北京郵電大學
Beijing University of Posts and Telecommunications

通向泛在学习的系统软件之路
The Way towards Ubiquitous Learning: a System Perspective

Mengwei Xu（徐梦炜）@
CS Dept of BUPT

Being ubiquitous means…

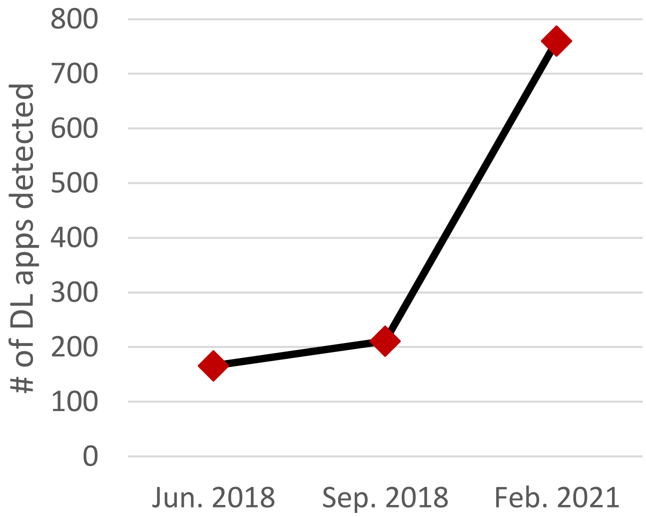
“The most profound techniques are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

- Mark Weiser

Is AI ubiquitous now…?

- **DL apps are increasing rapidly**

Jun. 2018	Sep. 2018	Mar. 2021
	↑27%	↑260%
166/16,500	211/16,500	760/16,500
1.0%	1.3%	4.6%

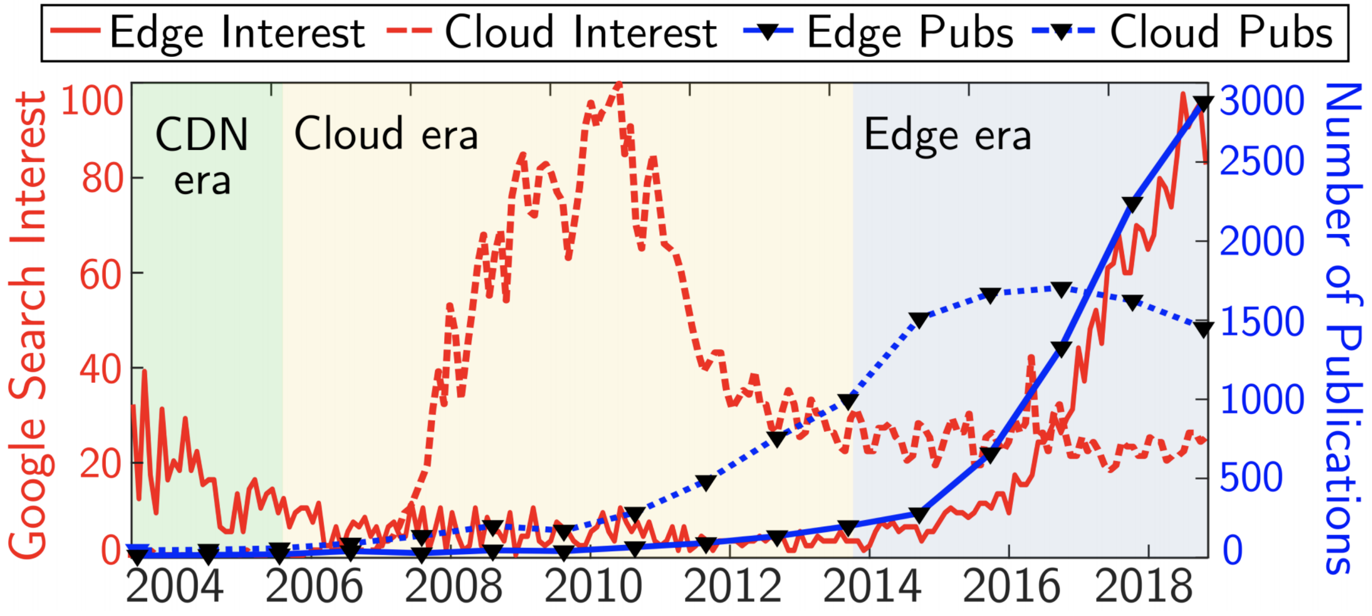


- **DL apps are popular apps**
  - **Contributing to billions of downloads**

[1] Mengwei Xu, et al. "A First Look at Deep Learning Apps on Smartphones" In the Web Conference (WWW) 2019

Some trends

- Edge devices (smartphones, IoTs, etc) are becoming important computing platforms, not just user equipment [1].



Edge is the new field for emerging techniques like AI[2]

- ❑ Preserving privacy
- ❑ Low delay
- ❑ Personalization
- ❑ etc..

[1] Mengwei Xu, et al. “A case for camera-as-a-service”, IEEE Pervasive Computing, 2021.
[2] N. Mohan, et al. "Pruning Edge Research with Latency Shears." HotNets, 2020.

Some trends

- Edge devices (smartphones, IoTs, etc) are becoming important computing platforms, not just user equipment [1].

Framework	Owner	Target Platforms	Release Year
TensorFlow Lite	Google	Android, iOS, Microcontroller	2017
ncnn	Tencent	Android, iOS	2017
MNN	Alibaba	Android, iOS	2019
PaddleLite	Baidu	Android, iOS	2018
MACE	Xiaomi	Android, iOS	2018
MindSpore Lite	Huawei	Android, iOS, LiteOS	2020
SNPE	Qualcomm	Snapdragon CPU, Adreno GPU, Hexagon DSP	2017
PytorchMobile	Facebook	Android, iOS	2019
ComputeLibrary	ARM	Arm Cortex CPU, Arm Mali GPU	2017
MegEngine	Megvii	Android, iOS	2020
tengine	Open AI Lab	Android, iOS	2018
Core ML	Apple	iOS	2017
CMSIS-NN	ARM	Cortex-M Microcontroller	2018

Not enough!

Only inference (static deployment); no training (learning from environments)

面向泛在学习的机器学习系统

中国计算机学会通讯 第17卷 第10期 2021年10月

徐梦炜[1] 黄 康[2] 刘譞哲[3]
[1] 北京邮电大学
[2] 领规科技
[3] 北京大学

关键词：人工智能 终端设备 系统软件

Ubiquitous Learning

The devices can learn from the environments
at anywhere and anytime

- **Autonomous**: on-device transfer learning / personalization / …
- **Cooperative**: federated learning / split learning / …

A review of DL system evolution on cloud/edge

AlexNet, 2012
- With GPU used

Parameter Server, 2014
- Many GPUs

Since then…

TensorFlow (Google), PyTorch (FB), PaddlePaddle (Baidu), Petuum (CMU), MXNet (Apache), DMTK (MS), Horovod, etc.

TF Serving, 2016
- For deployment

TVM, 2018
- Auto Compilation

…

TensorRT (Nvidia), PRETZEL (SNU), Rammer (MSRA), TorchScript (FB), DeepCPU (MS), etc.

A review of DL system evolution on cloud/edge

AlexNet, 2012
- With GPU used

Parameter Server, 2014
- Many GPUs

Since then…

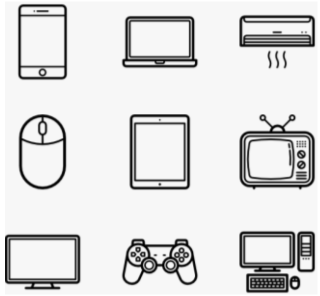
TensorFlow (Google), PyTorch (FB), PaddlePaddle (Baidu), Petuum (CMU), MXNet (Apache), DMTK (MS), Horovod, etc.

TF Serving, 2016
- For deployment

TVM, 2018
- Auto Compilation

…

TensorRT (Nvidia), PRETZEL (SNU), Rammer (MSRA), TorchScript (FB), DeepCPU (MS), etc.



Federated Learning (Google), 2017
- A killer usage for on-device training
- A special case of decentralized learning

MNN (Alibaba), Core ML (Apple), 2019
- Preliminary support: only CPU and a few ML operators & optimizers

LONG WAY TO GO

DL for mobile sensing, 2015

TensorFlow Lite, 2017
- Rich support for heterogeneous processors and ML operators

…

Caffe2 (FB), Core ML (Apple), ncnn (Tencent), Paddle Lite (Baidu), MACE (Xiaomi), SNPE (Qualcomm), etc.

A review of DL system evolution on cloud/edge

AlexNet, 2012
- With GPU used

Parameter Server, 2014
- Many GPUs

Since then...

TensorFlow (Google), PyTorch (FB), PaddlePaddle (Baidu), Petuum (CMU), MXNet (Apache), DMTK (MS), Horovod, etc.

TF Serving, 2016
- For deployment

TVM, 2018
- Auto compilation

Federated Learning (Google), 2017
- A killer usage for on-device training
- A special case of decentralized learning

MNN (Alibaba), Core ML (Apple), 2019
- Preliminary support: only CPU and a few ML operators

TensorFlow Lite, 2017
- Rich support for heterogeneous processors and ML operators

Caffe2 (FB), Core ML (Apple), ncnn (Tencent), Paddle Lite (Baidu), MACE (Xiaomi), SNPE (Qualcomm), etc.

[WWW'21] The first Heterogeneity-aware FL platform

[MobiCom'22] Mandheling: mixed-precision training on SoC

[UbiComp'18] DeepType: The first attempt of on-device training for input prediction personalization

[MobiSys'22] Melon: breaking memory wall for on-device training

More to come...

Many attemps at on-device inference:
[WWW'22] DL lib Benchmark
[USENIX ATC'21] ZC2
[MobiSys'20] Elf
[WWW'19] An empirical study
[TMC'19] DeepWear
[MobiCom'18] DeepCache
...

Key incentives to UL

Data Privacy

Federated learning, differential privacy, homomorphic encryption, secure aggregation, etc..

Amortized training cost

On-device transfer learning, personalized model, etc..

