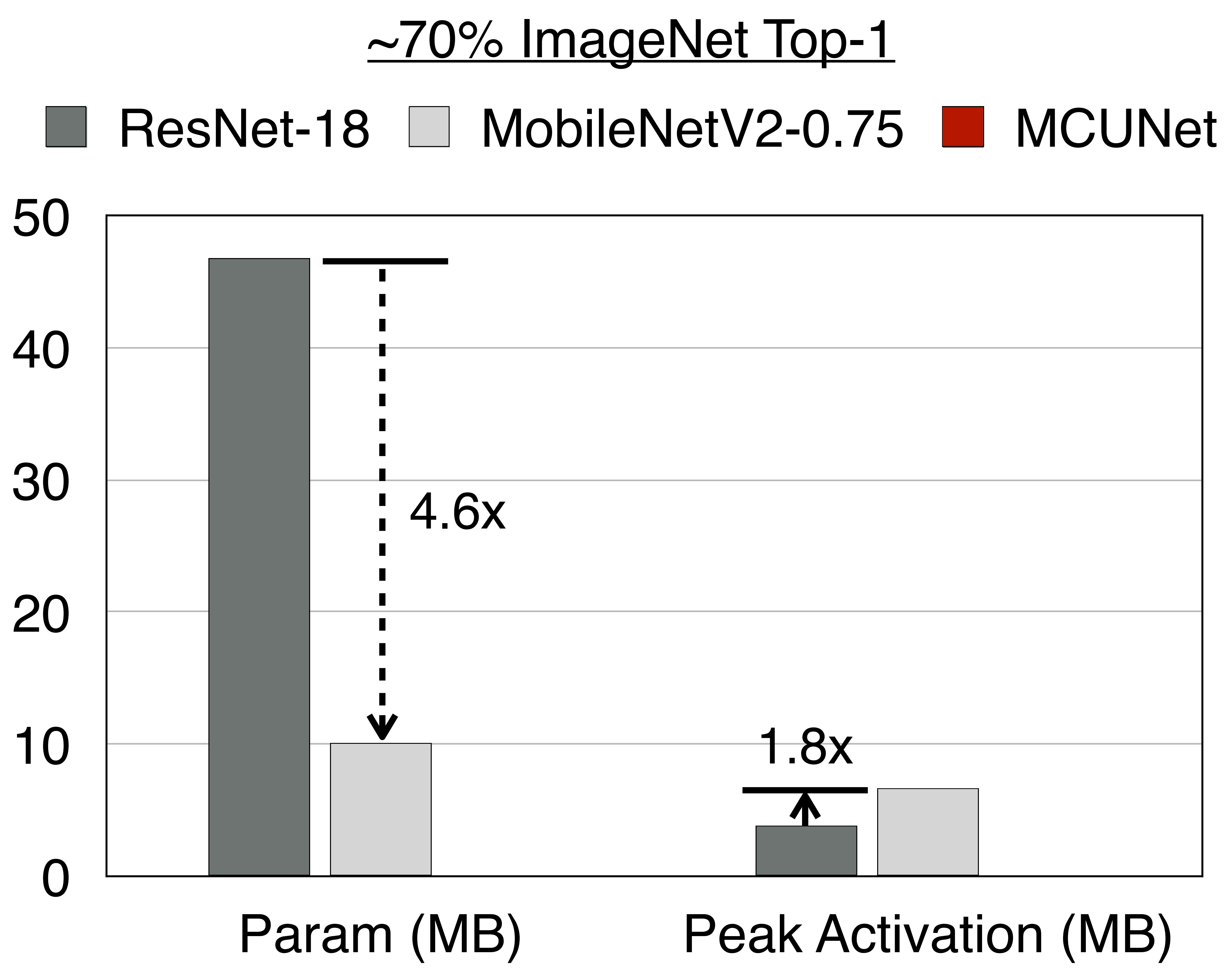
Handling Diverse Hardware

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

