北京大学 PEKING UNIVERSITY

北京郵電大學 Beijing University of Posts and Telecommunications

Linggui Tech Company

Mandheling: Mixed-Precision On-Device DNN Training with DSP Offloading

Daliang Xu[1*], Mengwei Xu[2*], Qipeng Wang[1], Shangguang Wang[2], Yun Ma[1], Kang Huang[3], Gang Huang[1], Xin Jin[1], Xuanzhe Liu[1]

(*co-primary)

[1]Key Lab of High Confidence Software Technologies (Peking University)

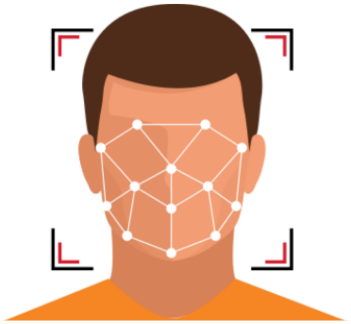
[2]State Key Laboratory of Networking and Switching Technology (BUPT)

[3]Linggui Tech Company

acm sigmobile

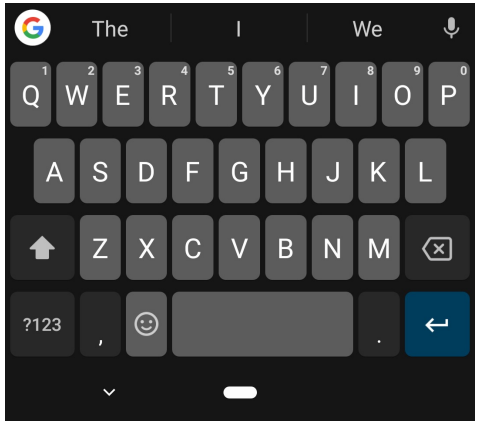
MobiCom 2022

Emerging mobile DNN applications

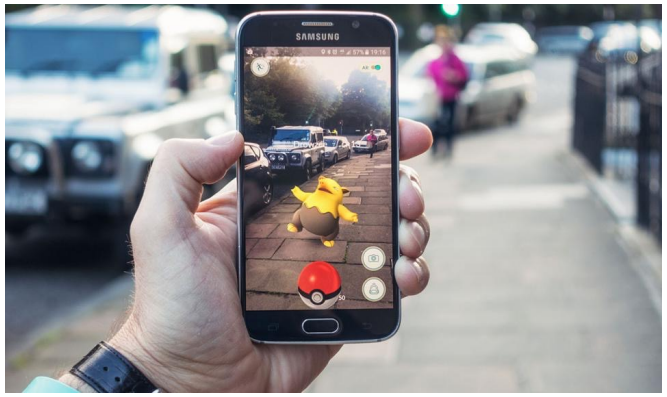


Face detection

Voice recognition



Input method editor



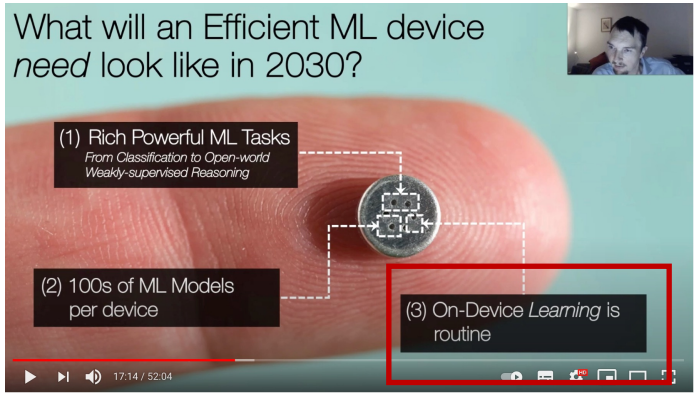
Augmented Reality

Mainly DNN inference now

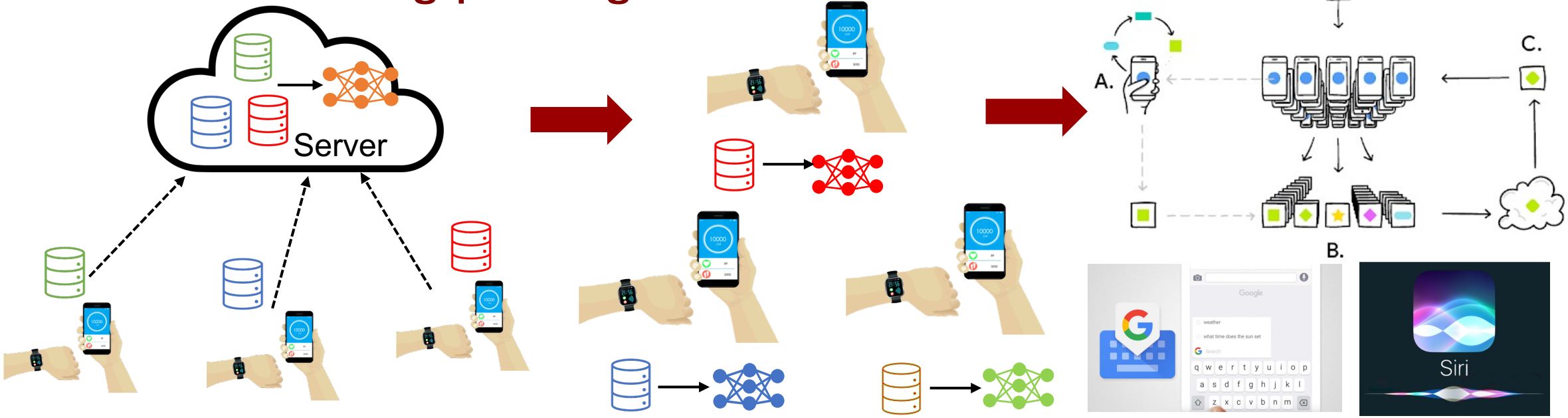
Emerging on-device learning

Privacy concerns

MobiCom 2020 - Rock Star Award - What is Next for the Efficient Machine Learning Revolution?

