Melon: Breaking the Memory Wall for Resource-Efficient On-Device Machine Learning

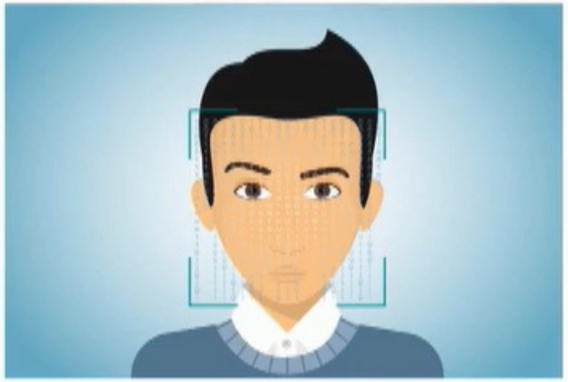
MobiSys'22

Qipeng Wang[1*], Mengwei Xu[2*#], Chao Jin[1], Xinran Dong[1], Jinliang Yuan[2], Xin Jin[1], Gang Huang[1], Yunxin Liu[3], Xuanzhe Liu[1#]

[1]Key Lab of High Confidence Software Technologies (Peking University), Beijing, China
[2]State Key Laboratory of Networking and Switching Technology (BUPT), Beijing, China
[3]Institute for AI Industry Research (AIR), Tsinghua University, Beijing, China
wangqipeng@stu.pku.edu.cn,{chaojin,dongxinran0805,xinjinpku,hg,xzl}@pku.edu.cn
{mwx,yuanjinliang}@bupt.edu.cn
liuyunxin@air.tsinghua.edu.cn

Introduction

DNN is a key component for mobile app.



