ACM MobiSys2022

sigmobile

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

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✉]

([*]co-primary, [✉]corresponding)

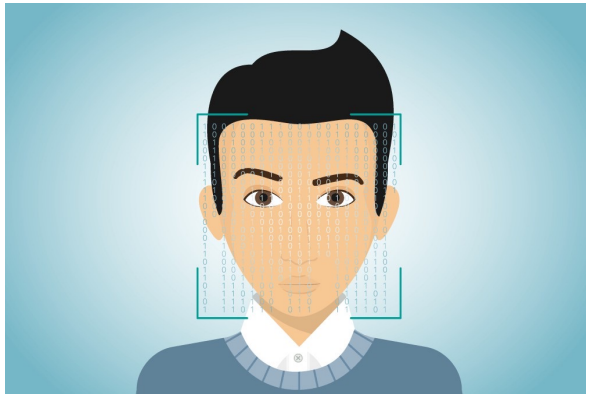
[1]Key Lab of High Confidence Software Technologies (Peking University)
[2]State Key Laboratory of Networking and Switching Technology (BUPT)
[3]Institute for AI Industry Research (AIR), Tsinghua University

北京大学 PEKING UNIVERSITY

北京郵電大學 Beijing University of Posts and Telecommunications

AIR 清華大學 智能产业研究院 Institute for AI Industry Research, Tsinghua University

DNN is a key component for mobile app