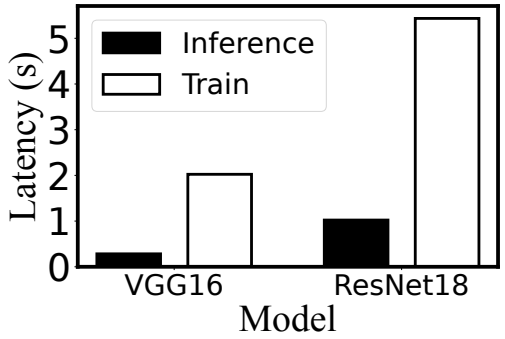


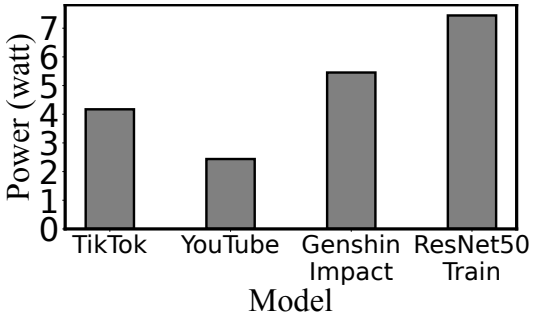
On-device training paradigm

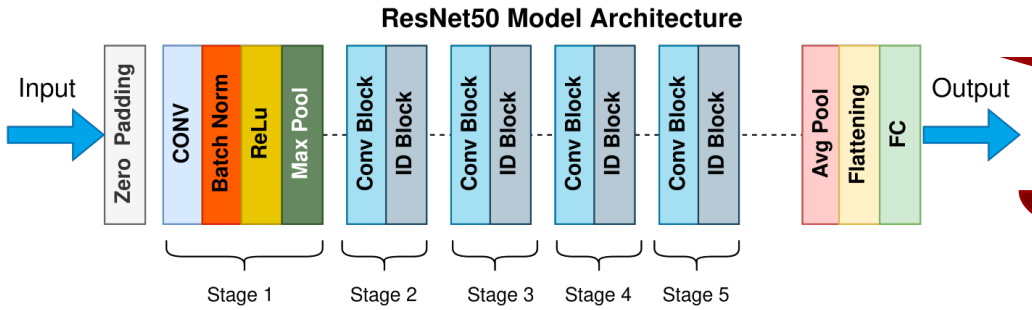


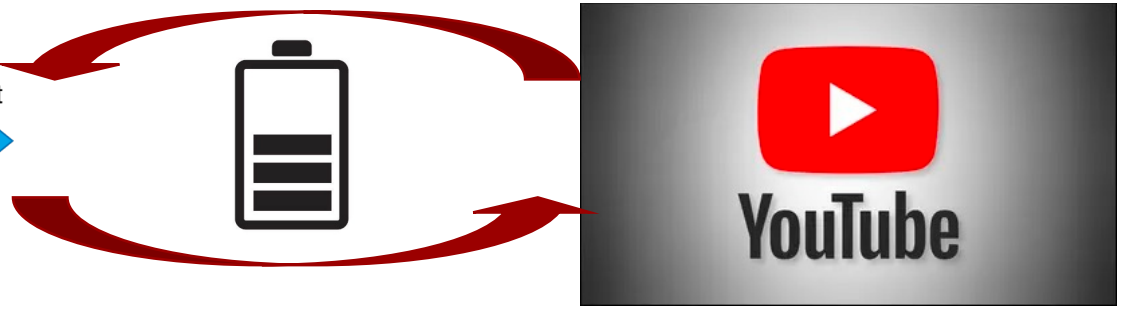
On-device training incurs huge resource cost

- On-device DNN training takes 7.14x longer than inference with the same batch size
- On-device training consumes more energy than typical video players (TikTok and YouTube) and gaming apps (Genshin Impact)