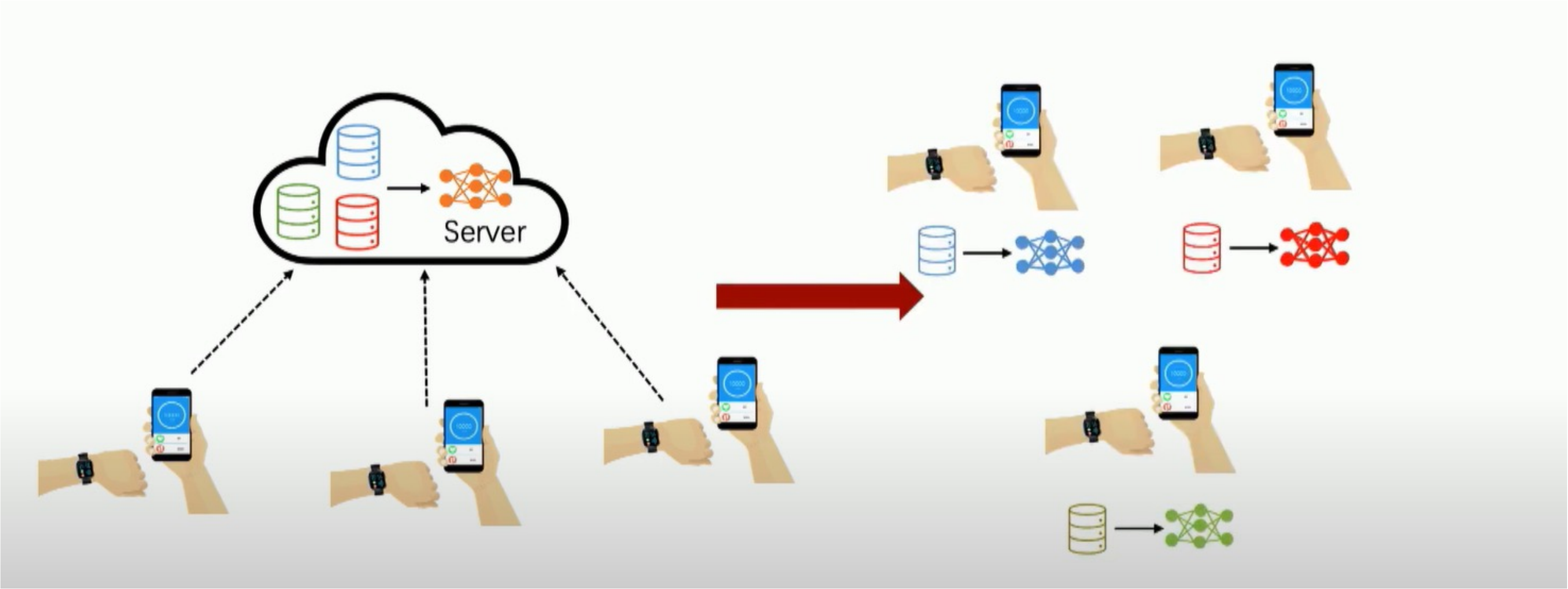
Face detection

Voice recognition

Augmented Reality

Mainly DNN inference now

Introduction

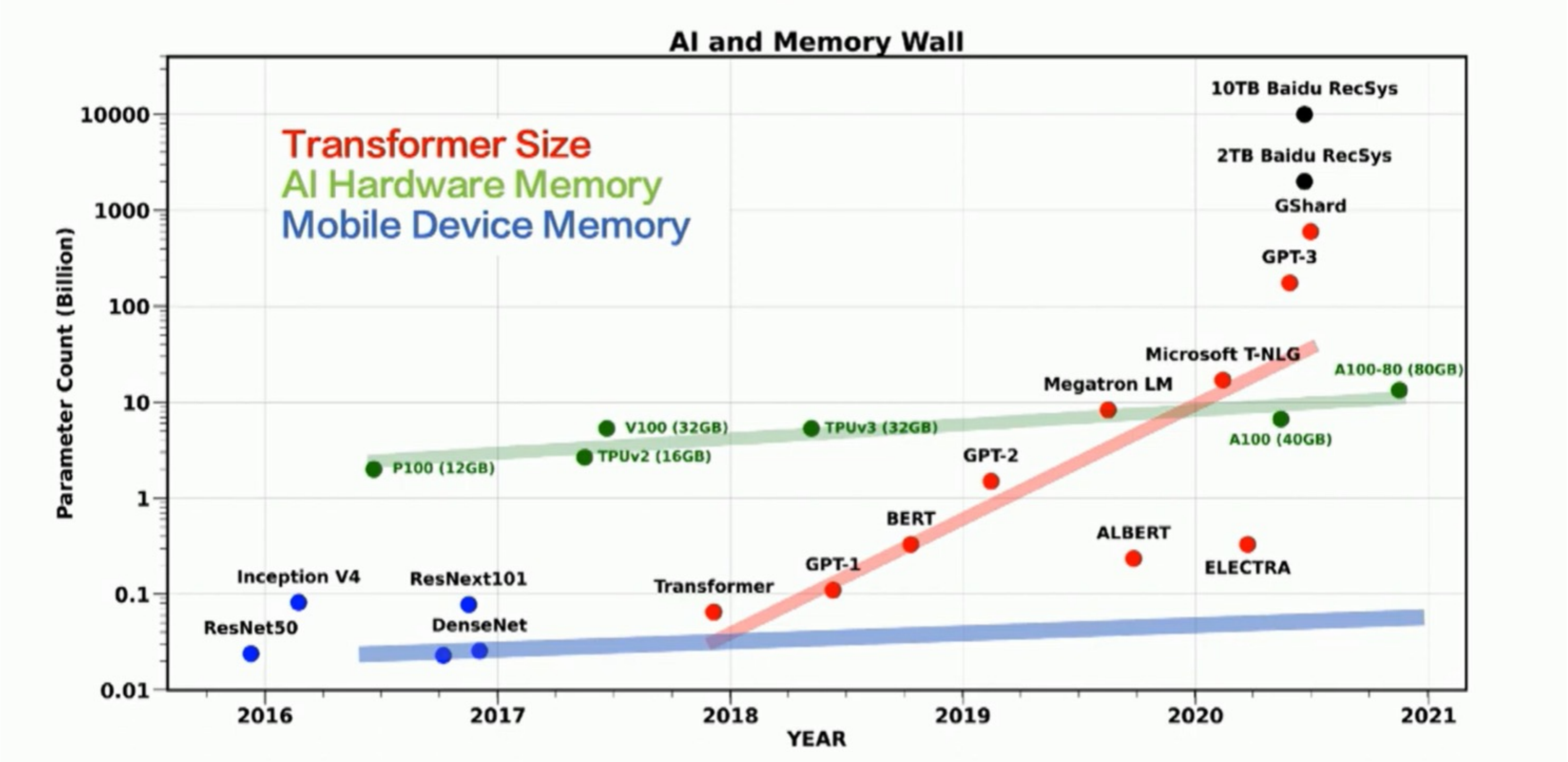


on-server
date leak

on-device
privacy protection

Whether the training of modern DNN is affordable on mobile devices?

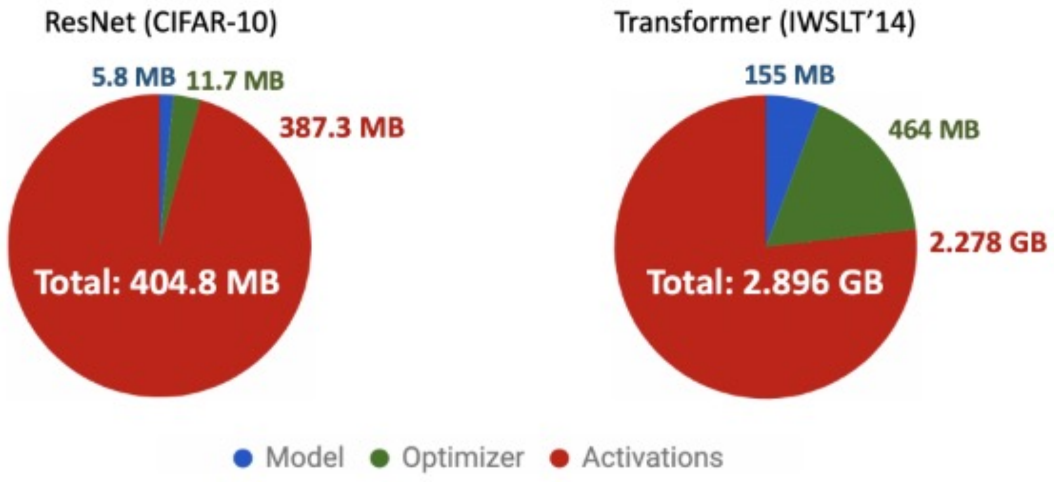
Introduction



Training DNN with large batch size is not affordable on mobile device.

Motivation

Breakdown the use of memory while training



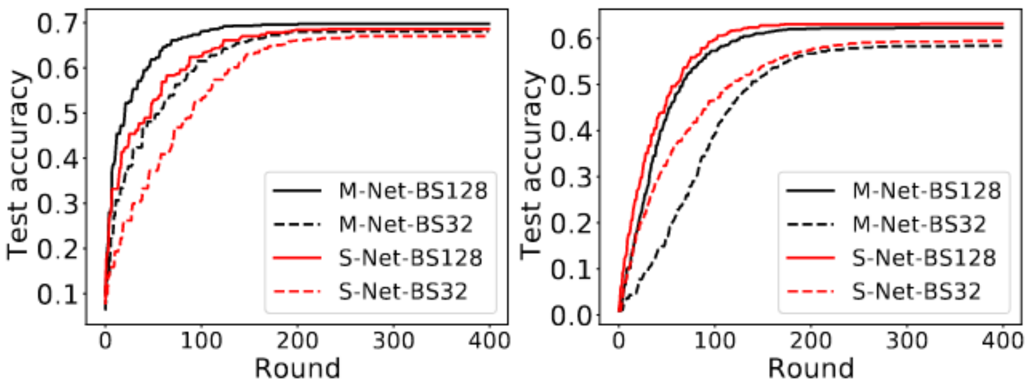
Model: storing parameters

Optimizer: storing gradients

Activations: storing intermediate outputs

Motivation

Why we aim to break the memory wall?