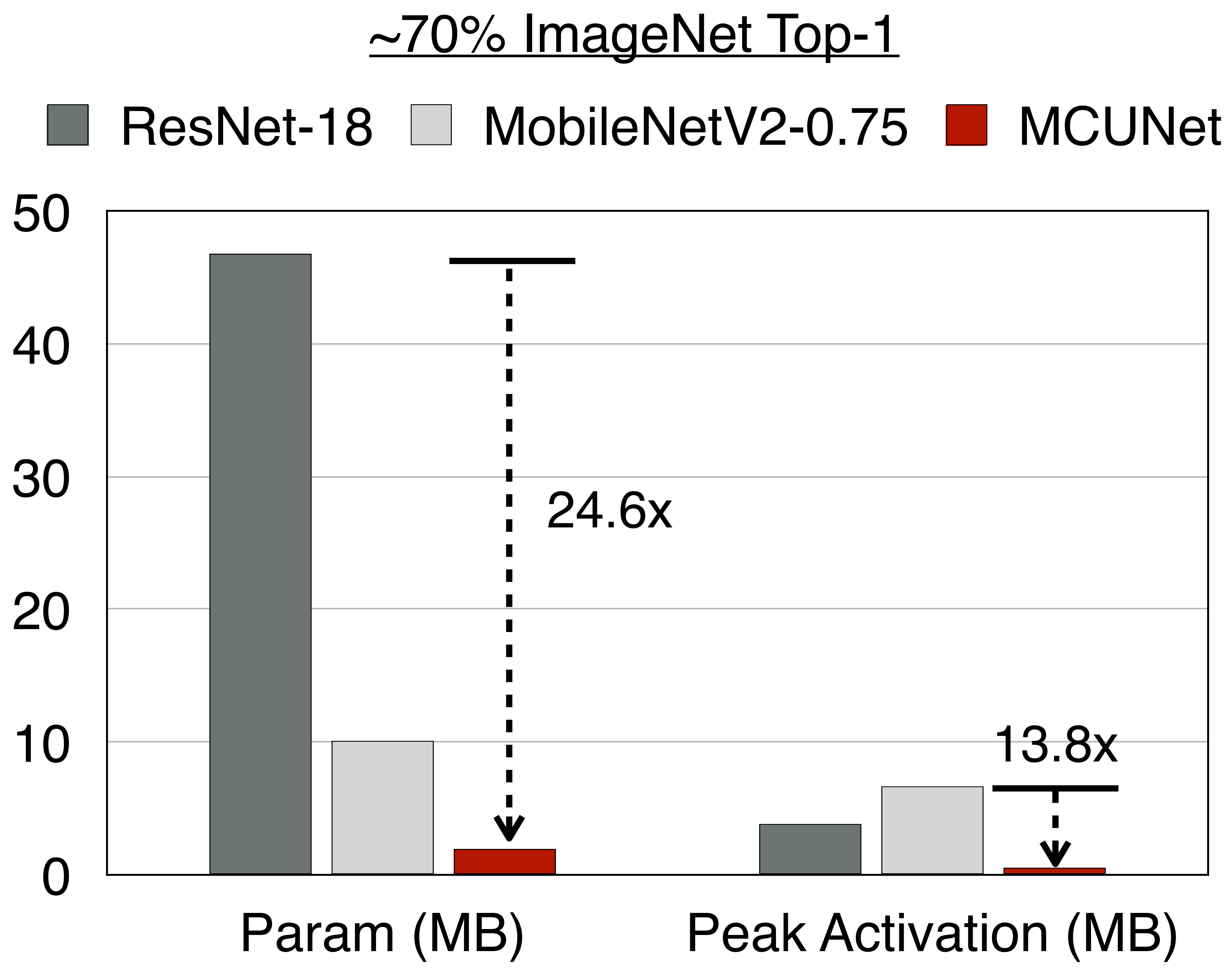


The first to achieve >70% ImageNet accuracy on commercial MCUs

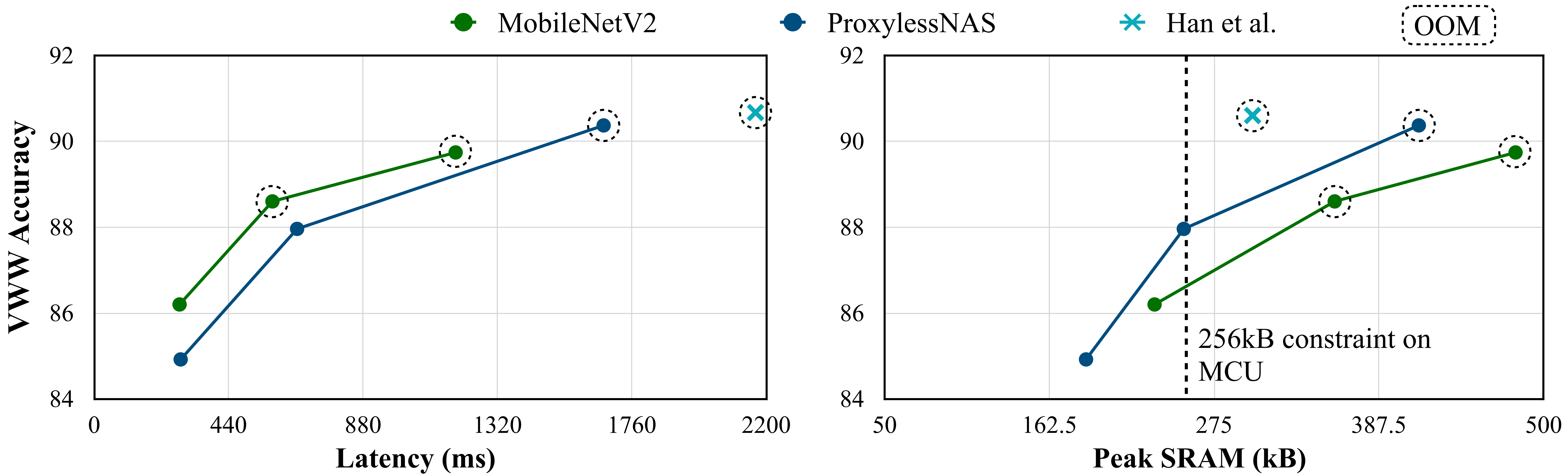