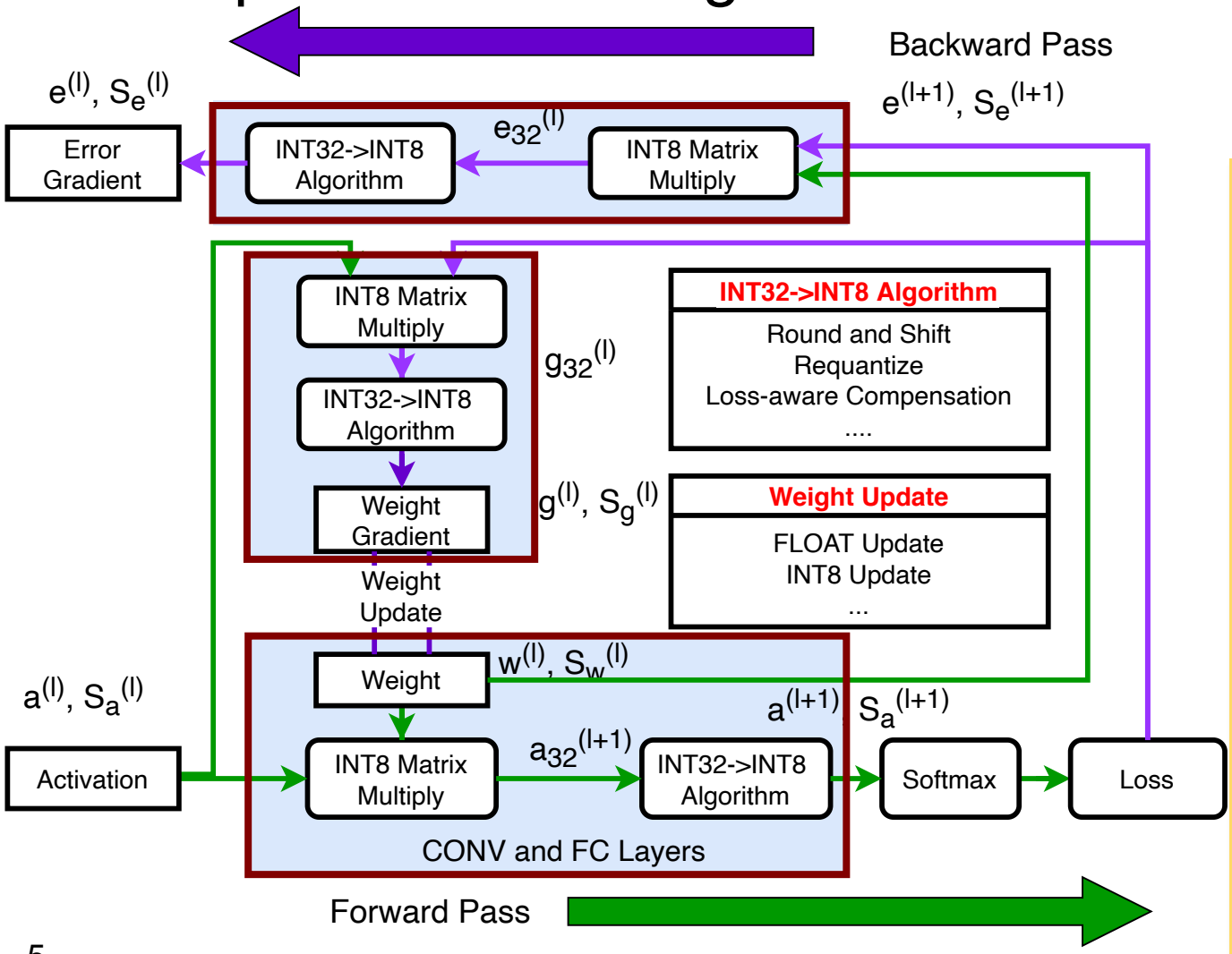


Train an epoch CIFAR10 dataset on ResNet50 model

Watch more than 7.91 hour video

Opportunity#1: mixed-precision training

- Mixed-precision training workflow



Guaranteeing convergence accuracy ✓

Lower training cost ✓

Abstraction Example

- *FP32 operators to mixed-precision operators*
  - FP32 Conv => INT8 Conv+ReduceMax+Shift
  - FP32 MaxPool => INT8 MaxPool
- *Backpropagation rules*
  - FP32 Conv Error Grad: INT8 Deconv
  - FP32 Conv Weight Grad: INT8 ConvBackpropFilter
- *Weight information*
  - *Initializer :* Xavier_normal
  - *Type: INT8*
  - *Update: INT8*
- *Optimizer information*
  - *SGD*
  - Loss: Cross-Entropy
  - Weight_decay: 5e-4
  - Learning rate: 0.01

Opportunity#1: mixed-precision training

- Support of different mixed-precision algorithms

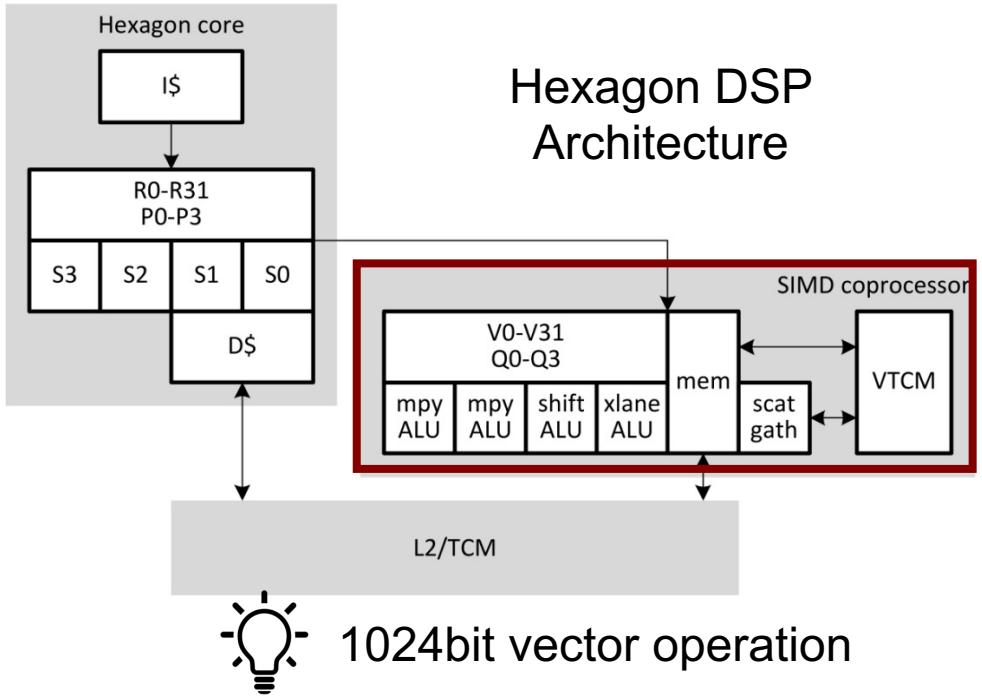
Mixed-precision algo.	Weight	Activation	Gradient	Weight Update	Support
NITI [TPDS'22]	INT8	INT8	INT8	INT8	✅
Octo [ATC'21]	INT8	INT8	INT8	INT8	✅
Adaptive Fixed-Point [CVPR'20]	INT8/INT16	INT8	INT8	FP32	✅
WAGEUBN [NN'20]	INT8	INT8	INT8	FP24	✅
MLS Format [arxiv'20]	INT8	INT8	INT8	FP32	✅
Chunk-based [NIPS'18]	FP8	FP8	FP8	FP16	❌
Unified INT8 Training [CVPR'20]	INT8	INT8	INT8	FP32	❌

- lack of support for certain low-precision operators such as FP8-based convolution
- lack of support for cosine-distance-based learning algorithms

[1] Wang M, Rasoulinezhad S, Leong P H W, et al. Niti: Training integer neural networks using integer-only arithmetic[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(11): 3249-3261.
[2] Zhou Q, Guo S, Qu Z, et al. Octo:{INT8} Training with Loss-aware Compensation and Backward Quantization for Tiny On-device Learning[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 177-191.
[3] Zhang X, Liu S, Zhang R, et al. Fixed-point back-propagation training[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 2330-2338.
[4] Yang Y, Deng L, Wu S, et al. Training high-performance and large-scale deep neural networks with full 8-bit integers[J]. Neural Networks, 2020, 125: 70-82.
[5] Zhong K, Zhao T, Ning X, et al. Towards lower bit multiplication for convolutional neural network training[J]. arXiv preprint arXiv:2006.02804, 2020, 3(4).
[6] Wang N, Choi J, Brand D, et al. Training deep neural networks with 8-bit floating point numbers[J]. Advances in neural information processing systems, 2018, 31.
[7] Zhu F, Gong R, Yu F, et al. Towards unified int8 training for convolutional neural network[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 1969-1979.