(a) Centralized learning (b) Federated learning

Using larger batch size leads to 2% higher accuracy or 39.02% faster convergence time

Higher accuracy and Faster

Training models with larger batch size **requires much more memory** capacity

Motivation

Why we aim to break the memory wall?

Settings	convergence accuracy and round			
	Original (BS=32)		Optimal (BS=128)	
	Accuracy	Round	Accuracy	Round
M-Net-centralized	67.58%	171	69.56%	123
M-Net-federated	58.22%	239	62.16%	164
S-Net-centralized	66.24%	211	68.28%	155
S-Net-federated	59.18%	191	62.96%	168

Table 1: The convergence result that can be achieved on devices with different memory capacities. "M-Net": MobileNetV2; "S-Net": SqueezeNet.

MNN can only support batch size 32

Target: break the memory wall through memory optimization techniques

Exploring existing techniques

The **existing memory saving techniques** that are originally designed for the **cloud.**

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

Exploring existing techniques

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

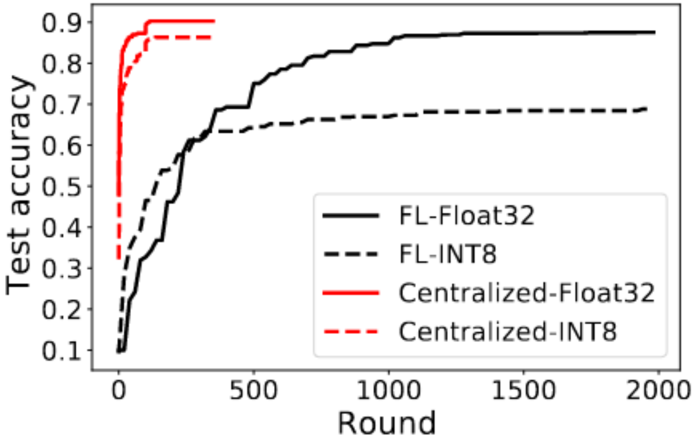


Figure 3: The accuracy loss of training-time compression is amplified in federated learning compared to centralized setting, with MobileNetV2 and CIFAR-10.

float32 → int8

siginificant accuracy drop

Exploring existing techniques

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

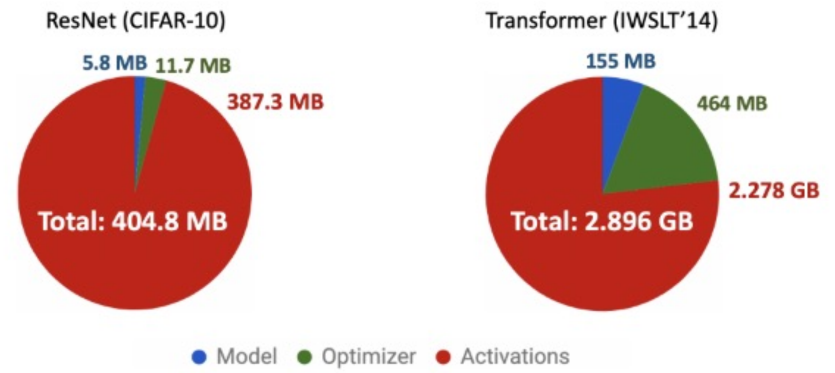
Performance is blocked significantly by I/O speed of mobile devices.

PICE5.0: 128GB/S CPU: 100MB-300MB/S

Exploring existing techniques

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

Discarding the intermediate activation during forward pass and **recomputing them when needed** at the backward stage.

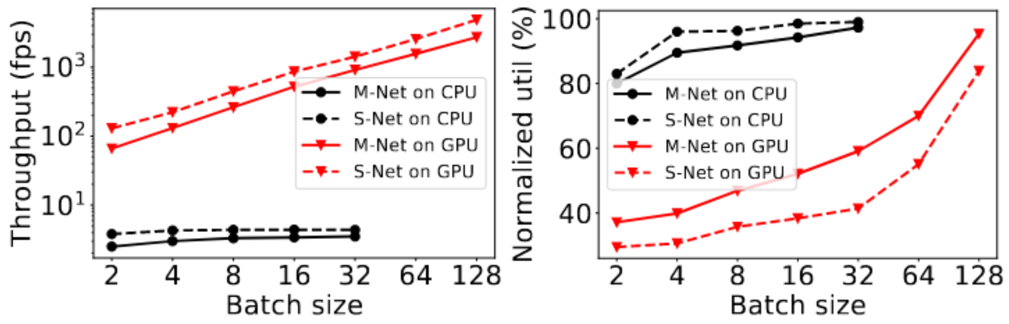


Useful and it does not decay model accuracy

Problem: current algorithms does not consider the effect of memory pool.

Exploring existing techniques

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



(a) Training throughput **(b) Hardware utilization**

Figure 4: A relatively small batch size is enough to fully exploit mobile CPU capacity. "M/S-Net": MobileNetV2/SqueezeNet-50; "CPU": Samsung Note10 CPU; "GPU": Nvidia P100.

mini-batch 128 → micro batch 4*32

A small micro-batch size cannot fully utilize the high parallelism of cloud GPUs

On mobile devices, however, a relatively small batch size is **sufficient to reach the maximal hardware resource utilization**

Problem: Not support the models with BN

Exploring existing techniques

Summarized Implications

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

Melon needs to **retrofit** the proper techniques (microbatch and recomputation)

How to break the memory wall for on-device learning?