Reduce Both Model Size and Activation Size



Reduce Both Model Size and Activation Size



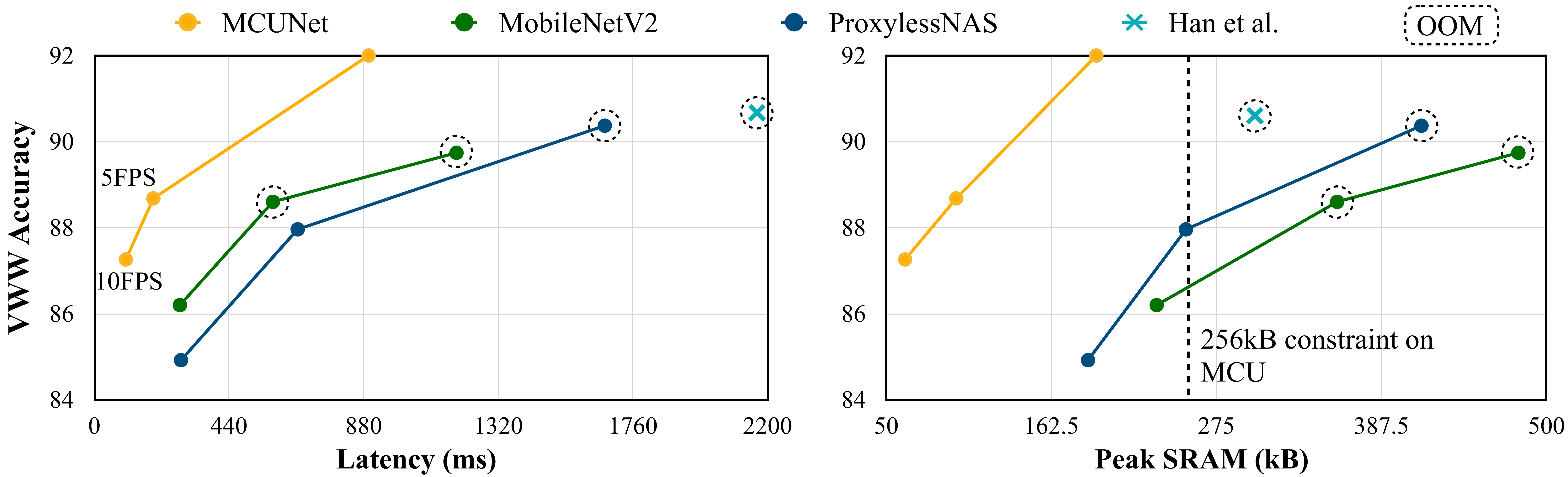
Visual Wake Words (VWW)