Opportunity#2: DSP efficiency in mixed-precision inference

- Hexagon Digital Signal Processor (Hexagon DSP)
  - DSP is designed to allow significant compute workloads for **advanced image processing and computer vision**
  - DSP particularly suits integer operations, i.e., **INT8-based matrix multiplication**
  - DSP has been demonstrated to be **11.3×/4.0×** more energy-efficient than CPU/GPU in DL inference tasks



Hexagon DSP Architecture

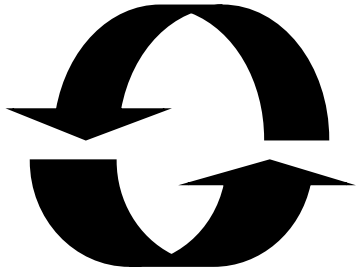
1024bit vector operation

Inference energy results on TFLite (mAh)

	CPU FP32	CPU INT8	DSP INT8
MobileNet-V1-224	0.00494	0.00126	0.000317
MobileNet-V2-224	0.00356	0.00127	0.000336
ResNet-224	0.0332	0.0073	0.00161
Inception-V4-299	0.11	0.0222	0.0033
EfficientNet-V2-224	0.0104	0.00333	0.000693

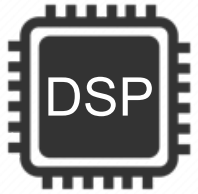
Opportunity

Mixed-precision inference



Integer matrix multiplication workload

offloading



Low power
Vector integer operation

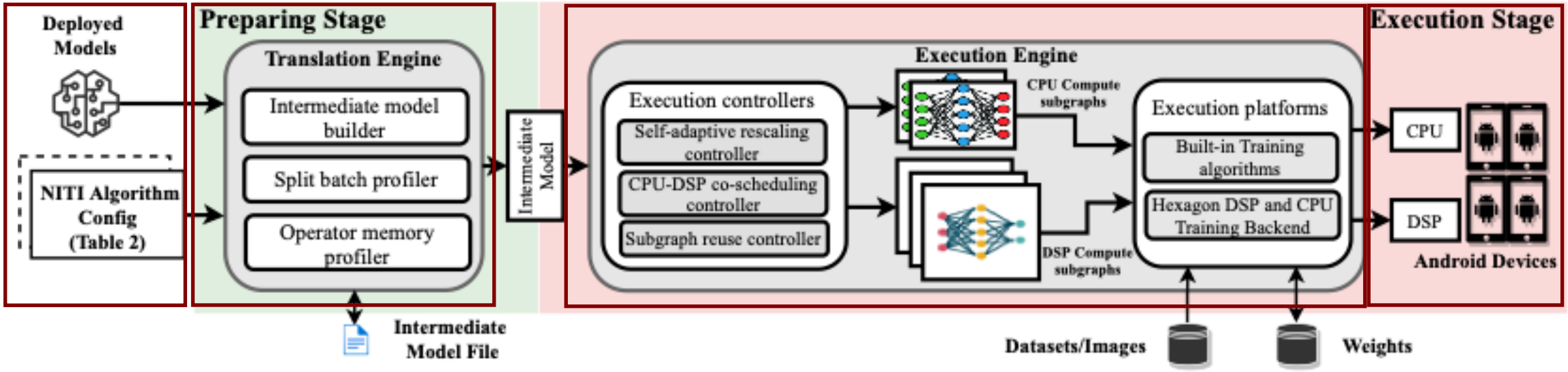
Efficient !!!

Mixed-precision training

offloading

Can we offload the mixed-precision training workload to DSP?

Design overview

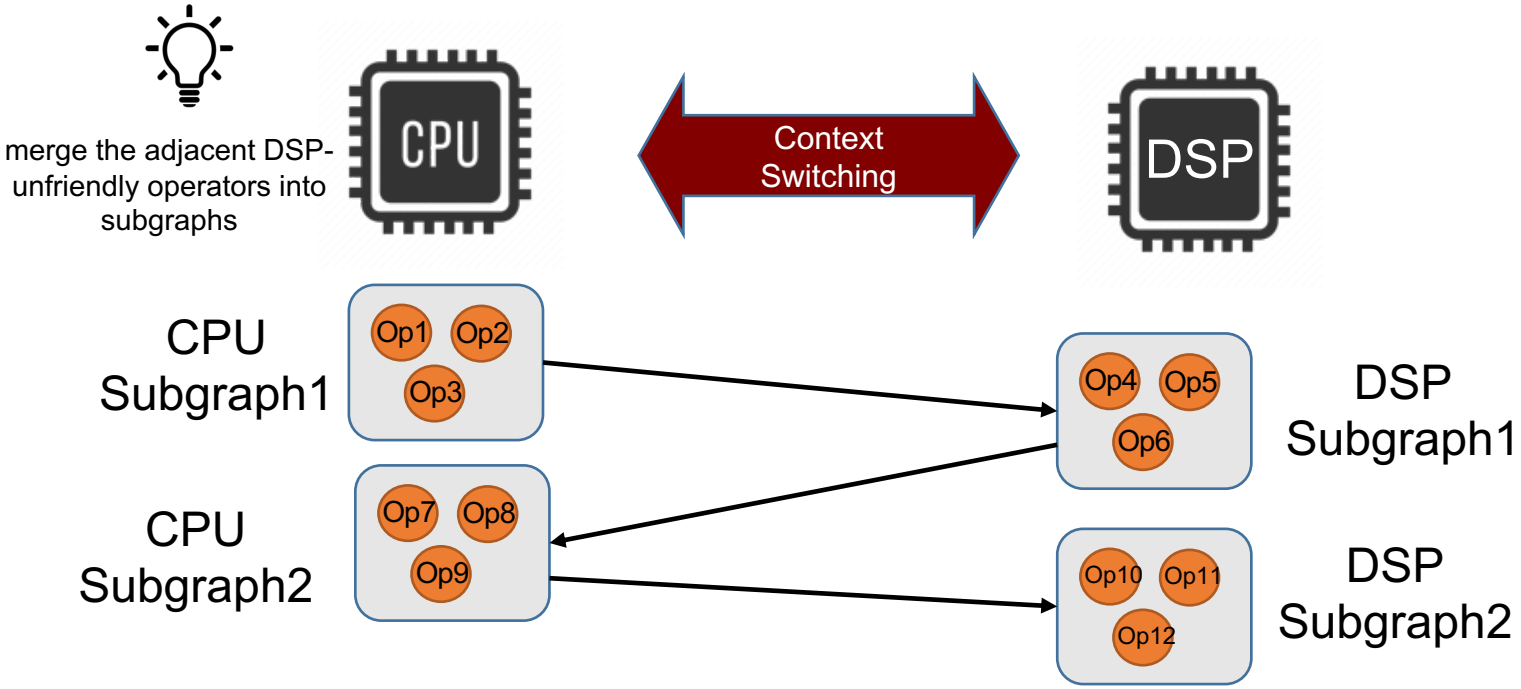


1. Input mixed-precision training algorithms config, like NITI, and deployed model file
2. Translate models from different front-end frameworks
3. Generate CPU and DSP compute subgraphs
4. Perform compute subgraph execution