Melon Design

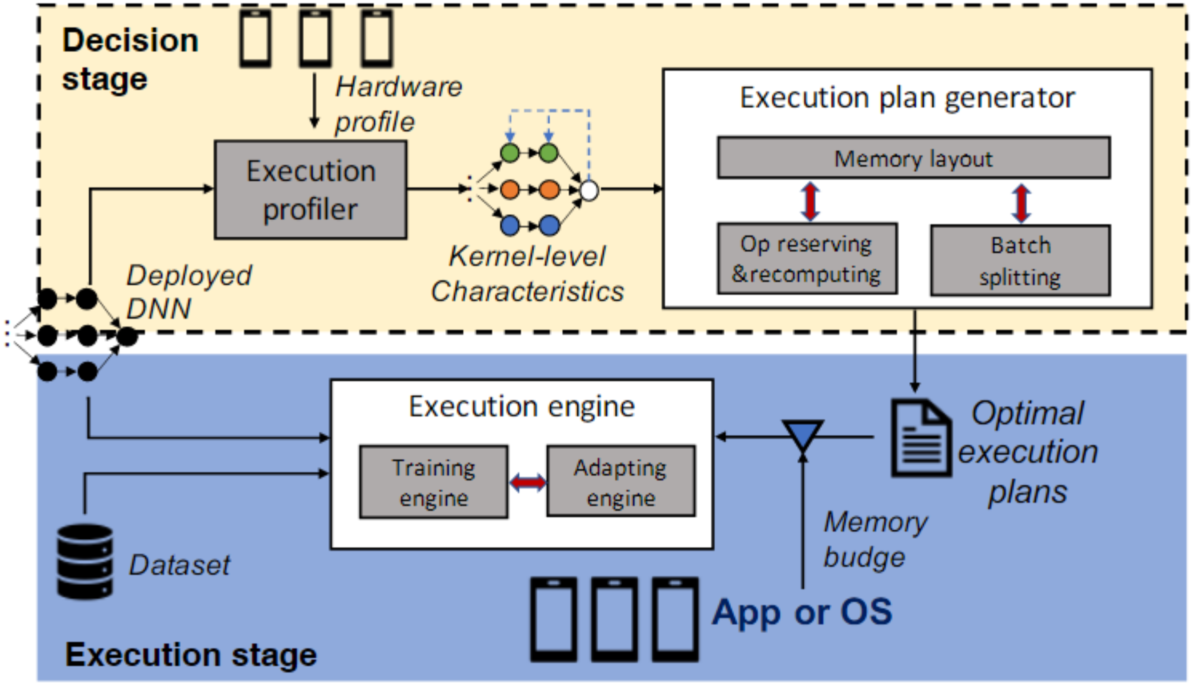


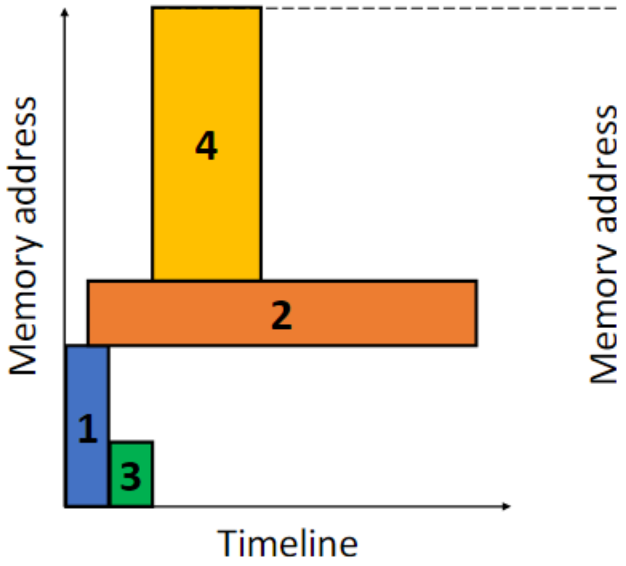
Figure 5: An overview of Melon.

Challenges & Techniques

#1 efficient memory management

How to manage memory pool efficiently and specifically for DNN training?

Memory pool is widely adopted

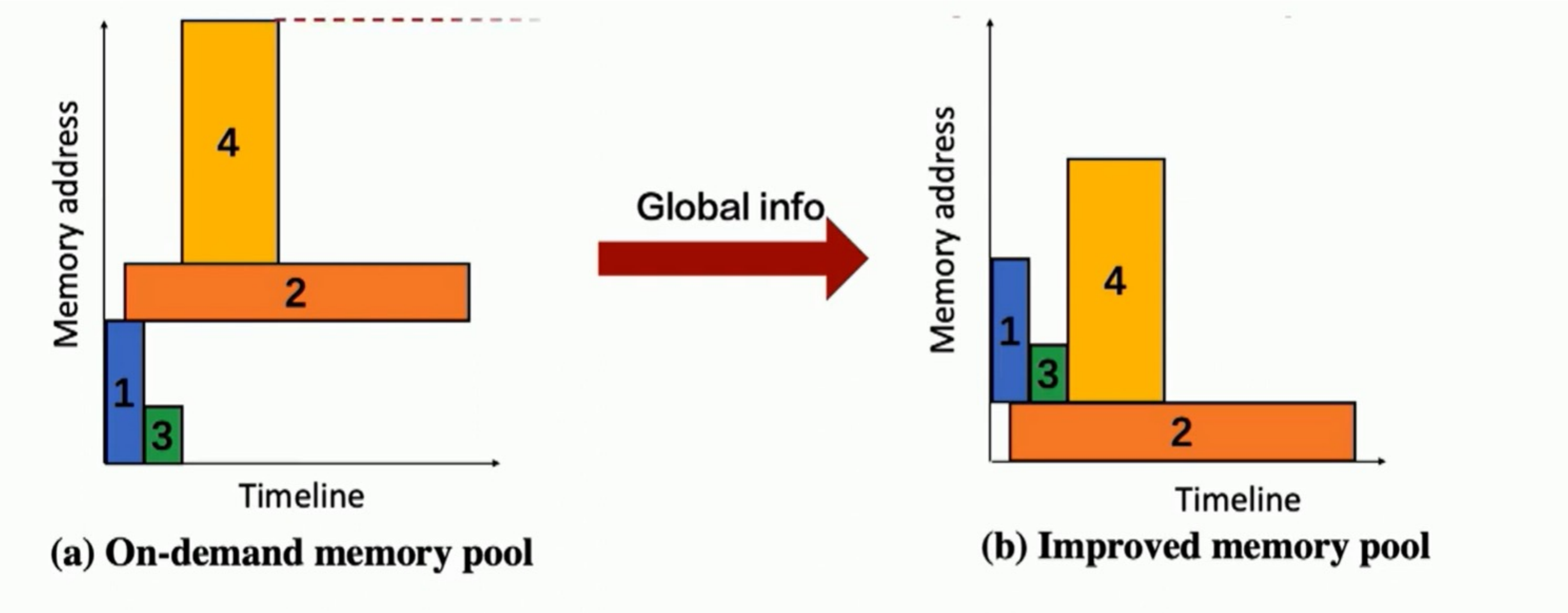


(a) On-demand memory pool

Challenges & Techniques

#1 effiicient memory management

How to manage memory pool efficiently and specifically for DNN training?