Key research questions in UL

1. Training data is limited, non-IID, or even not labelled
    - Model accuracy heavily relies on data!

2. Devices have constrained hardware resources
    - Training a ML model is notoriously resource-hungry!

2 Key research questions in UL

1. Training data is limited, non-IID, or even not labelled
    - Model accuracy heavily relies on data!

2. Devices have constrained hardware resources
    - Training a ML model is notoriously resource-hungry!

We need a system-algorithm co-design

Outline

- **A measurement study of on-device training**
  - **EMDL’20**
- **Memory optimization of on-device training**
  - **MobiSys’21**
- **Mixed-precision training with on-chip offloading**
  - **MobiCom’22**

Outline

- **A measurement study of on-device training**
  - **EMDL'20**
- Memory optimization of on-device training
  - MobiSys'21
- Mixed-precision training with on-chip offloading
  - MobiCom'22

On-device training: a measurement study

- Target library: MNN[1] by Alibaba
- 6 Android devices
- 5 classic CNN models
  - LeNet, AlexNet, MobileNetv2, SqueezeNet, GoogLeNet
- CPU by default

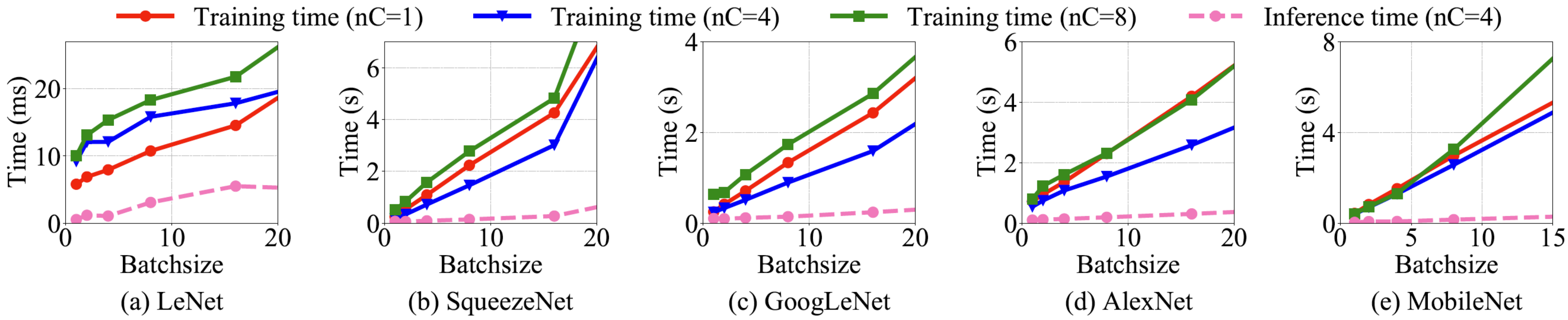
Testing platform	Training library	Training time (ms)		
		BS = 1	BS=2	BS=4
Samsung Note 10	MNN	516	812	1365
	DL4J	3,032	6,129	OOM
RPI 3B+	MNN	6698	10,651	OOM
	TensorFlow	10,468	14,157	27,574
	PyTorch	48,274	79,097	OOM

Device	Specifications	Yr.
Redmi Note 9 Pro	Snapdragon 720G, 6GB RAM	2020
Xiaomi MI 9	Snapdragon 855, 6GB RAM	2019
Huawei Mate 30	Kirin 990, 8GB RAM	2019
Meizu 16T	Snapdragon 855, 6GB RAM	2019
Samsung S8+	Snapdragon 835, 6GB RAM	2017
Huawei Honor 8	Kirin 950, 3GB RAM	2016

[1] https://github.com/alibaba/MNN

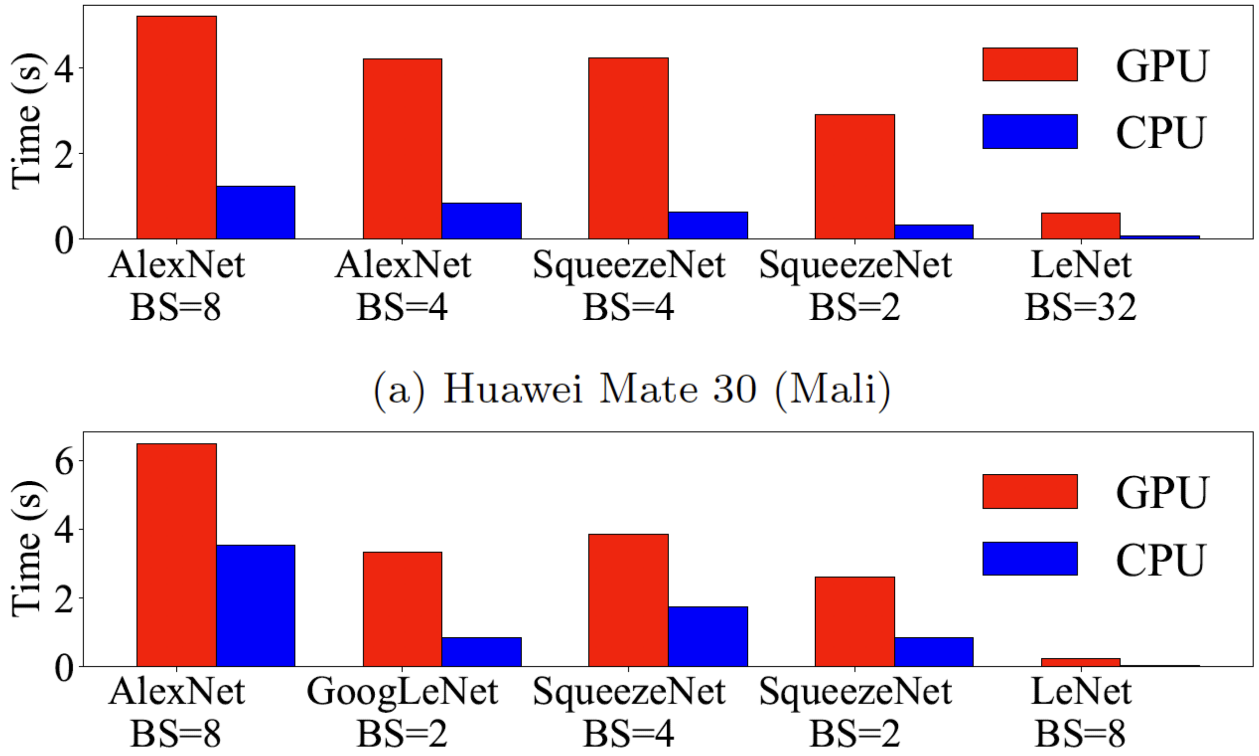
Training time

- Training takes much more time than inference
  - Up to 17.8x gap, much larger than the FLOPs-gap



Training time

- Training takes much more time than inference
- GPU cannot speedup



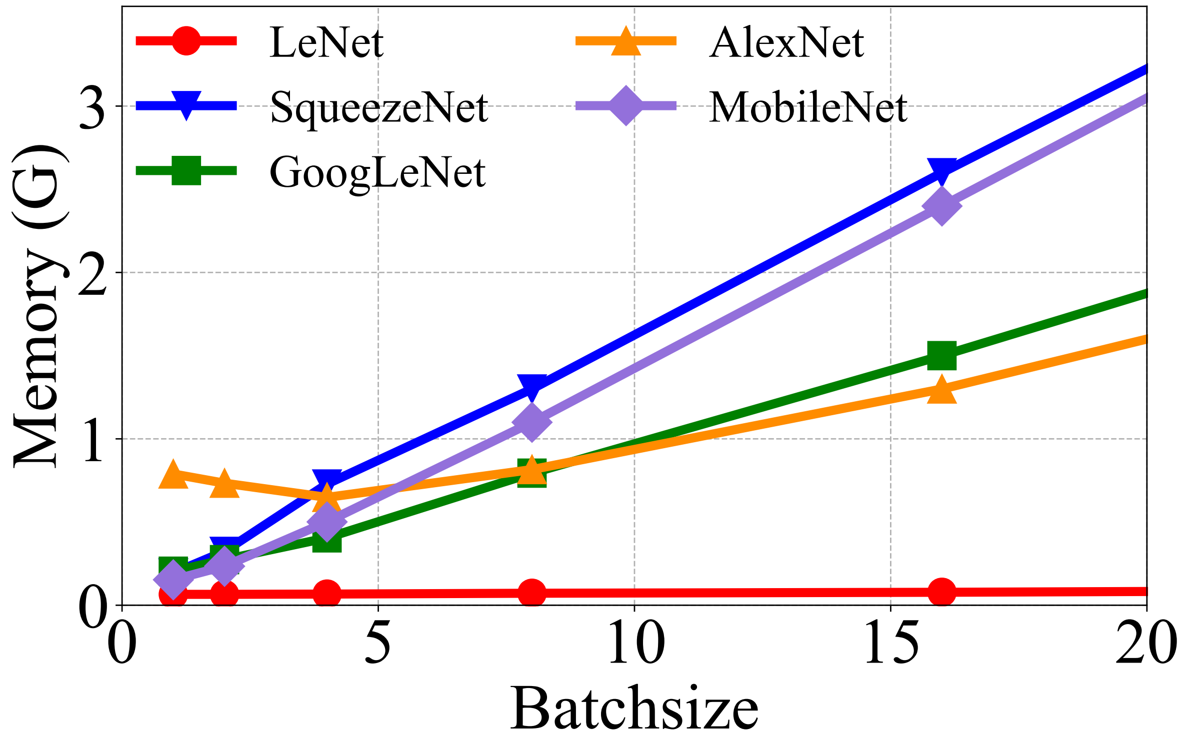
Training time

- Training takes much more time than inference
- GPU cannot speedup

- Why?
  - The training support of MNN is still at very preliminary stage
  - Training is far more complex than inference: much more operators, dynamic weights update, variable batch size, etc…

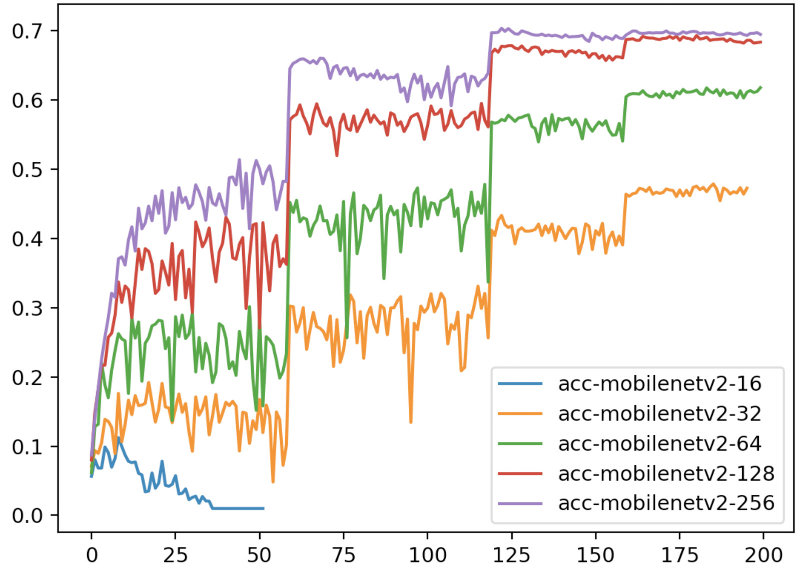
Memory footprint

- Training is very memory-intensive
  - 16-32 is typically the max batch size supported by a high-end device (6~8 GBs)