Challenge#1: DSP-unfriendly operators

Operator	CPU	DSP
Transpose	3 ms	25 ms
WeightRotate	4 ms	20 ms
Slice	4 ms	17 ms
Normalization Quantization Round Sqrt	✅Support	❎ Not Support

DSP-unfriendly operators



Context switching overhead between DSP and CPU is unbearable, e.g. 25ms on Xiaomi 10.
DSP-unfriendly operators can occur anywhere.

CPU-DSP co-scheduling

- Insight#1: Reduce the context-switching overhead of DSP-unfriendly operators rather than to provide maximal CPU-DSP parallelism.

  - Topological sort: find an execution order for all operators
  - Dynamic programming algorithm: determine the operator placement

Recursion formula

$$T[i+1, CPU] = min \begin{cases} T[i, CPU] + L_{i+1}^{CPU} \\ T[i, DSP] + L_{i+1}^{CPU} + L_{switch} \end{cases}$$

$$T[i+1, DSP] = min \begin{cases} T[i, CPU] + L_{i+1}^{DSP} + L_{switch} \\ T[i, DSP] + L_{i+1}^{DSP} \end{cases}$$

Objective function

$$T_{model} = min\{T[N, CPU], T[N, DSP]\}$$

Challenge#2: dynamic rescaling does not fit DSP

- Dynamic rescaling

```
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;
}
```

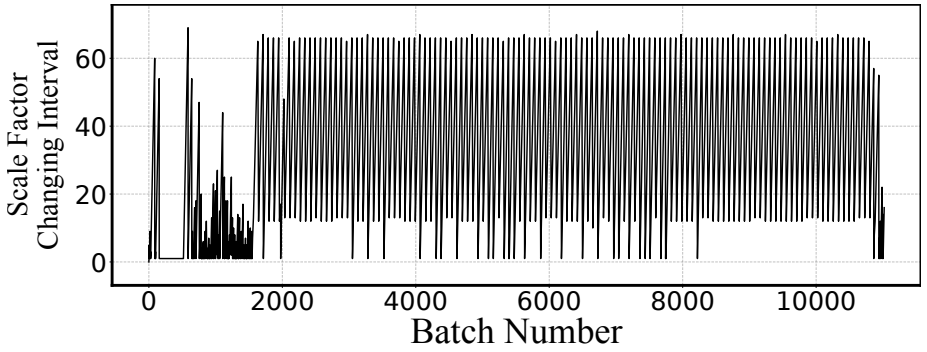
```
  scale = 0

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
```

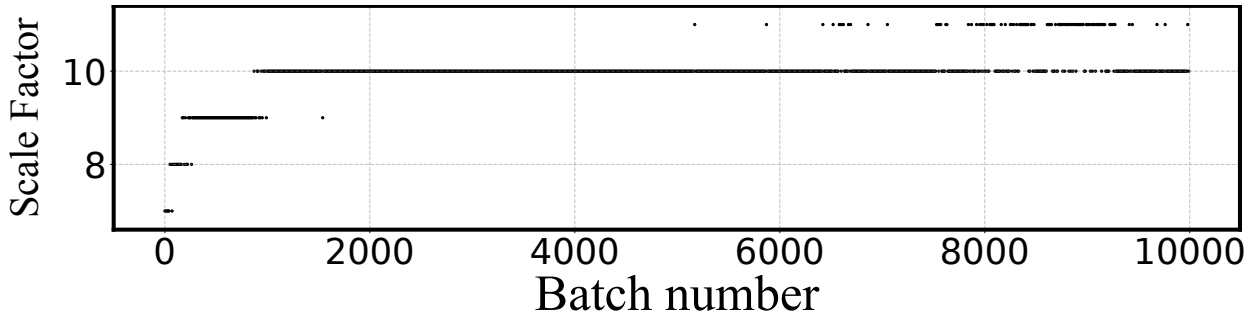
☹ The scale factor needs to be dynamically adapted.

☹ Downscaling the temporal outputs to final INT8 results needs to store the temporal outputs and reload them.

Add at least 2x latency compared with static rescaling

Self-adaptive rescaling





- Insight#2
  - The scale factor jumps between two alternative values.
  - The actual changing frequency of the scale factor is low, e.g., per 10–60 batches.

Self-adaptive rescaling: Periodically enable rescaling, instead of per batch

Challenge#3: exhausted data cache

Input size	Latencies of different batch sizes (ms)					
	2	4	8	16	32	64
8 x 8	0.63	0.63	0.85	1.03	1.27	1.84
16 x 16	0.84	0.89	4.23	3.98	4.64	12.24
32 x 32	1.69	2.50	59.11	62.35	68.14	152.89

- *Training tasks often require a large batch size*
- *The DSP cache is often smaller than the CPU cache*