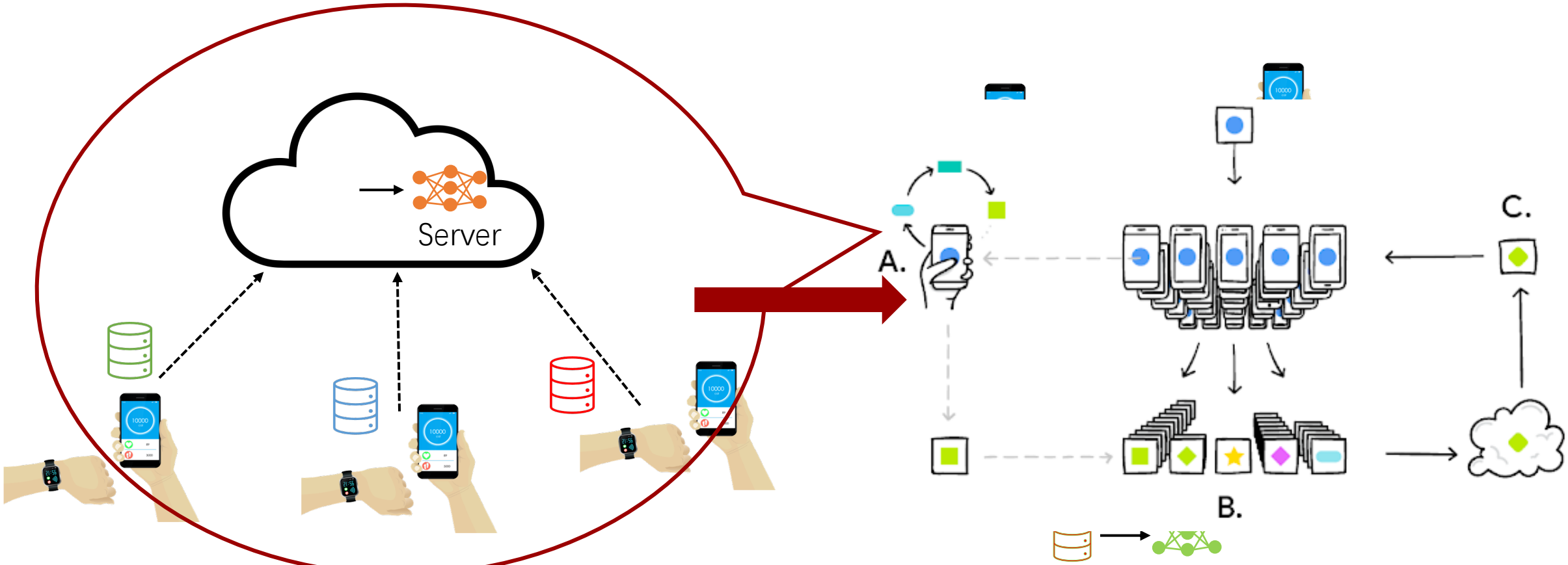
Face recognition

Voice recognition

Augmented Reality

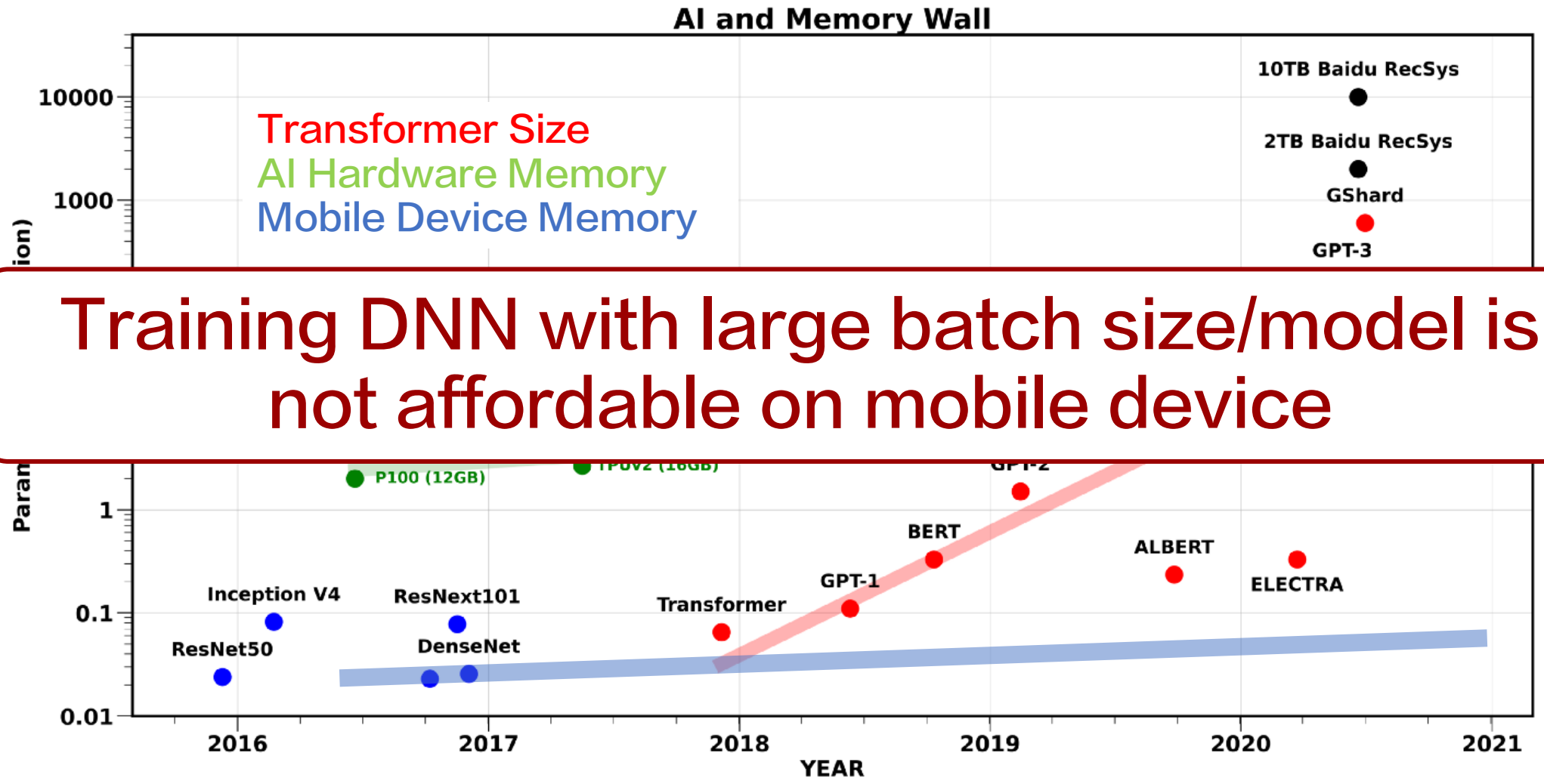
Mainly DNN inference now

Emerging on-device learning



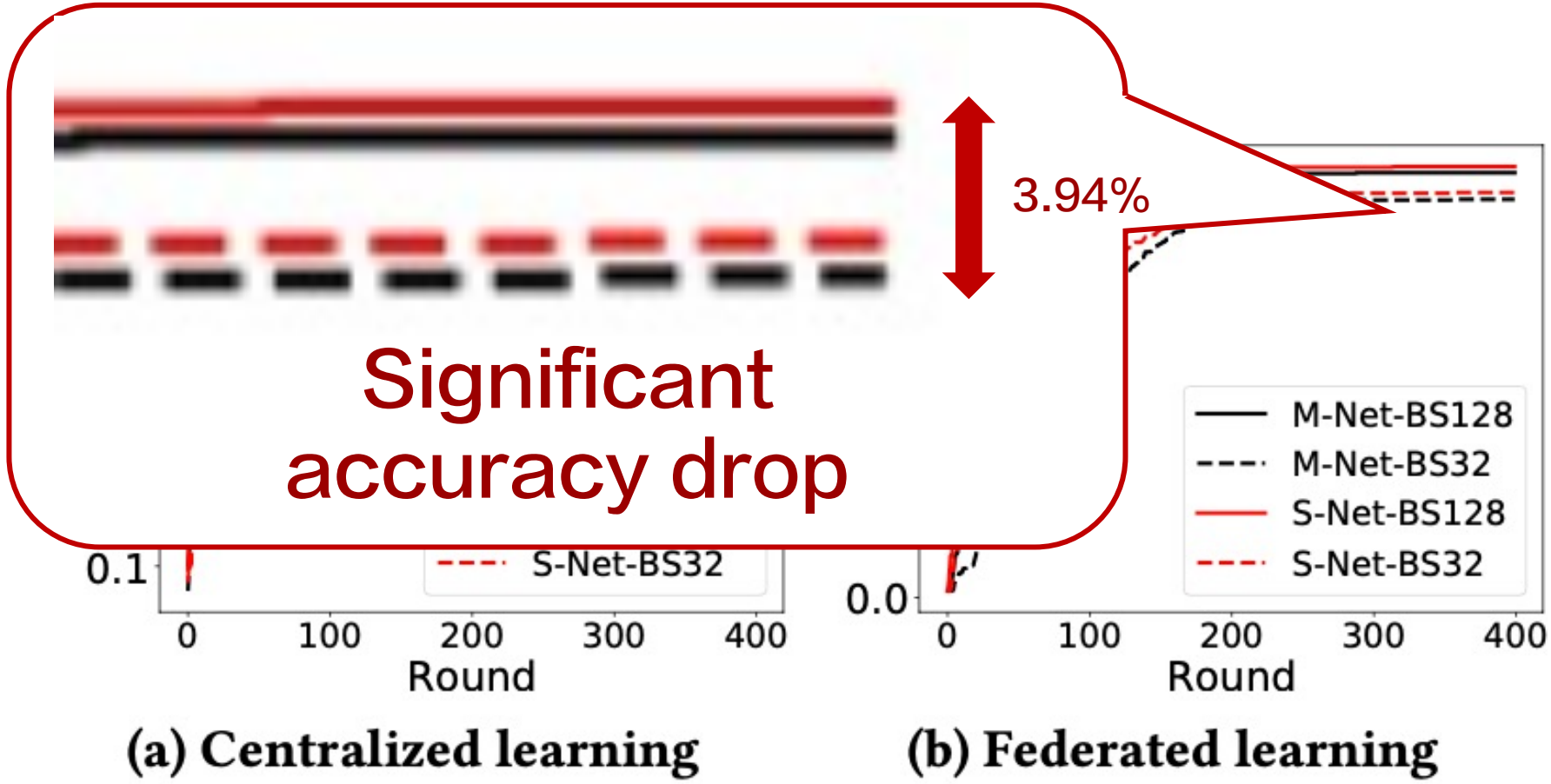
Whether the training of modern DNN is affordable on mobile devices?