Place those long-lifetime tensors beneath short-lifetime ones.

Challenges & Techniques

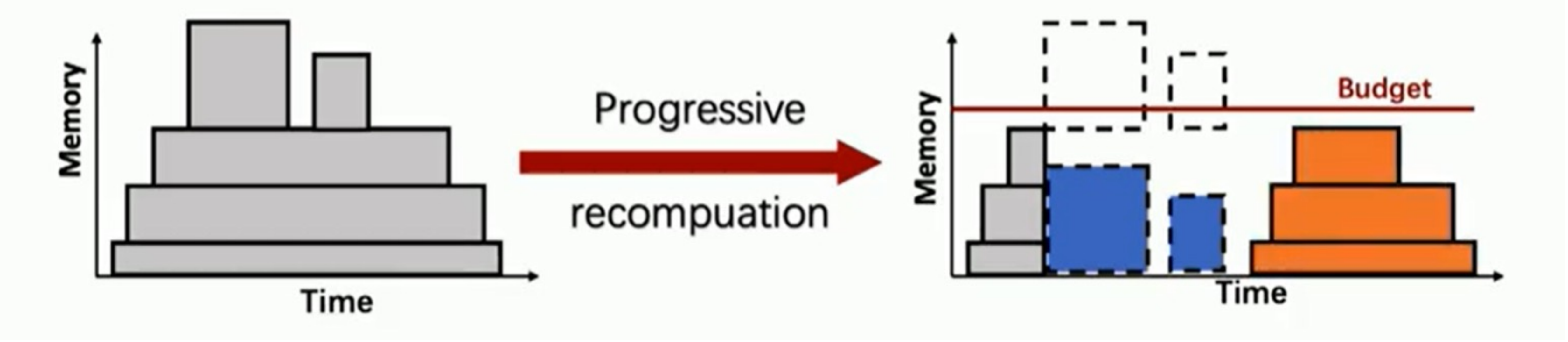
#2 efficient recomputation.

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

Recomputation mechanism

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

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

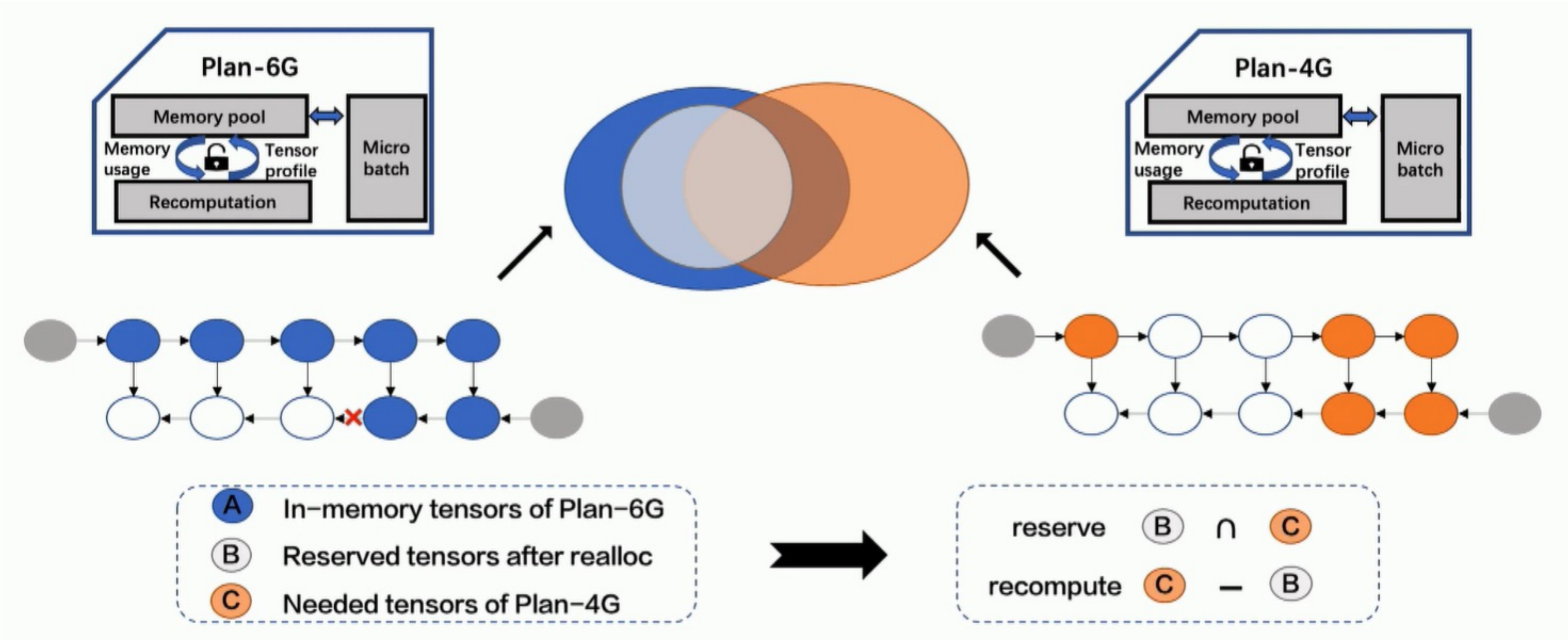


Challenges & Techniques

#3 efficient trainging context switch

How to switch context efficiently with less resource waste?