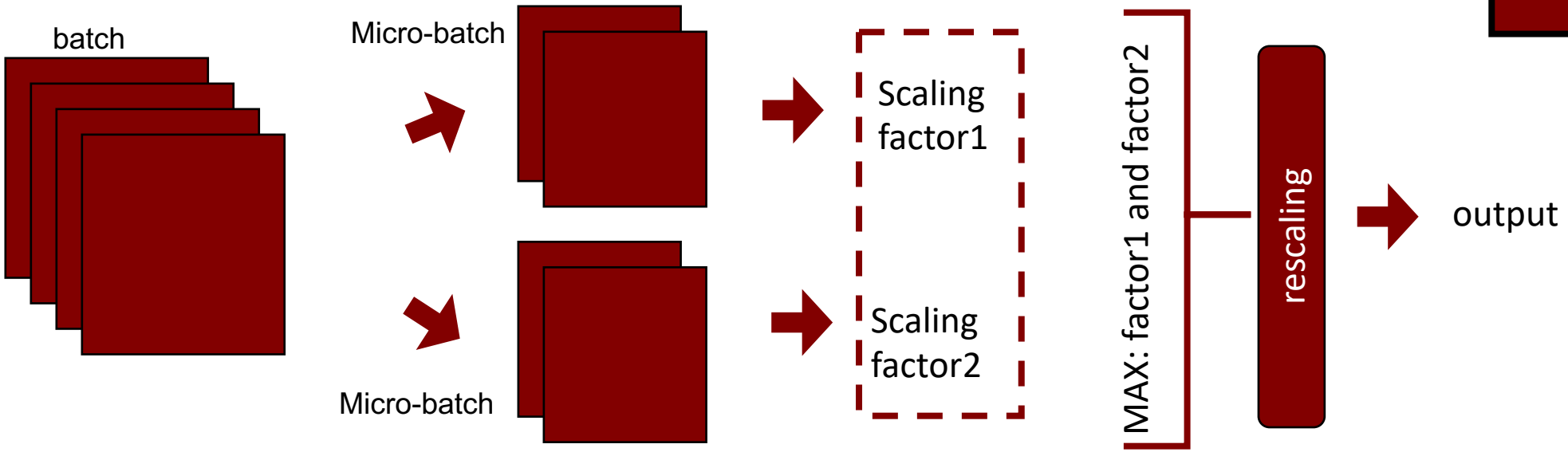
8× theoretical workload
27.3×latemcy

Batch splitting

Input size	Latencies of different batch sizes (ms)					
	2	4	8	16	32	64
8 x 8	0.63	0.63	0.85	1.03	1.27	1.84
16 x 16	0.84	0.89	4.23	3.98	4.64	12.24
32 x 32	1.69	2.50	59.11	62.35	68.14	152.89

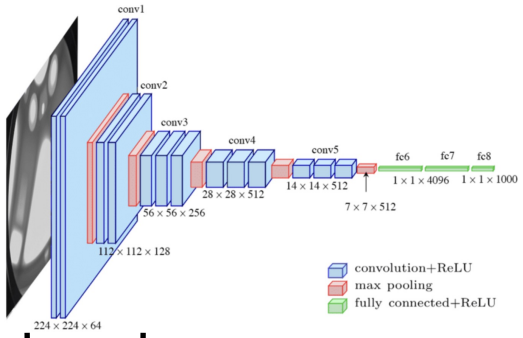
Abnormally

- *Training tasks often require a large batch size*
- *The DSP cache is often smaller than the CPU cache*



Challenge#4: costly compute-subgraph preparation

VGG16 Model



Establish one operator node

Build the whole compute subgraph

Establish operator's inputs and outputs list

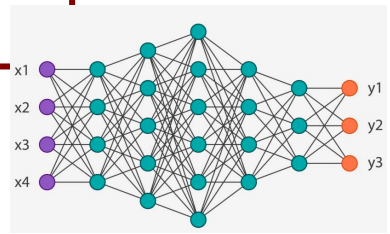
Establish a new graph node

Establish const attribute nodes, like kernel size and stride

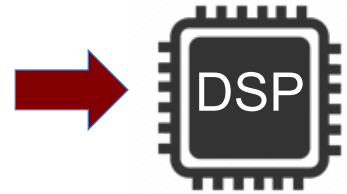
Allocate memory for inputs and outputs

Transform data layouts, e.g. plain to d32

Optimize the compute graph, like operator fusion



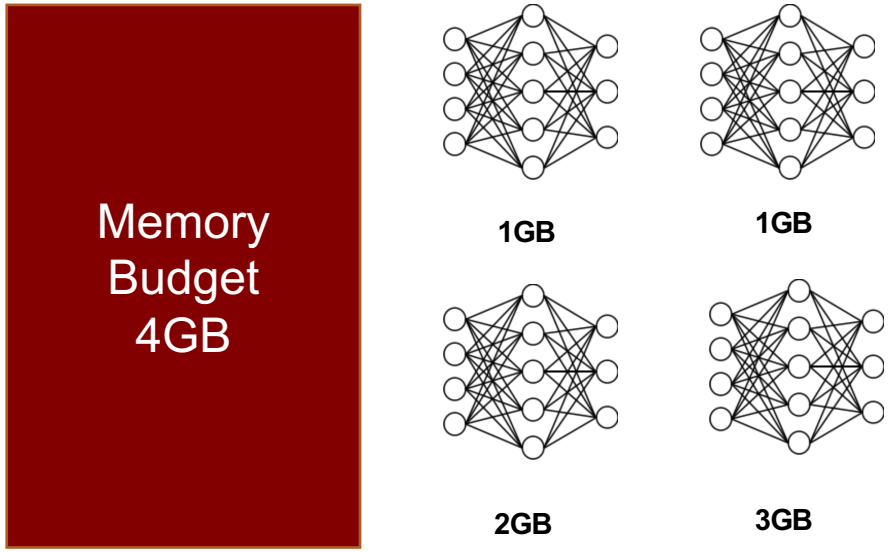
Hexagon DSP compute subgraph



304ms on TFLite and 212ms on MNN for the VGG16 model

DSP compute subgraph reuse

- Insight#4: the models are rarely modified during on-device training



- Preparing stage: Mandheling profiles all memory regions used by compute subgraphs.
- Execution stage: Mandheling releases the MRU memory regions and allocates space for new subgraphs