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

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

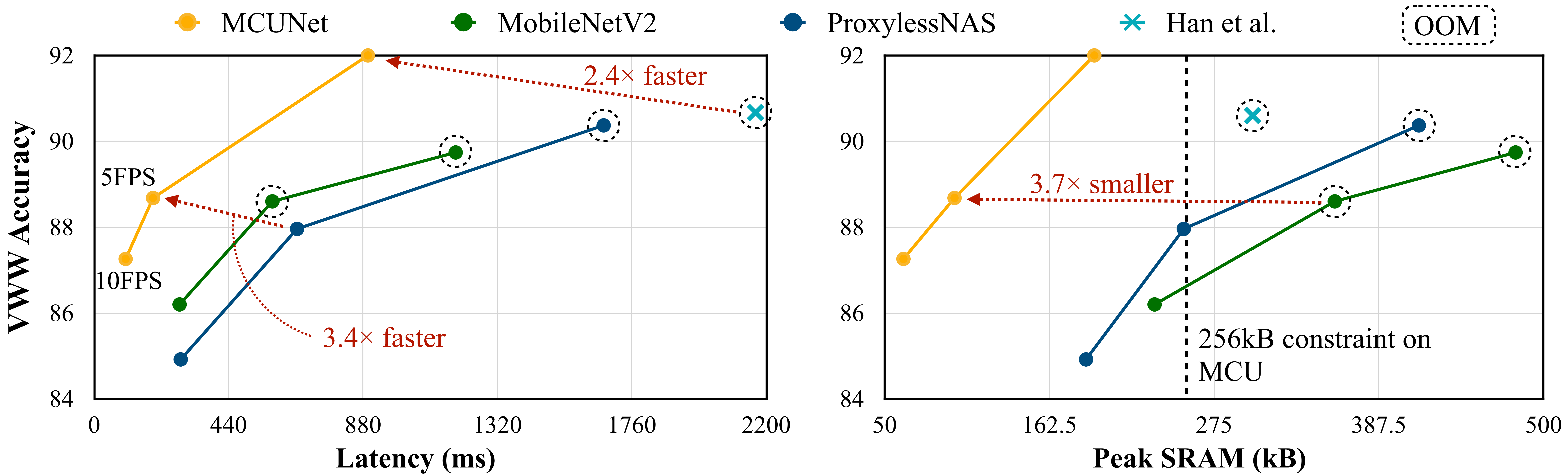
Visual Wake Words (VWW)