To avoid resource waste when memory budget changes



Evaluation

Devices and models

Device	SoC	Memory	Model	Params
SN10	SD 855	8 GB	MobileNetV1 [22]	3.3M
VIN3	SD 865	6 GB	MobileNetV2 [53]	2.4M
RN9P	SD 720	6 GB	SqueezeNet [25]	0.8M
RN8	SD 655	4 GB	ResNet50 [19]	23.8M

Table 3: Mobile devices and models used in experiments. “SD”: Qualcomm snapdragon. “SN10": Samsung Note10; “VIN3": Vivo IQOO Neo3. “RN9P": Redmi Note9 Pro. “RN8": Redmi Note8.

Evaluation

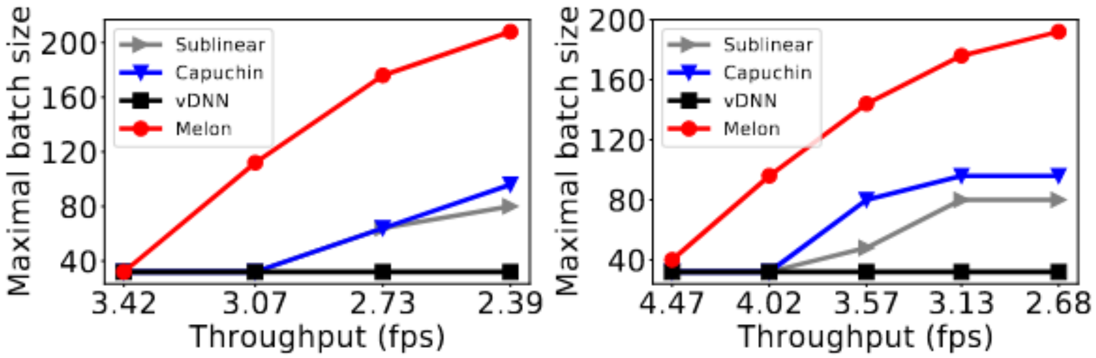
Baselines

- Ideal: infinite memory capacity
- vDNN: swapping. Swap data between memory and disk
- Sublinear: evicts a tensor when memory usage exceeds a threshold
- Capuchin: combines swapping and recomputation.

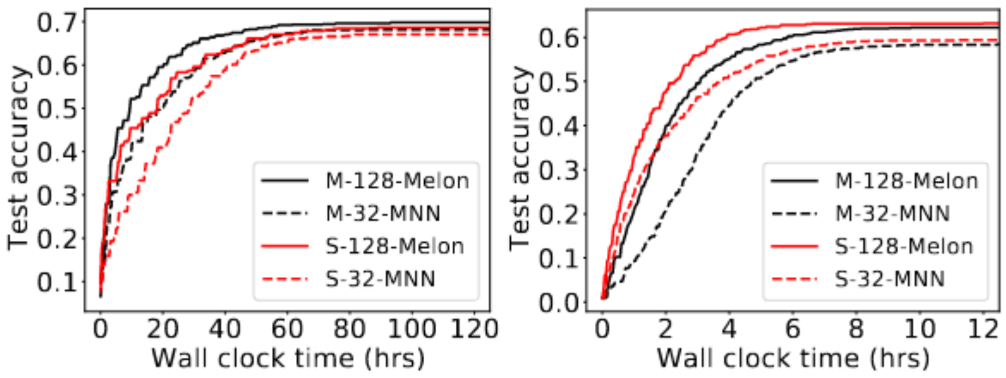
Evaluation

Overall performance

Maximal batch size supported



(a) MobileNetV2 (b) SqueezeNet



(a) Centralized learning (b) Federated learning

Evaluation

Overall performance

Throughput with the same batch size

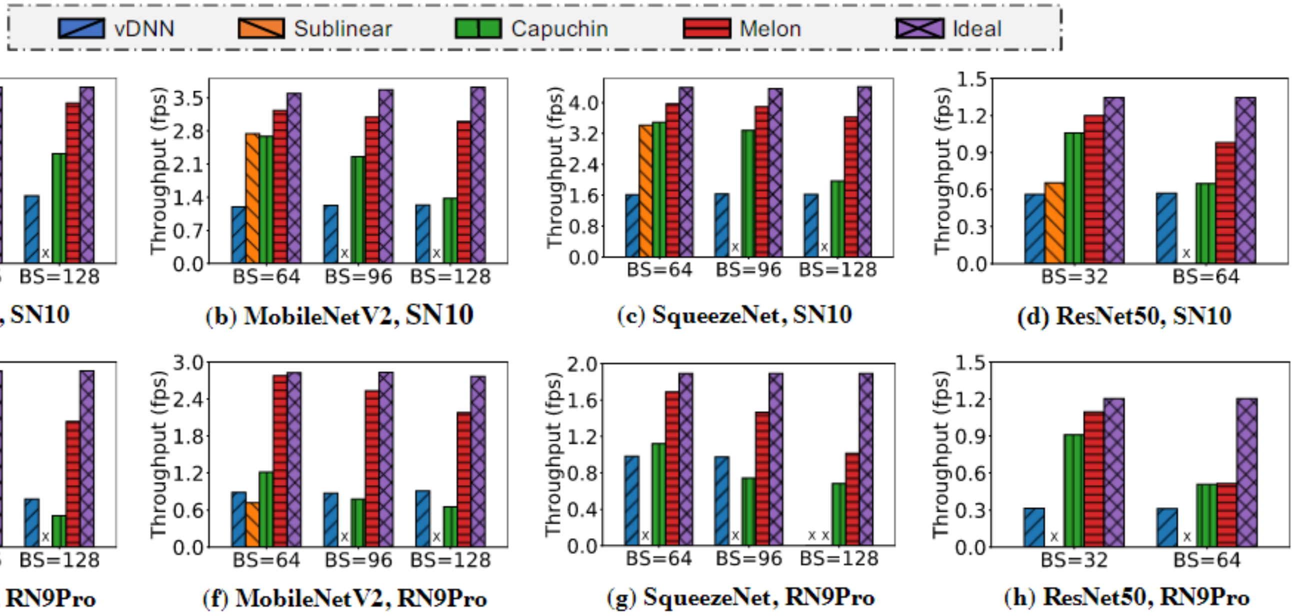
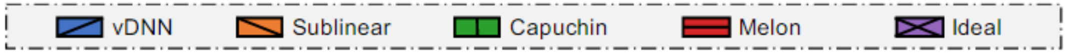


Figure 11: Training throughput (Y-axis) under various batch sizes (X-axis) for different models/devices with batch normalization. “X” means the approach cannot support the training of that batch size. “SN10": Samsung Note10; “RN9Pro": Redmi Note9 Pro.