Memory budget limits the number of DSP compute graph

Evaluation: settings

- Devices and models

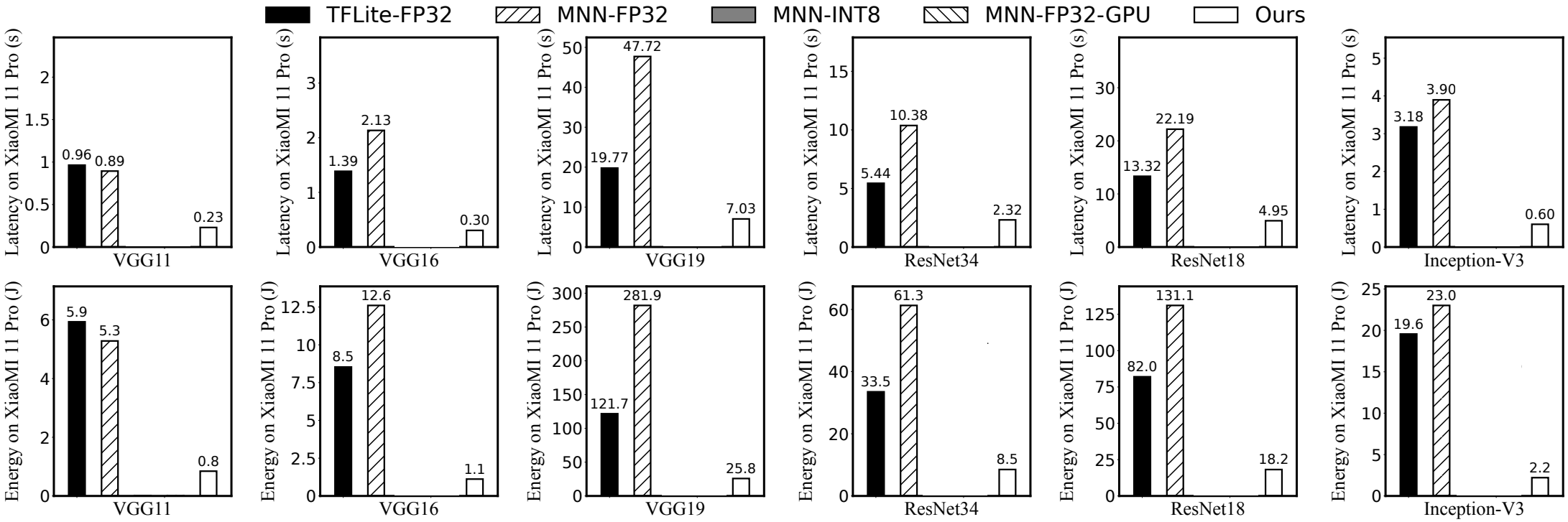
Device	SoC	Memory
Xiaomi 11 Pro	SnapDragon 888	12GB
Xiaomi 10	SnapDragon 865	8GB
Redmi Note9 Pro	SnapDragon 750G	8GB

Model	Dataset	FLOPS	# of CONVs
VGG-11	CIFAR-10	914M	8
VGG-16	CIFAR-10	1.35G	13
VGG-19	ImageNet	26.92G	16
ResNet-34	CIFAR-10	7.26G	36
ResNet-18	ImageNet	11.66G	20
InceptionV3	CIFAR-10	2.43G	16

- Baselines
  - TFLite-FP32: TFLite FP32-based training method
  - MNN-FP32: MNN FP32-based training method
  - MNN-INT8: the INT8-based training method implemented by us based on MNN (NEON is extensively used to optimize the performance of this baseline)
  - MNN-FP32-GPU: FP32-based training on mobile GPU through the OpenCL backend

Evaluation: per-batch performance

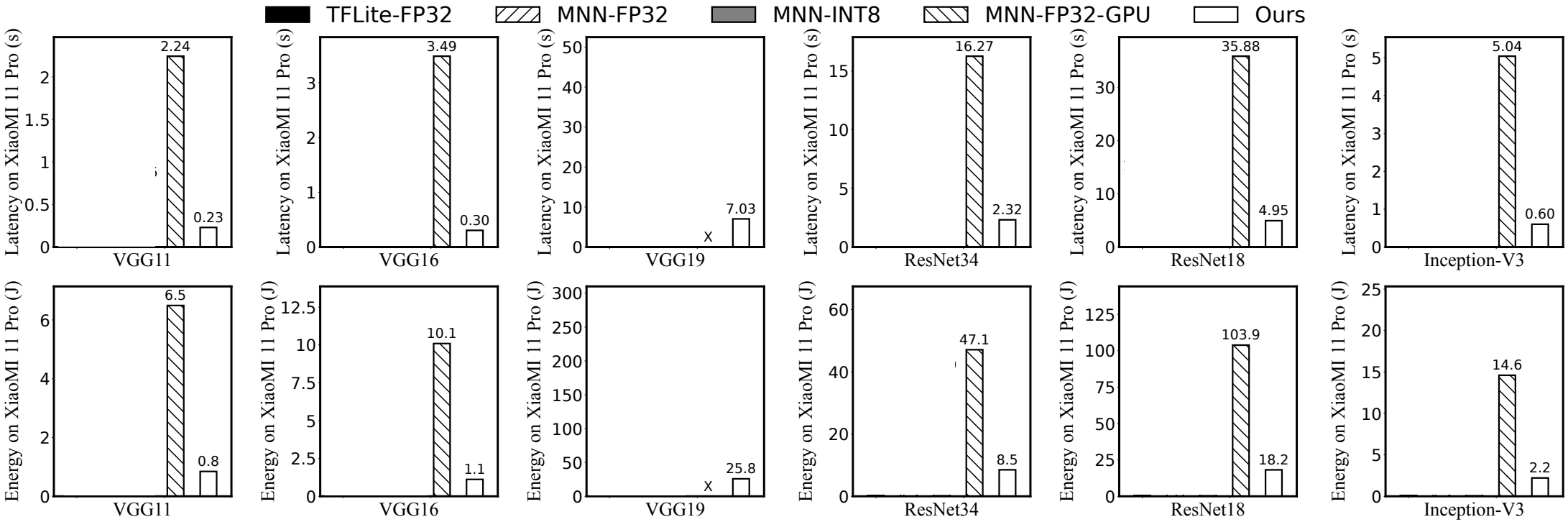
- Mandheling v.s. FP32 training



- Reduce per-batch training time by up to 8.5×.
- Reduce per-batch training energy consumption by up to 11.2×.