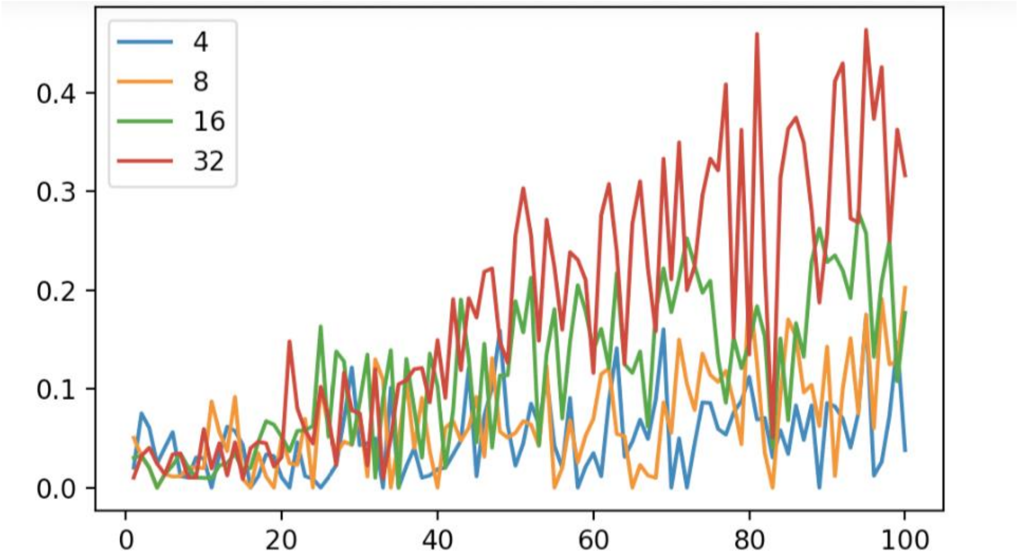


Memory footprint

- Training is very memory-intensive
  - 16-32 is typically the max batch size supported by a high-end device
  - Enough for a good convergence? NO!



Single-machine setting (MobileNet-v2)



Federated setting (MobileNet-v2)

[1] Smith, Samuel L., et al. "Don't decay the learning rate, increase the batch size." *arXiv preprint arXiv:1711.00489* (2017).

Memory footprint

- Training is very memory-intensive
  - 16-32 is typically the max batch size supported by a high-end device
  - Enough for a good convergence? NO!

A ”clean” environment.

In practice? NO!

Outline

- A measurement study of on-device training
  - EMDL'20
- **Memory optimization of on-device training**
  - **MobiSys'21**
- Mixed-precision training with on-chip offloading
  - MobiCom'22

Design goals and principles

- Goal: supporting larger batch size training with given upper bound of peak memory usage

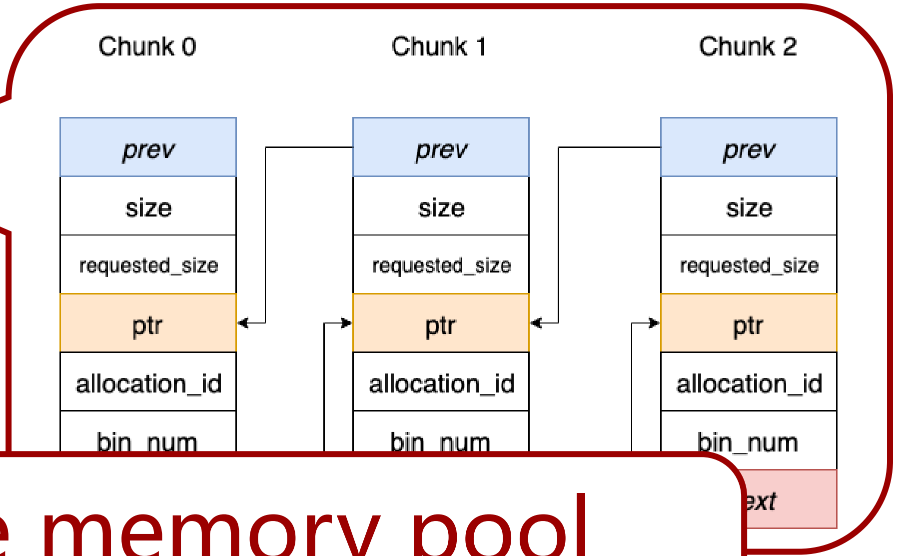
- **Borrowed wisdoms**
    - *Model & gradients compression*
    - *Host-device memory swapping*
    - *Splitting mini-batch to micro-batch*
    - *Activation recomputation*

Design goals and principles

- Goal: supporting larger batch size training with given upper bound of peak memory usage

Challenge#1: efficient memory management

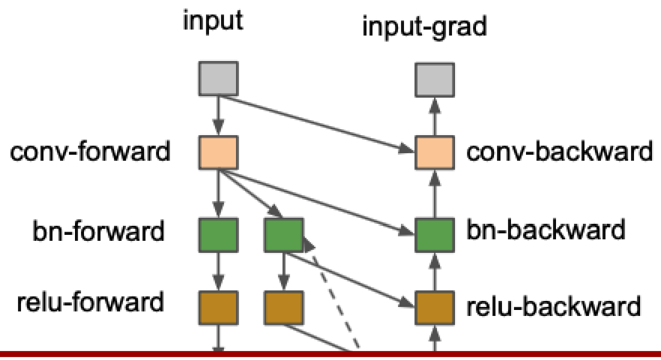
Memory pool is widely adopted – severe fragmentation

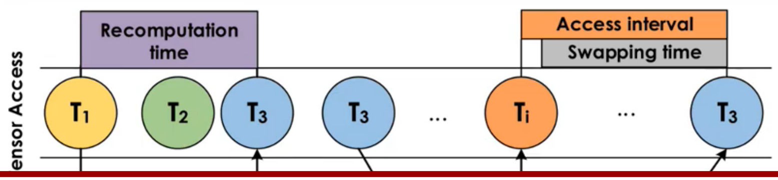


How to manage memory pool efficiently for DNN training?

Challenge#2: efficient recomputation

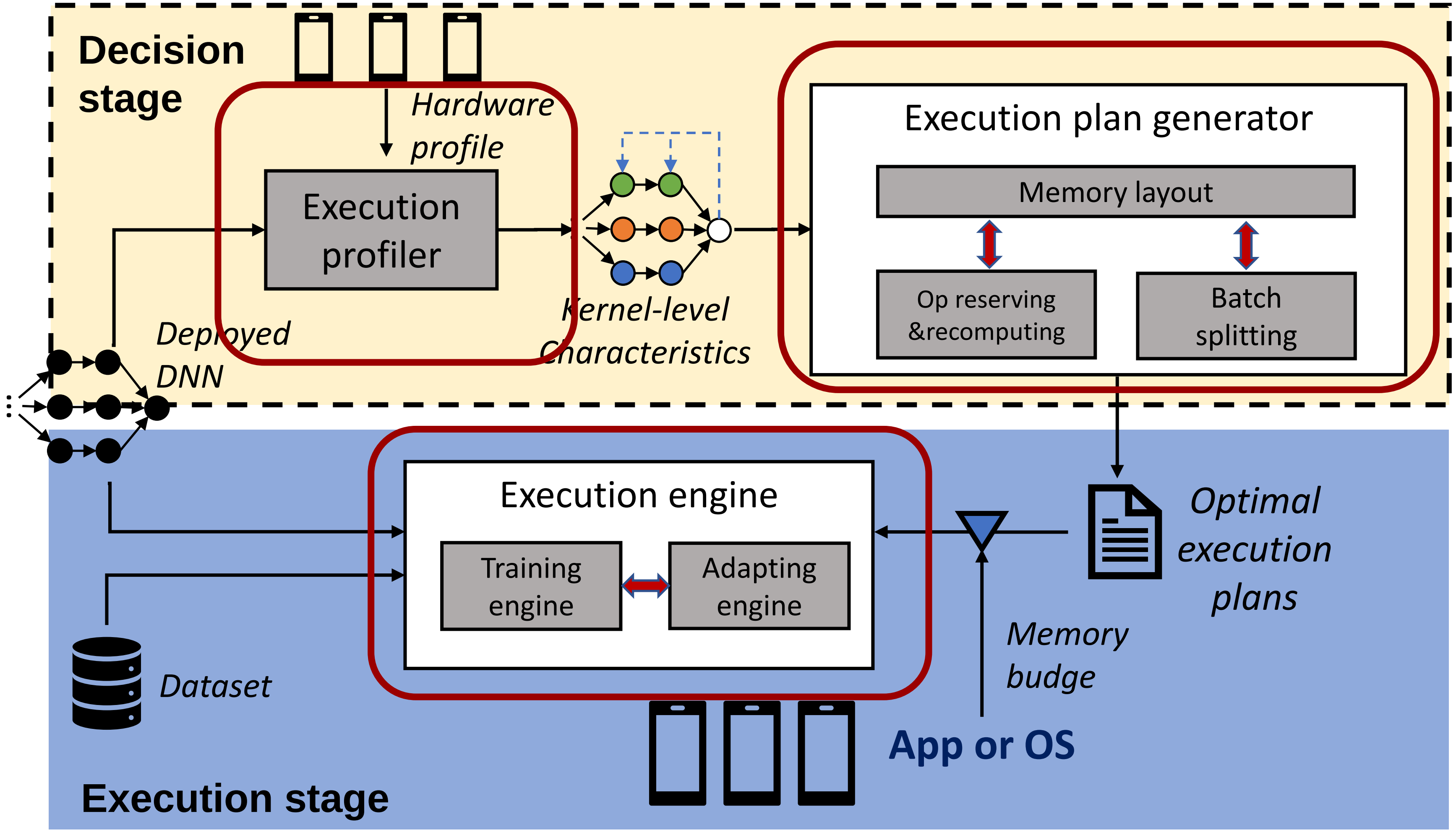
Current recomputation ignores impact of memory pool





How to recompute efficiently based on DNN training specific memory pool?