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

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

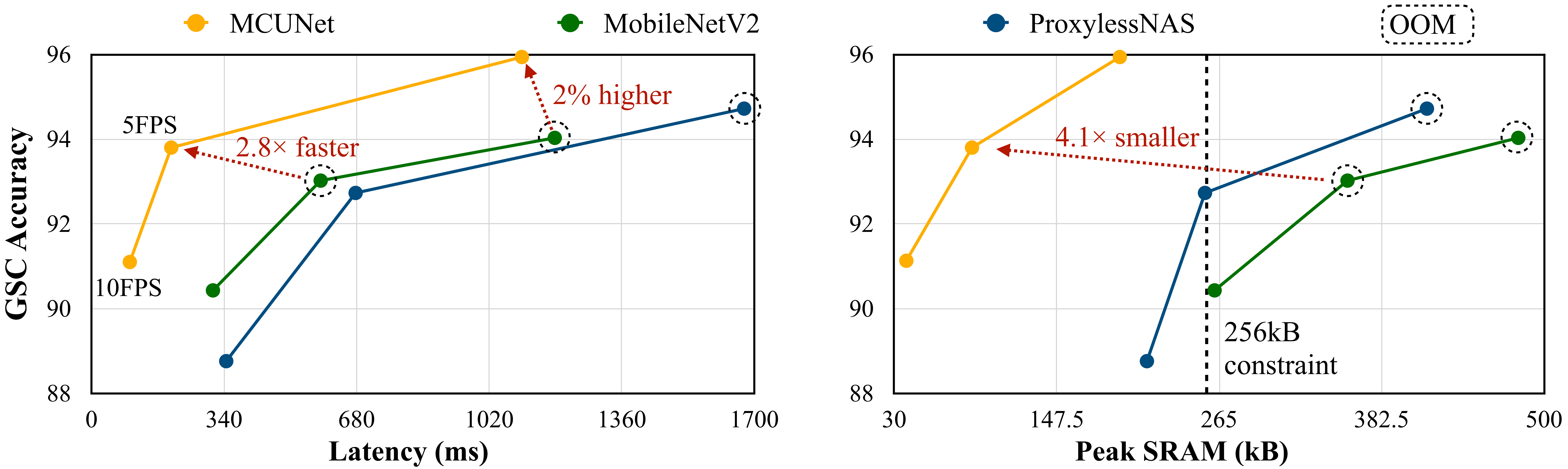
Visual Wake Words (VWW)



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

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

Audio Wake Words (Speech Commands)