Evaluation: per-batch performance

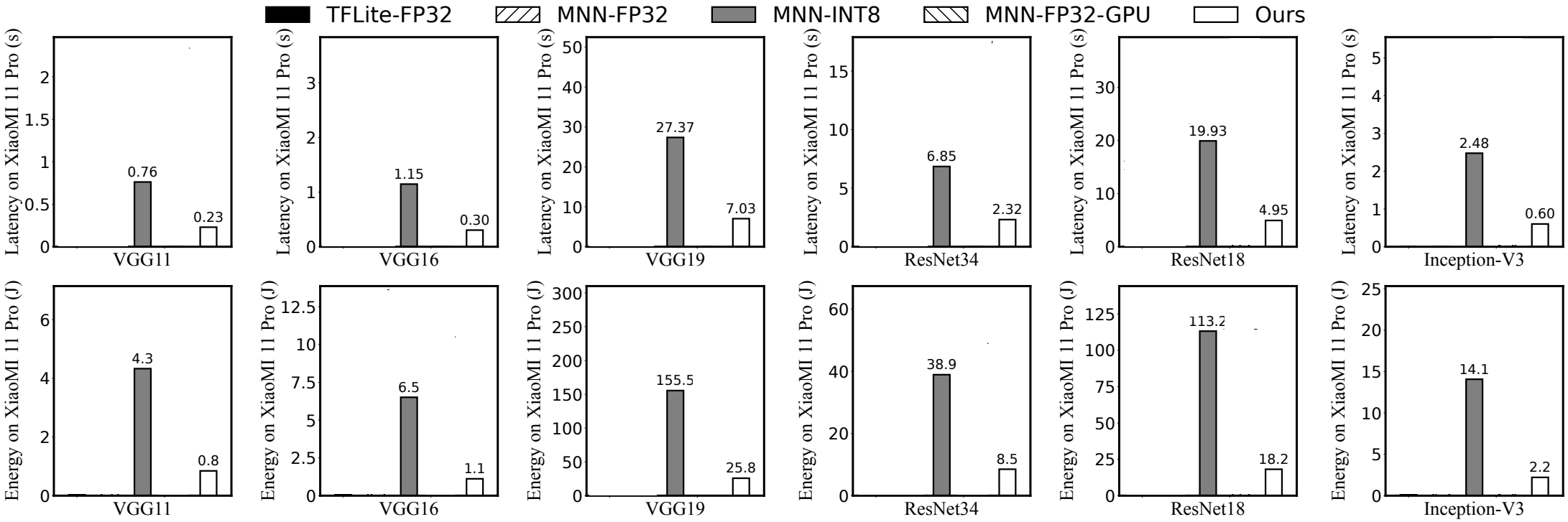
- Mandheling v.s. GPU



- Reduce per-batch training time by 3.98-11.63×.
- Reduce per-batch training energy consumption by 3.46-10.95×.

Evaluation: per-batch performance

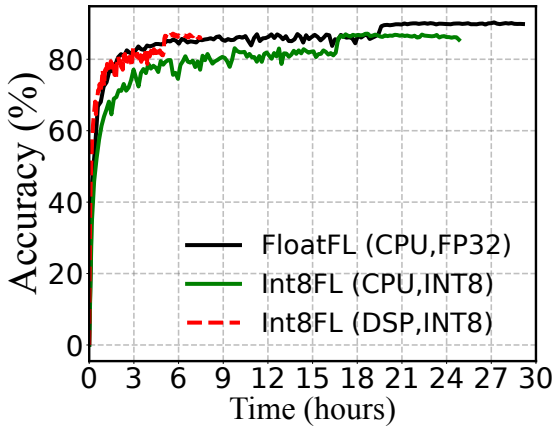
- Mandheling v.s. MNN INT8



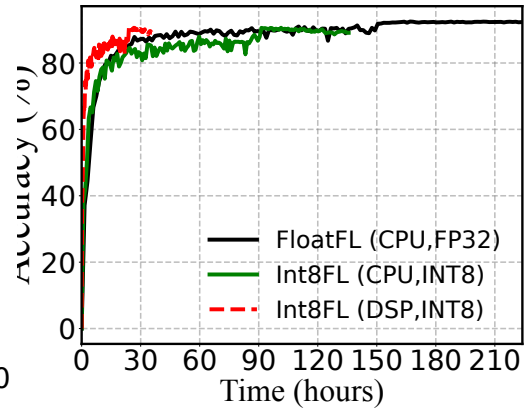
- Reduce per-batch training time by up to 4.13×.
- Reduce per-batch training energy consumption by up to 6.5×.

Evaluation: end-to-end performance

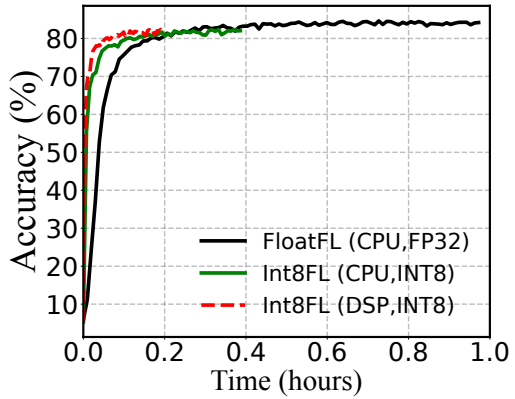
- Training on device takes 3.55× less time
- Cross-device federated learning achieves a 5.26× and 10.75× speedup



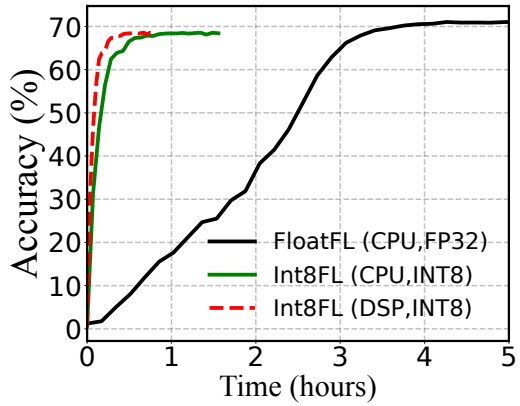
VGG11-CIFAR10-Local



ResNet18-CIFAR10-Local



LENET-FEMNIST-Federated



VGG16-CIFAR100-Federated

Evaluation: end-to-end performance

- Reduce the energy consumption of a single client by 5.16-13.1×
- Only 1.9%-2.7% loss compared to the accuracy of the FP32 precision setting

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

Conclusion

- The first on-device training system that enables mixed-precision training with on-chip DSP offloading

- Support many DSP-friendly operators

- Open source: **https://github.com/UbiquitousLearning/Mandheling-DSP-Training**

Mandheling: Mixed-Precision On-Device DNN Training with DSP Offloading

On-device Training

Mixed-precision

DSP offloading

Code: https://github.com/UbiquitousLearning/Mandheling-DSP-Training

xudaliang@pku.edu.cn

website https://daliangxu.github.io/

More about our affordable AI systems: http://www.liuxuanzhe.com