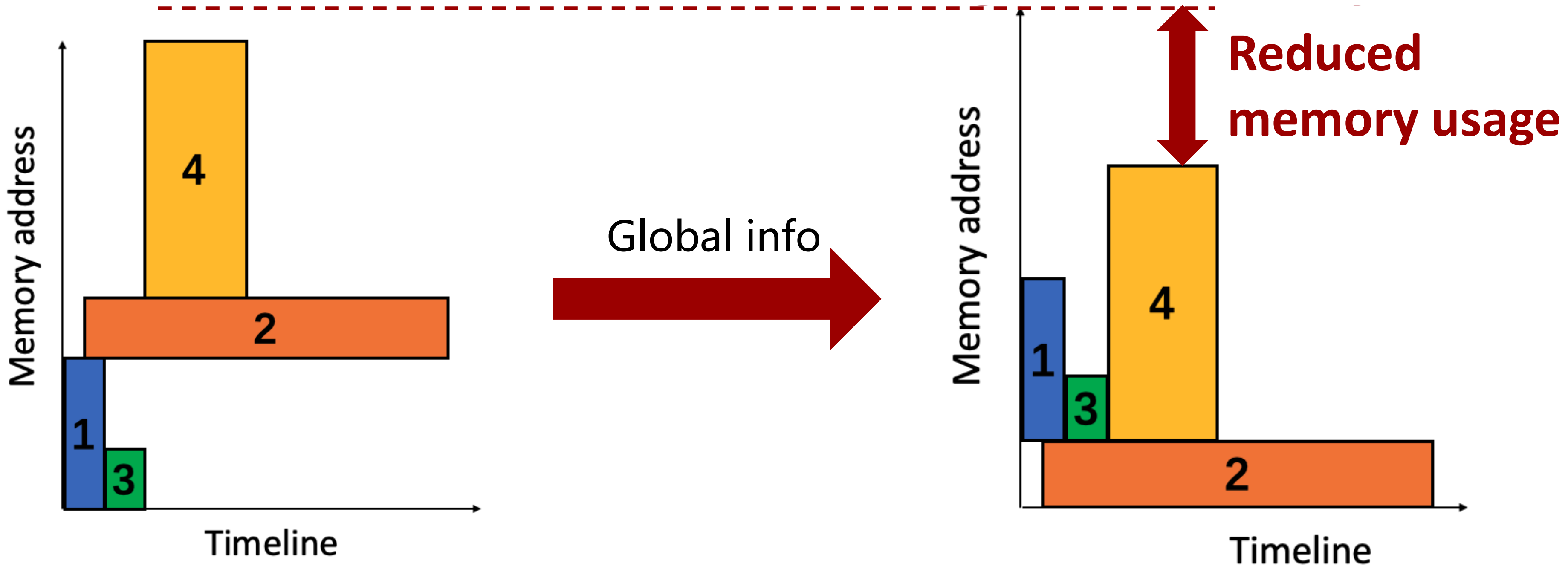
Melon: design overview



Tensor lifetime-aware memory pool

Heuristics

- Memory access pattern of DNN training is fixed
- Tensors allocated earlier are released later



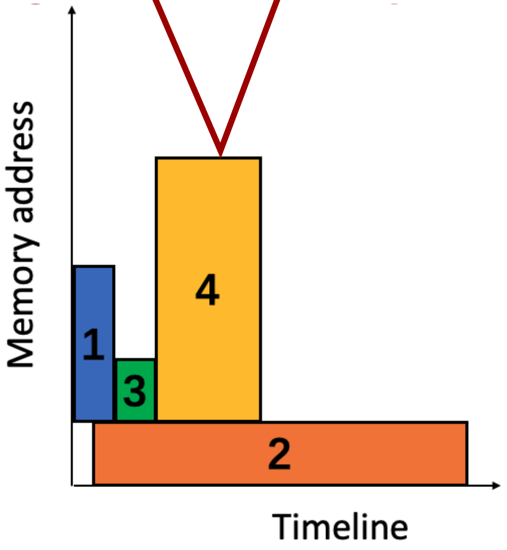
(a) On-demand memory pool **(b) Improved memory pool**

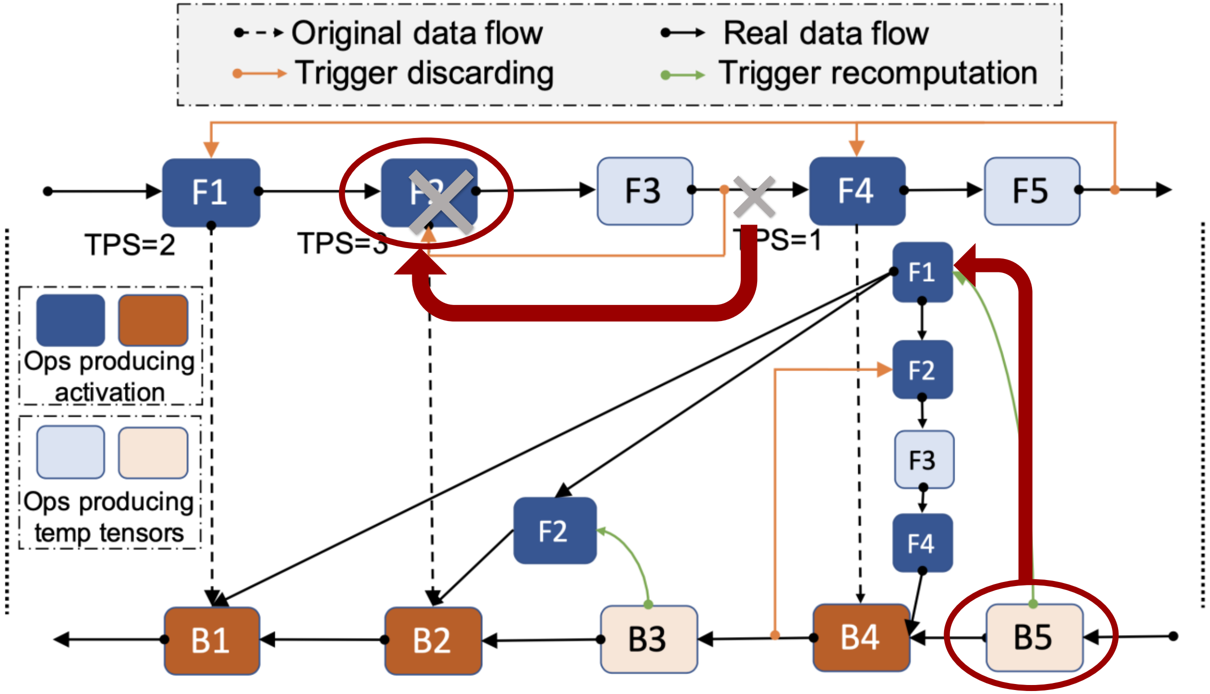
Memory-calibrated progressive recomputation

Recomputation mechanism

- Evict tensor when exceeding memory budget
- Recompute tensor when it is not appeared

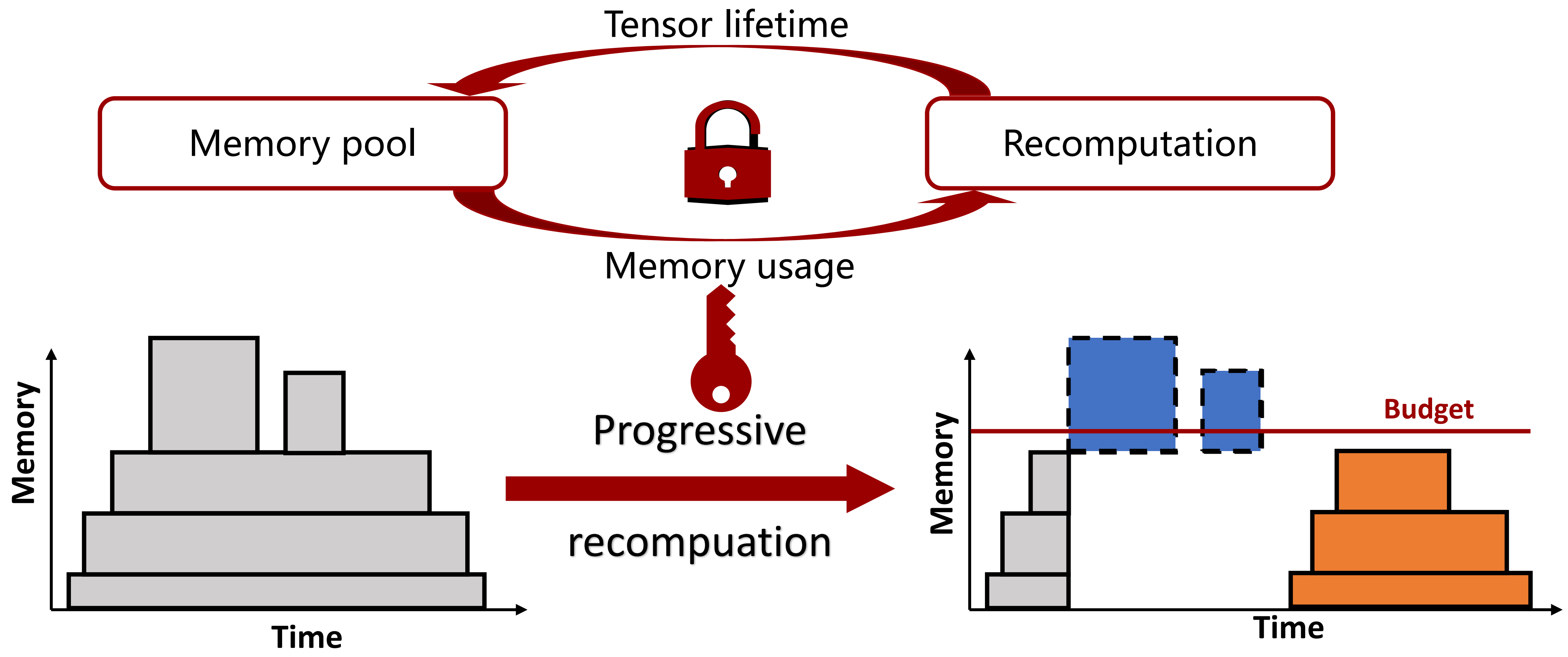
$$TPS = \frac{\boxed{TensorSize * FreedLifetime}}{RecomputationTime}$$