Evaluation

Overall performance

Throughput with the same batch size

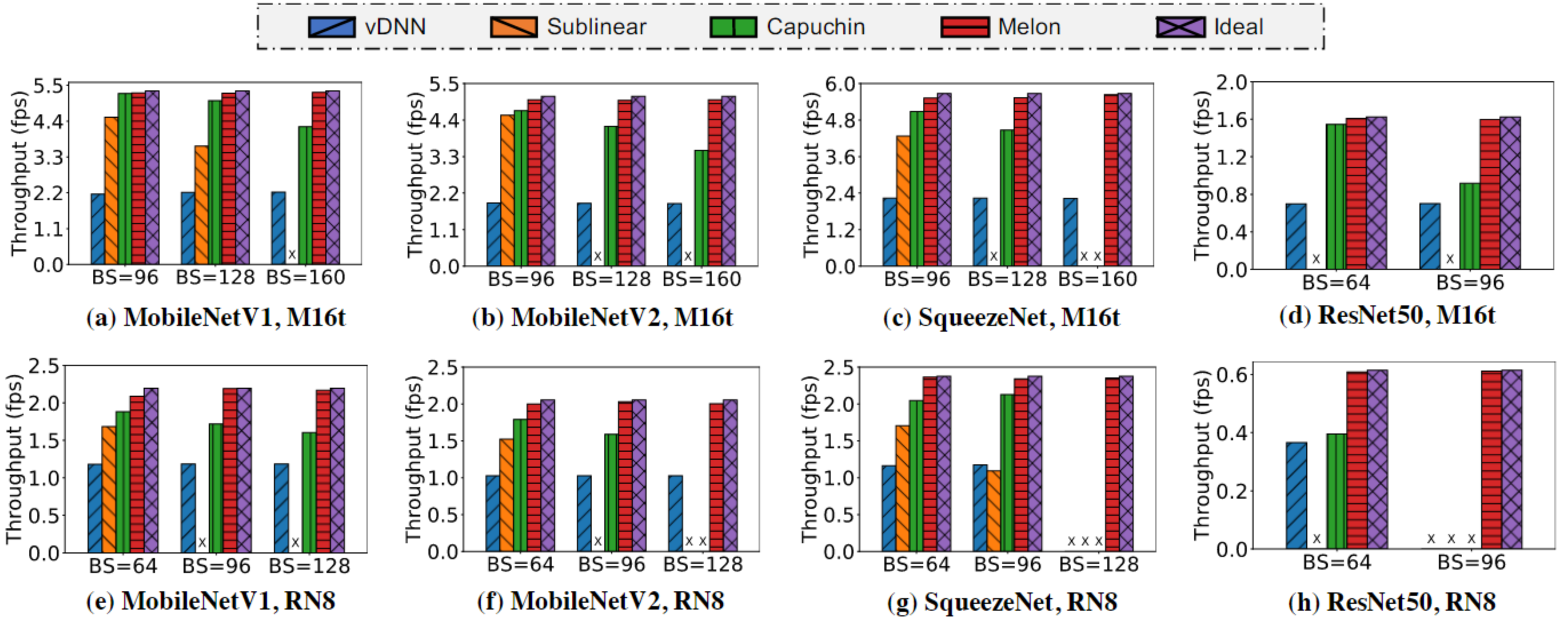
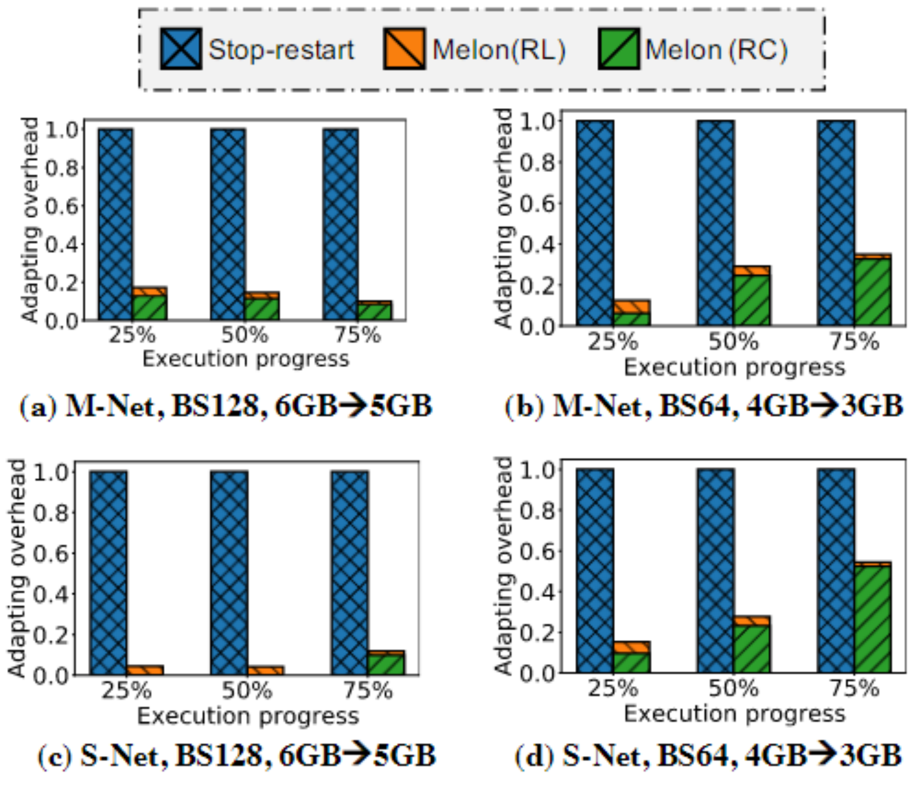


Figure 12: Training throughput (Y-axis) under various batch sizes (X-axis) for different models/devices without batch normalization. "X" means the approach cannot support the training of that batch size. "M16t": Meizu 16t; "RN8": Redmi Note8.