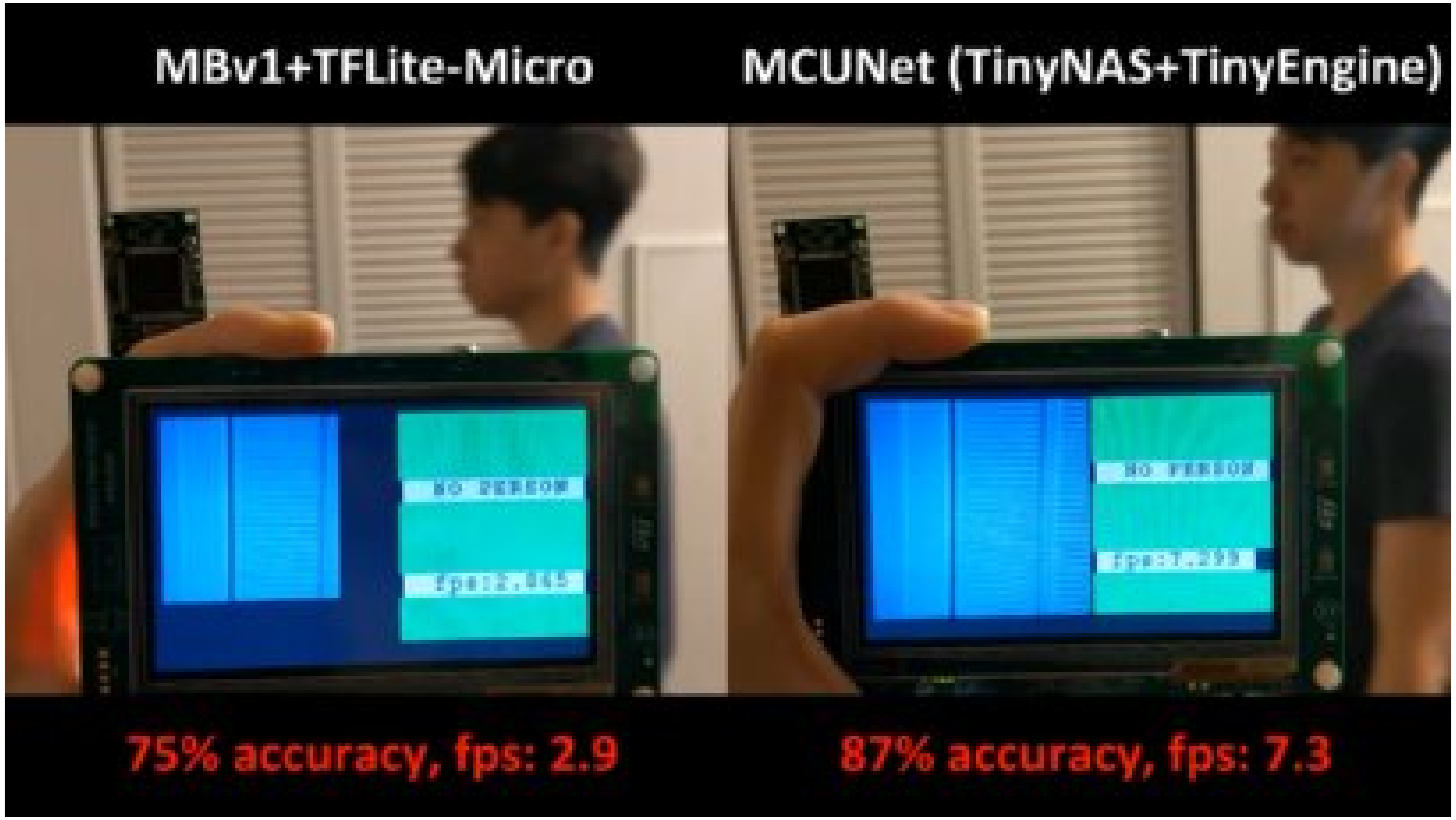


(a) Trade-off: accuracy *vs.* measured latency

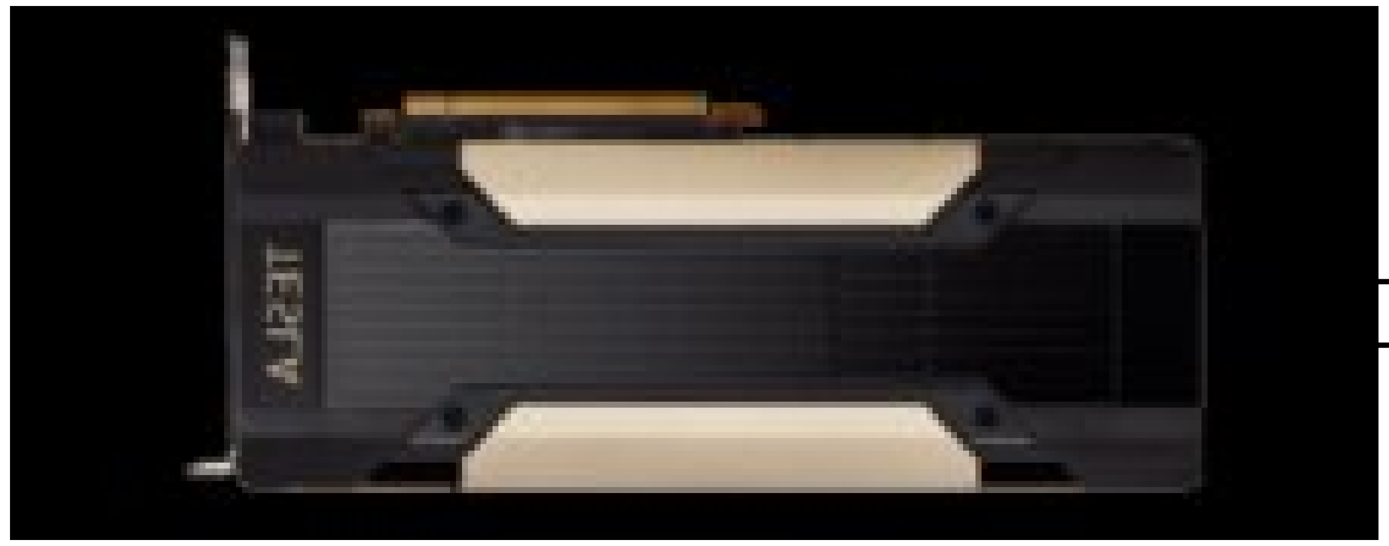
(b) Trade-off: accuracy *vs.* peak memory

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**