Memory-calibrated progressive recomputation

Take memory pool into consideration



Evaluation

Artifact Available Artifact Evaluated-Functional Results Reproduced

The only paper with 3 AE badges in MobiSys'22

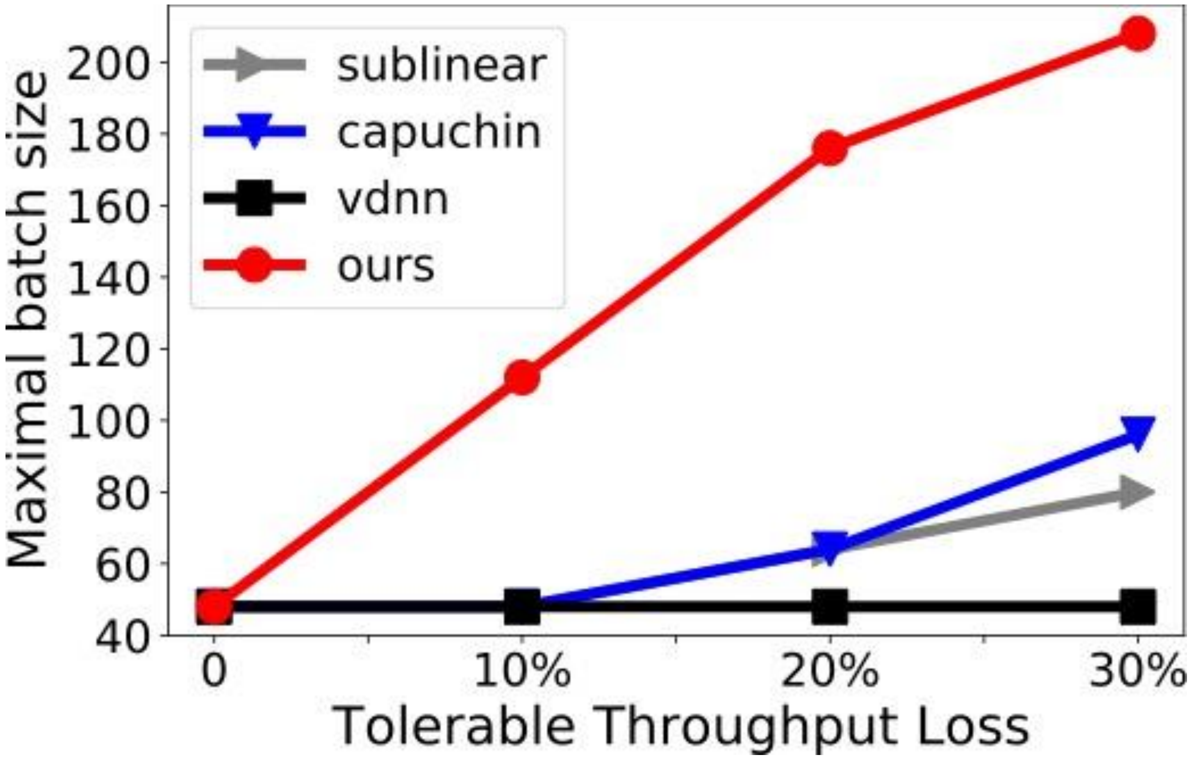
- Implemented atop MNN.
- 4 CNN models and 3 Android devices.

Device	SoC	Memory	Model	Params	FLOPs
Samsung Note10	SD 855	8 GB	MobileNetV1	3.3M	45.5M
Vivo IQOO Neo3	SD 865	6 GB	MobileNetV2	2.4M	67.6M
Redmi Note9 Pro	SD 720	6 GB	SqueezeNet	0.8M	34.4M
Redmi Note8	SD 655	4 GB	ResNet50	23.8M	1336.3M

- Baselines:
  - *Ideal*: the upper bound
  - [Micro'16] *vDNN*: memory swapping
  - [arxiv'16] *Sublinear*: recompute
  - [ASPLOS'20] *Capuchin*: recompute + swapping

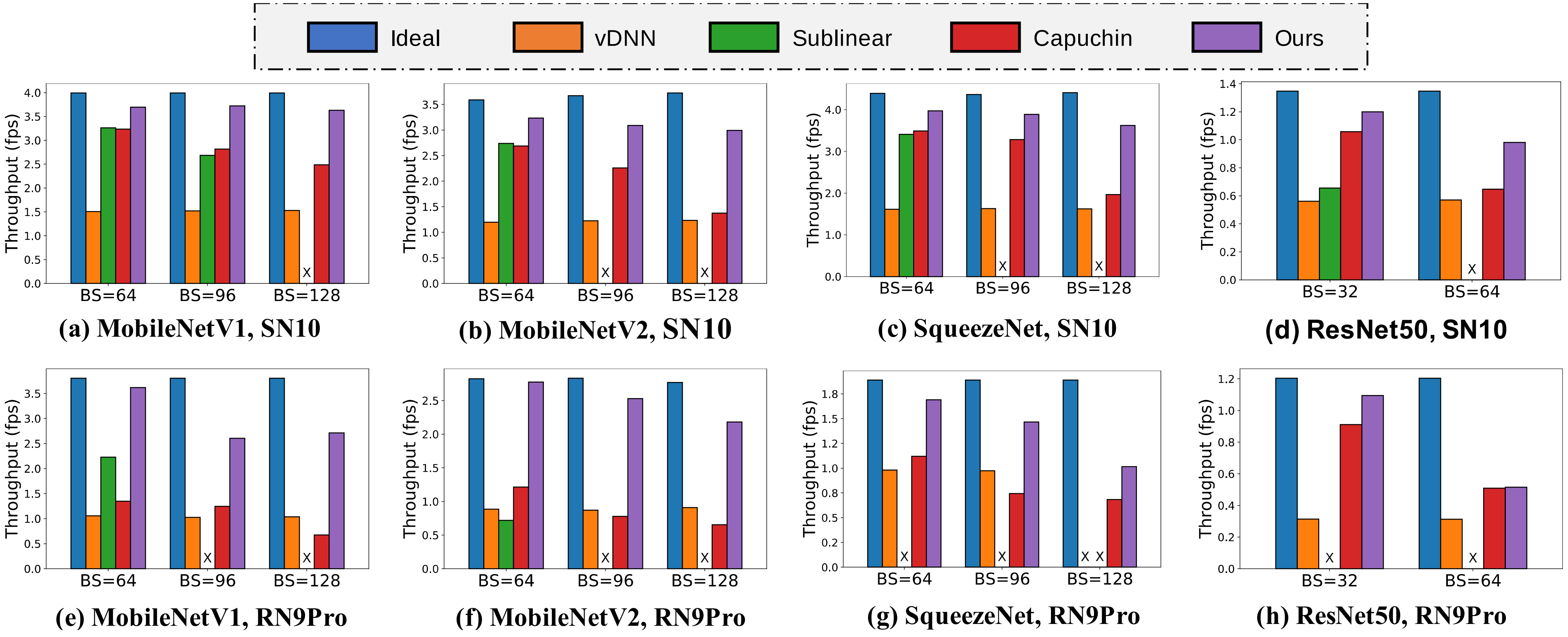
Highlighted results

- Our system supports much larger batch size



Highlighted results

- Our system incurs much less performance loss

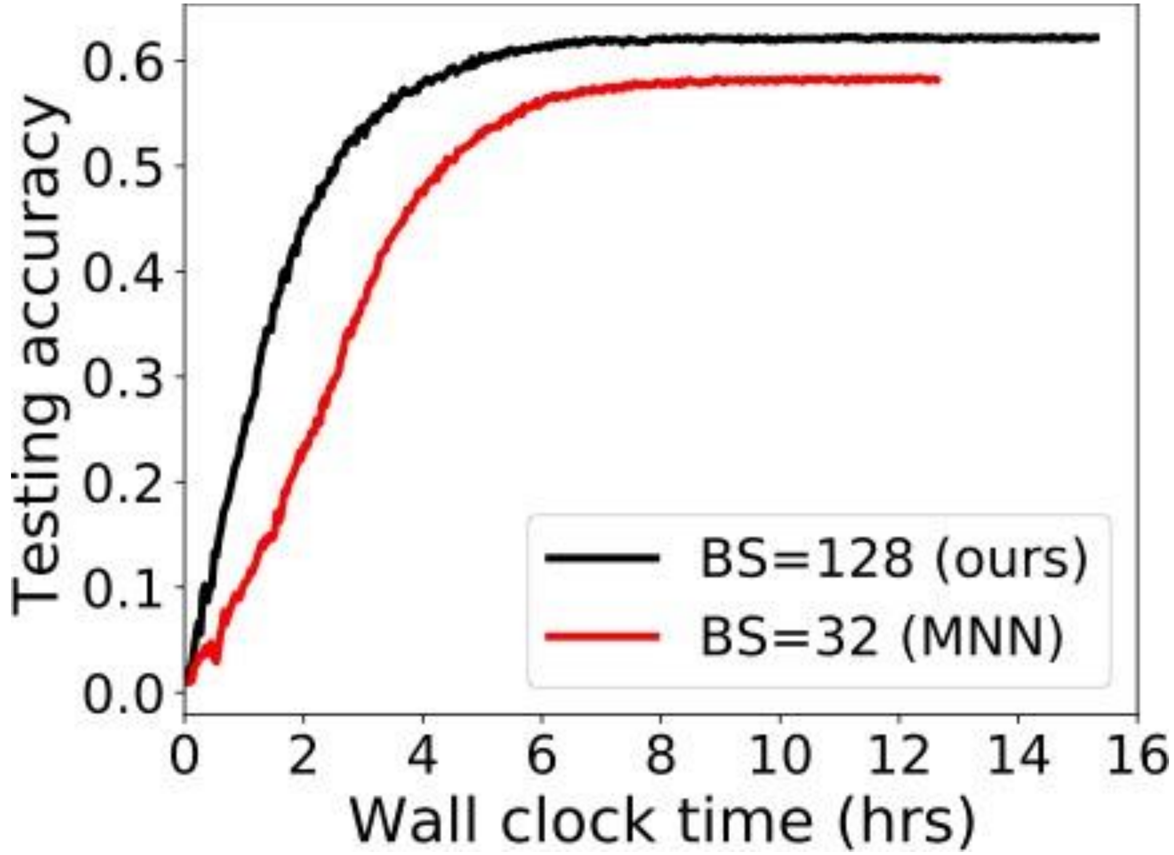


(a) MobileNetV1, SN10 (b) MobileNetV2, SN10 (c) SqueezeNet, SN10 (d) ResNet50, SN10