Evaluation

Memory Budget Adaptation



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

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

Evaluation

Energy Consumption

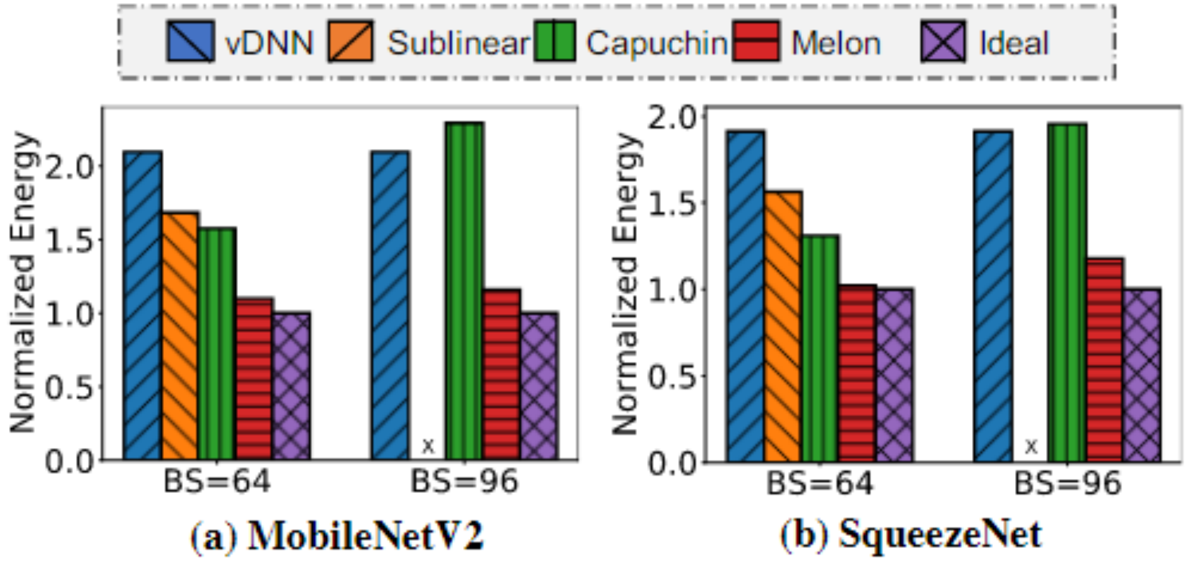
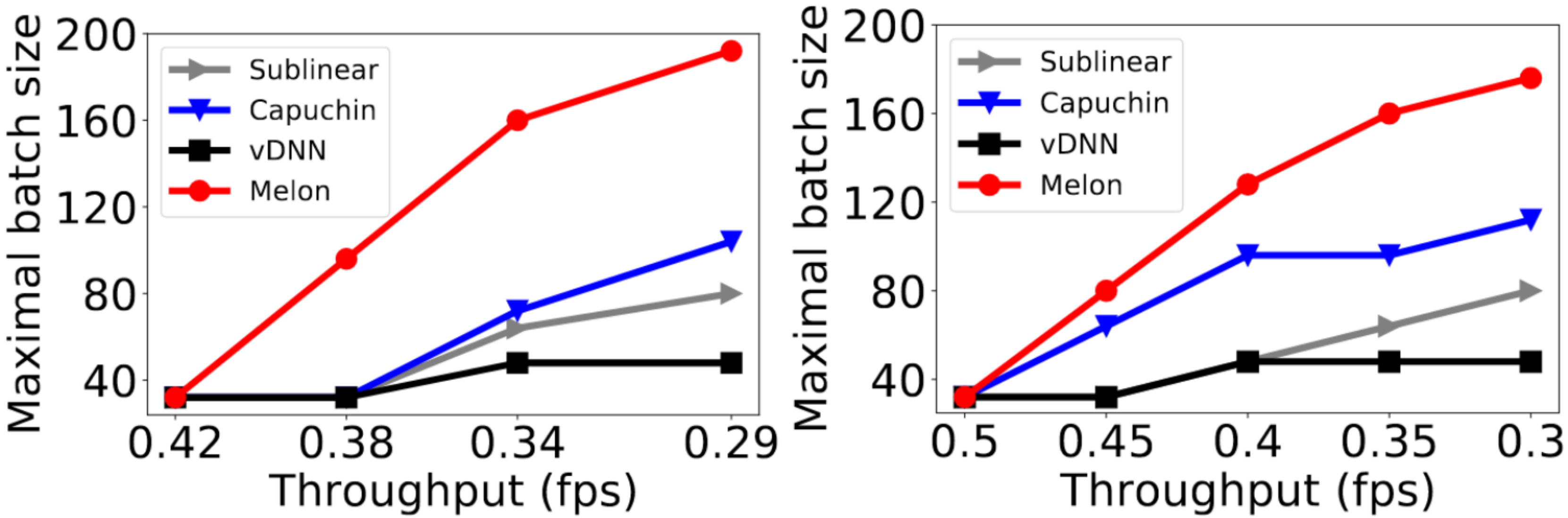


Figure 13: The energy consumption of Melon.

Evaluation

GPU support



(a) MobileNetV2 (b) SqueezeNet

Figure 15: Maximal batch size with GPU support.

Conclusion

Advantages:

1) Break the memory wall on moblie-device, supporting large batch size training on-device.

2) Take memory pool into consideration, efficiently handle the small pieces.

3) Can quickly adapt the change of the memory dudget with low overhead.

Disadvantages:

1) Only support CNN now.

2) The perforfance of GPU training can be improved.

Thank You

2022-12-26

Presented by Ye Wan