(e) MobileNetV1, RN9Pro (f) MobileNetV2, RN9Pro (g) SqueezeNet, RN9Pro (h) ResNet50, RN9Pro

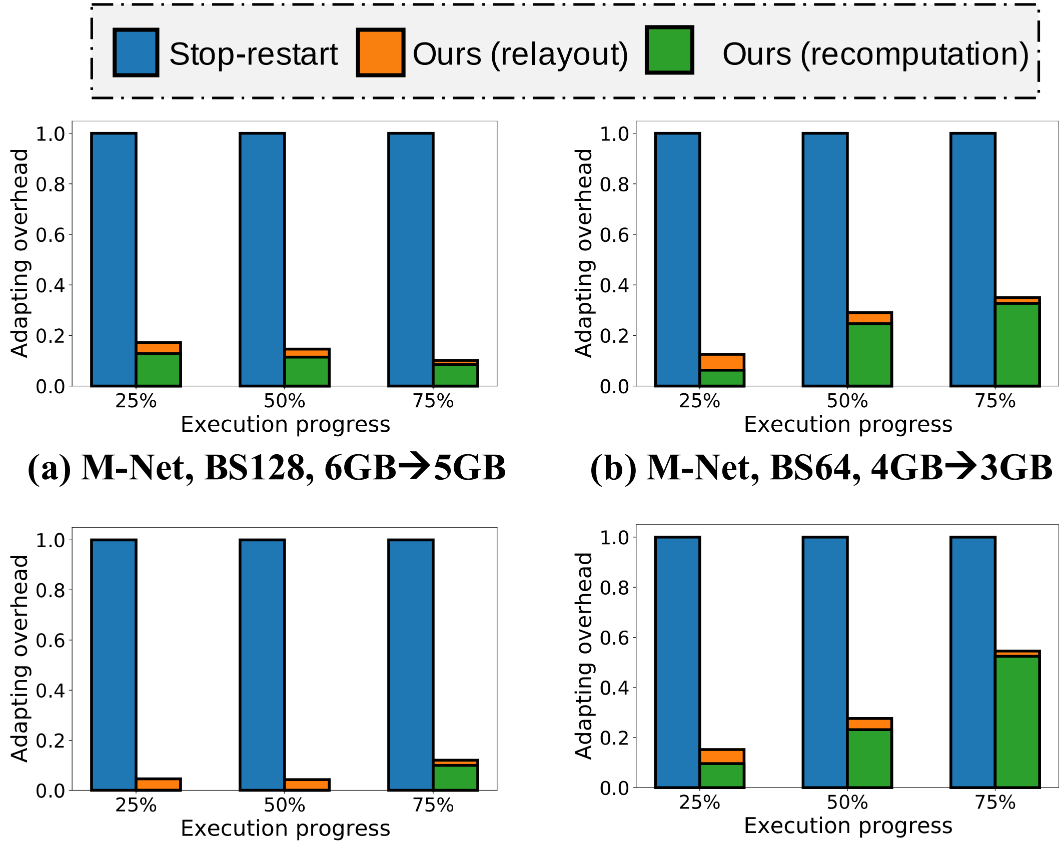
Highlighted results

- Our system improves federated learning from end to end



Highlighted results

- Our system incurs much less overhead during memory adapting



(a) M-Net, BS128, 6GB→5GB (b) M-Net, BS64, 4GB→3GB

(c) S-Net, BS128, 6GB→5GB (d) S-Net, BS64, 4GB→3GB

Outline

- A measurement study of on-device training
  - EMDL'20
- Memory optimization of on-device training
  - MobiSys'21
- **Mixed-precision training with on-chip offloading**
  - **MobiCom'22**

Motivation

- Mixed-precision training is emerging
  - INT8, INT16, FP16, etc…
- Mobile DSP is both ubiquitous and powerful
  - vs. CPU/GPU/NPU
  - Good at integer-based processing

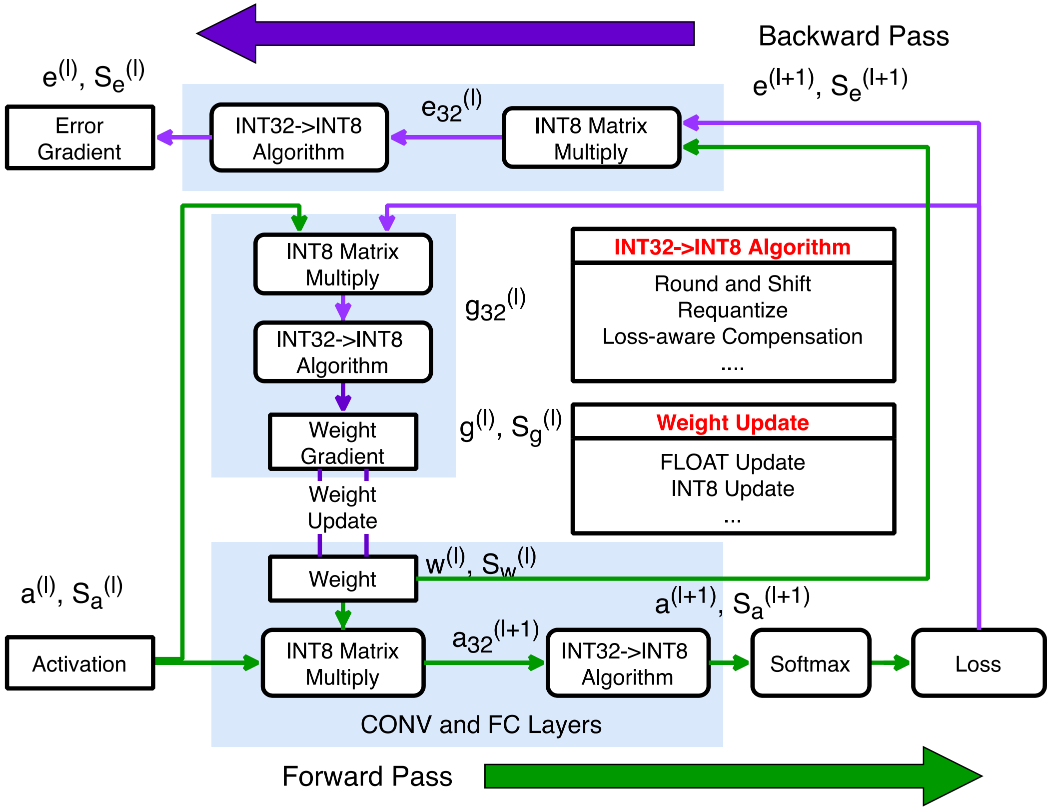
Motivation

- Mixed-precision training is emerging
  - INT8, INT16, FP16, etc…
- Mobile DSP is both ubiquitous and powerful
  - vs. CPU/GPU/NPU
  - Good at integer-based processing

Mandheling

An abstraction

- Making Mandheling a unified framework for various mixed-precision training algorithms – through a few configurations



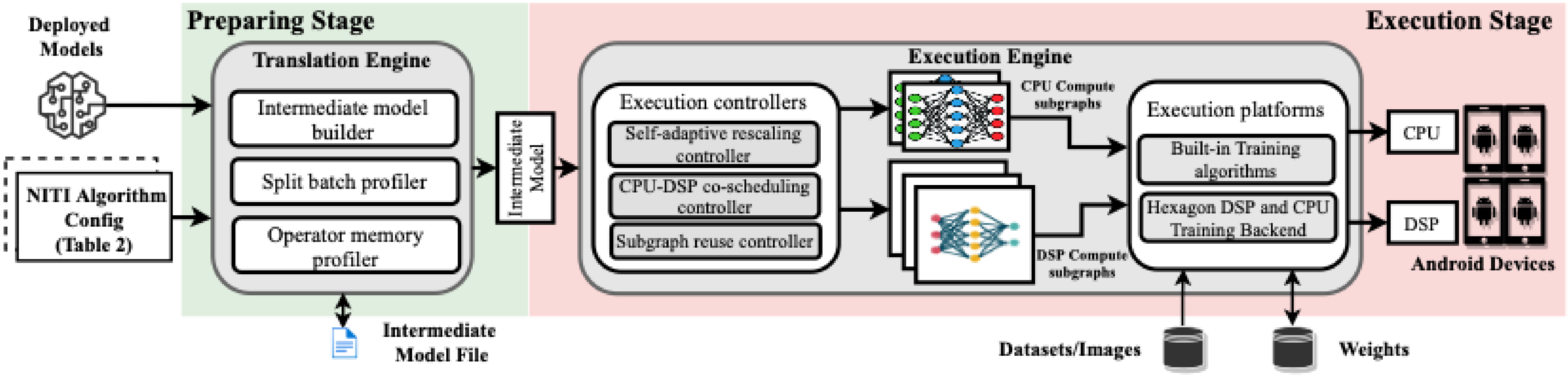
Mixed-precision algo.	W	A	G	WU	support
NITI [67]	INT8	INT8	INT8	INT8	✓
Octo [82]	INT8	INT8	INT8	INT8	✓
Adaptive Fixed-Point [79]	INT8/INT16	INT8	INT8	FP32	✓
WAGEUBN [74]	INT8	INT8	INT8	FP24	✓
MLS Format [81]	INT8	INT8	INT8	FP32	✓
Chunk-based [68]	FP8	FP8	FP8	FP16	×
Unifie					

"W", "A ... pdate.

Attribute	Contents	
	key	value
Translation	FP32 Conv	INT8 Conv+ReduceMax+Shift
	FP32 MaxPool	INT8 MaxPool
Backprop.	FP32 Conv Error Grad.	INT8 Deconv
	FP32 Conv Weight Grad.	INT8 ConvBackpropFilter
Weight	Initializer	Xavier_normal
	Type	INT8
	Update	INT8
Optimizer	Loss	Cross Entropy
	Optimizer	SGD

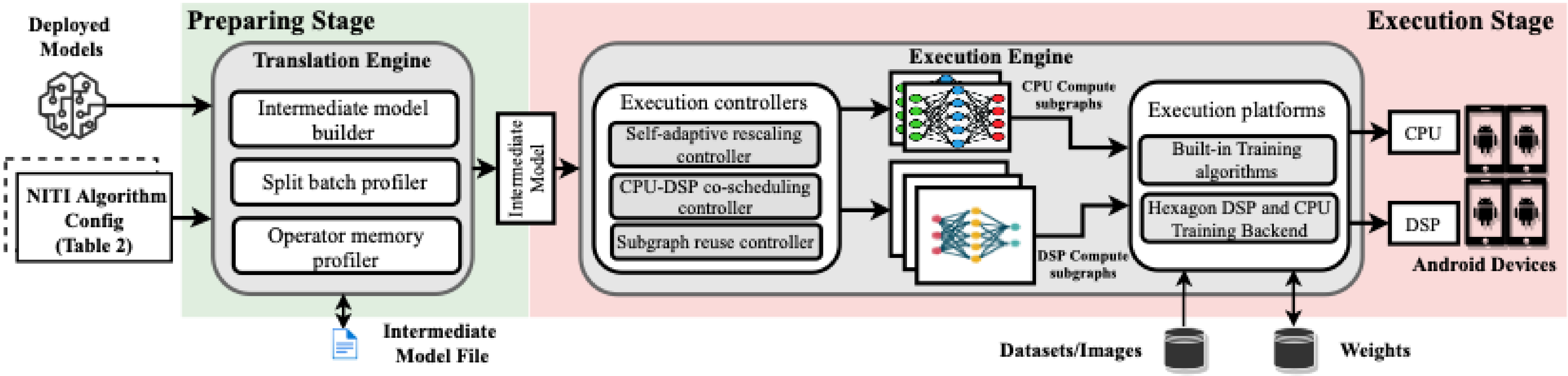
Table 2: A typical NITI algorithm training config.

System overview



Challenges	DSP-unfriendly operators	Slow dynamic rescaling (quantization ops)	Exhausted data cache	Costly compute graph preparation
Techniques	***CPU-DSP co-scheduling***	***Self-adaptive rescaling***	***Batch splitting***	***DSP-compute subgraph reuse***

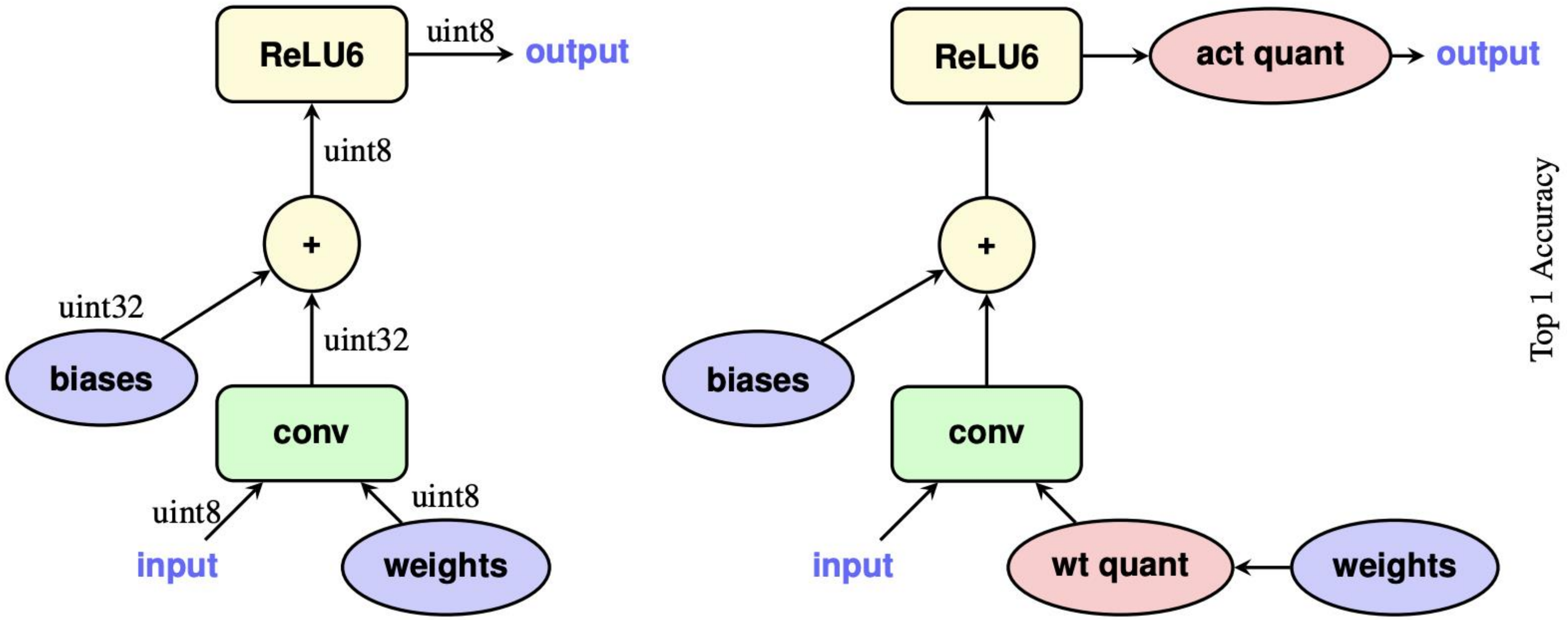
System overview



Challenges	DSP-unfriendly operators	Slow dynamic rescaling (quantization ops)	Exhausted data cache	Costly compute graph preparation
Techniques	***CPU-DSP co-scheduling***	***Self-adaptive rescaling***	***Batch splitting***	***DSP-compute subgraph reuse***

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.



(a) Integer-arithmetic-only inference (b) Training with simulated quantization

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
  - Memory-intensive

```c
int scale = 0;
/* Calculate INT32 temporal results */
for(int i = 0; i < length; i++) {
  Tensor x = input[i];
  Tensor w = weight[i];
  // CONV or matrix multiply
  Tensor temp_result = x * w;
  // count leading zero
  Tensor clz = clz(temp_result);
  int tscale = 32 - max(clz) - 7;
  scale = scale > tscale ? scale :
        tscale;
  temp_output[i] = temp_result;
}
/* Cast the INT32 to INT8 values */
for(int i = 0; i < length; i++) {
  Tensor temp = temp_output[i];
  // Downscale
  Tensor int8_result = temp / scale;
  result[i] = int8_result;
}
```

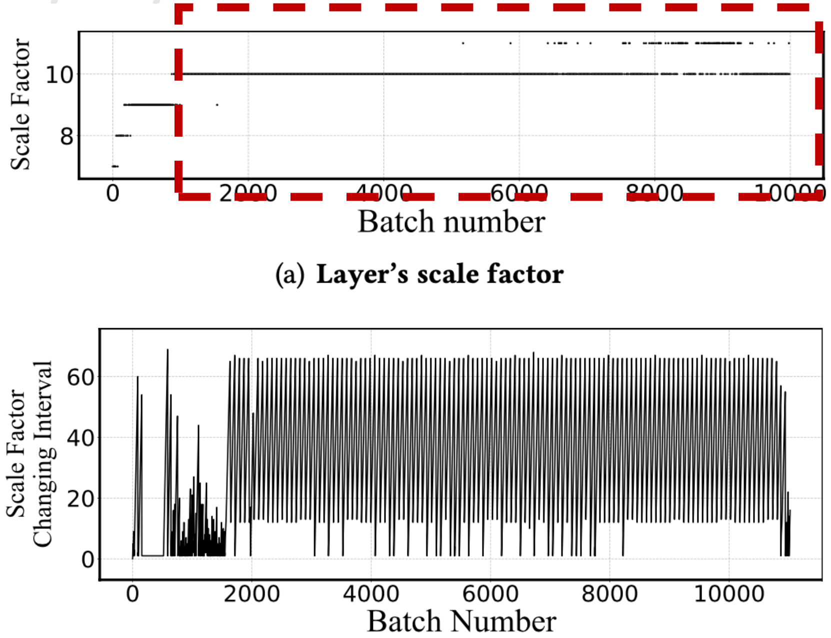
Listing 1: Key C code snippet of dynamic rescaling

```
 scale = 0

loop0:
  v0 = vmem ptr_i
  v1 = vmem ptr_w

  v2 = vrmpy v0, v1

  v3 = vclz v2
  tscale = vmax v3
  scale = mux scale >
        tscale, scale,
        tscale
  vmem ptr_t, v2
end loop0
loop1:
  v0 = vmem ptr_t

  v3 = vmpye v0, scale
  vmem ptr_v, v3
end loop1
```

Listing 2: Asm code version

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
- Opportunity
  - Very few candidates of n
  - Changing frequency is low



(a) **Layer's scale factor**

(b) **Layer's scale factor changing interval**

Figure 4: The scale factor and its changing interval of the first CONV layer in training VGG11 model (batch size = 64) on CIFAR-10 dataset.

Self-adaptive rescaling

- Scaling factor (n) needs to be dynamically adjusted.
- It runs slow on DSP, and it appears in every layer
- Opportunity
  - Very few candidates of n
  - Changing frequency is low
- **Solution: self-adaptive instead of every batch**
  - Determining the adapting frequency based on historical traces

Highlighted Results

- Implementation
  - 15k LoC in C/C++ and 800 LoC in assembly
  - Reuse ops on CPU from MNN
- Setups
  - 3 devices
  - 6 models
  - 2 datasets (CIFAR-10 & ImageNet)
- Baselines
  1. TFLite-FP32
  2. MNN-FP32
  3. MNN-INT8
  4. MNN-INT8-GPU
- Algorithm: NITI[1]

Devices	CPU	GPU	DSP
XiaoMI 11 Pro Snapdragon 888	2.84GHz Cortex-X1 3× 2.4GHz Cortex A78 4× 1.8GHz Cortex A55	Adreno 660 GPU 700MHz	Hexagon 780 DSP 500MHz
XiaoMI 10 Snapdragon 865	2.84GHz A77 3× 2.4GHz Cortex A77 4× 1.8GHz Cortex A55	Adreno 650 GPU 587MHz	Hexagon 698 DSP 500MHz
Redmi Note9 Pro Snapdragon 750G	2× 2.2GHz Cortex A77 6× 1.8GHz Cortex A55	Adreno 619 GPU 950MHz	Hexagon 694 DSP 500MHz

Table 5: Devices used in the experiments.

Model	Input Data	FLOPs	# of CONVs
VGG-11 [60]	CIFAR-10	914 M	8
VGG-16 [60]	CIFAR-10	1.35 G	13
VGG-19 [60]	ImageNet	26.92 G	16
ResNet-34 [29]	CIFAR-10	7.26 G	36
ResNet-18 [29]	ImageNet	11.66 G	20
InceptionV3 [62]	CIFAR-10	2.43 G	16

Table 6: DNN models used in the experiments.

[1] Wang, Maolin, et al. "Niti: Training integer neural networks using integer-only arithmetic." IEEE Transactions on Parallel and Distributed Systems (2022).

Highlighted Results

- Per-batch training time reduced by up to 8.3x.

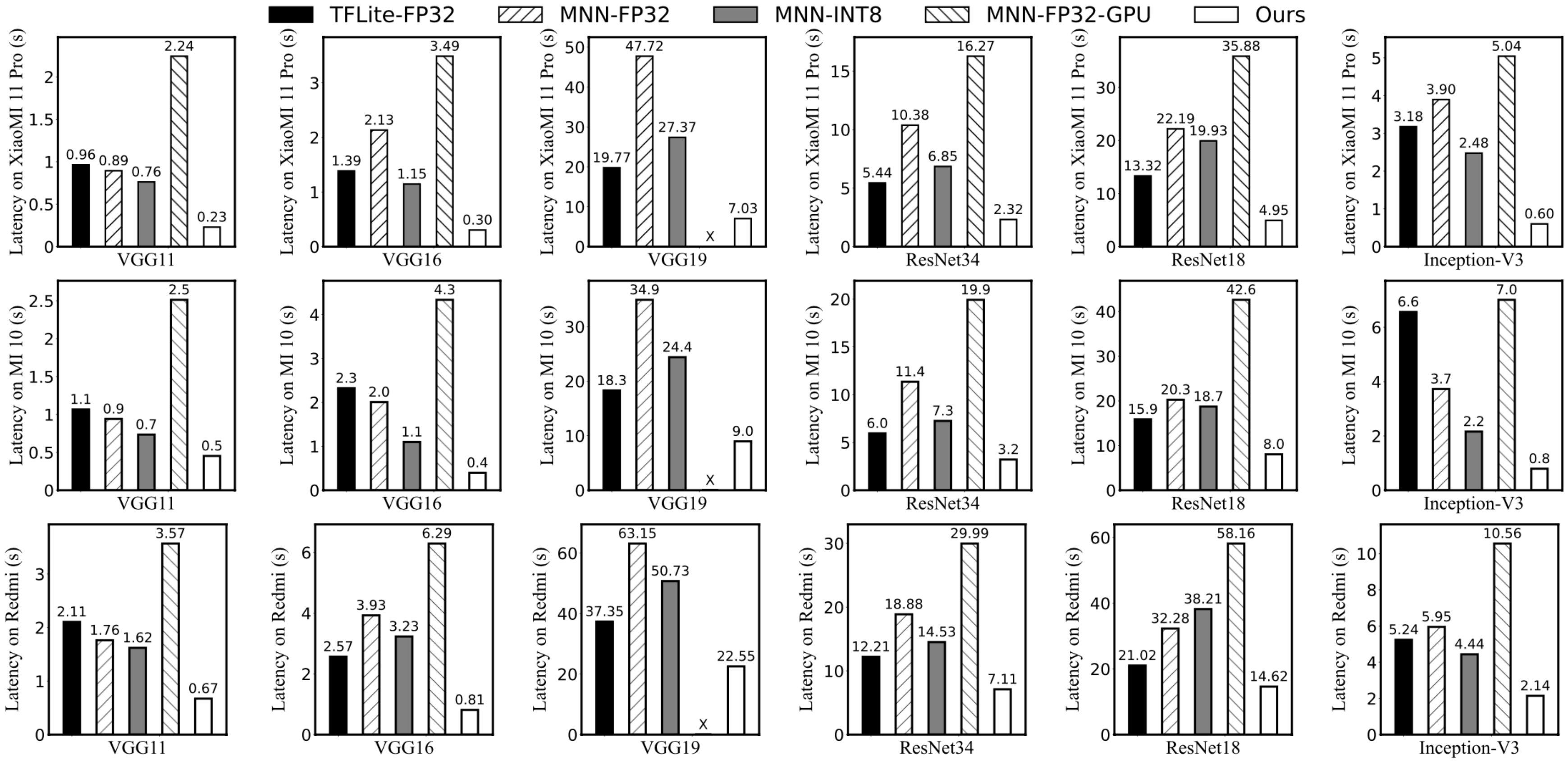


Figure 5: Per-batch training time on different models (batch size = 64) on different devices.

Highlighted Results

- Per-batch energy consumption reduced by up to 12.5x.

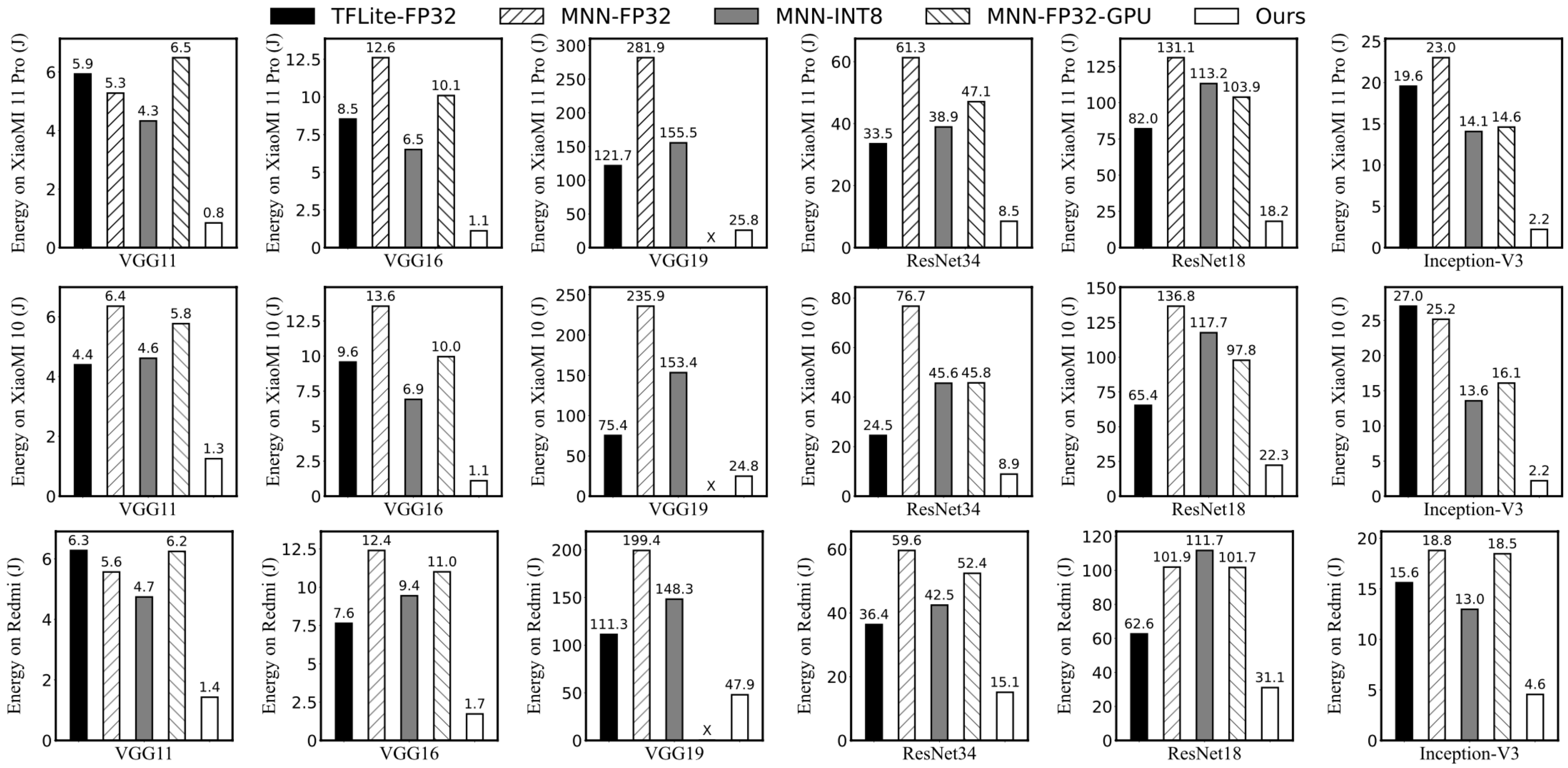


Figure 6: Per-batch energy consumption on different models (batch size = 64) on different devices.

Highlighted Results

- In end-to-end convergence tasks
  - Time reduced by 5.7x on average
  - Energy consumption reduced by 7.8x on average
  - 19.%--2.7% accuracy loss

Dataset	Model	Methods	Acc.	Training Cost to Convergence		
				Round number	Clock Hours	Energy (WH)
Centralized CIFAR-10	VGG11	MNN-FP32	89.87%	150	29.13	187.01
		MNN-INT8	87.17%	150	24.77	153.33
		Ours	87.17%	150	7.50	31.39
Centralized CIFAR-10	ResNet18	MNN-FP32	92.49%	150	223.55	1,435.19
		MNN-INT8	90.62%	150	135.71	840.04
		Ours	90.62%	150	35.68	149.32
Federated FEMNIST	LeNet	MNN-FP32	84.18%	990	0.97	0.00057
		MNN-INT8	82.04%	4,960	0.39	0.00029
		Ours	82.04%	4,960	0.19	0.00007
Federated CIFAR-100	VGG16	MNN-FP32	71.15%	1,960	8.35	2.74
		MNN-INT8	68.42%	2,200	1.56	1.26
		Ours	68.42%	2,200	0.78	0.21

Table 8: A summary of end-to-end training cost till convergence under different training scenarios.

Takeaways

- Machine (deep) learning is happening everywhere at anytime
- The system support for such ubiquitous learning is still at very preliminary stage – so many open problems!
- Open to discussion and collaboration on UL

Our code ⇒ https://github.com/UbiquitousLearning