Preliminaries: memory wall

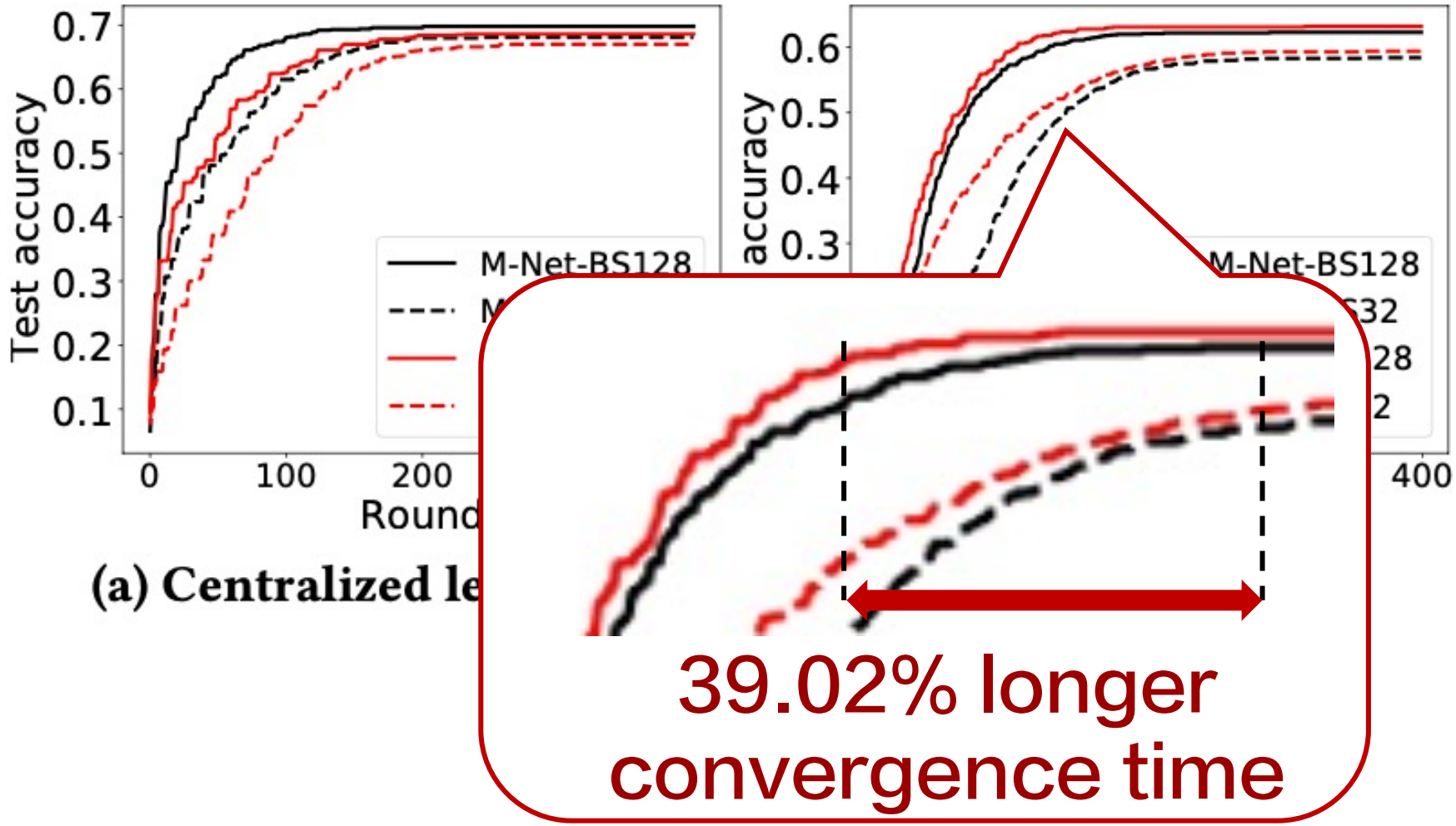


Training DNN with large batch size/model is not affordable on mobile device

Preliminaries: memory wall



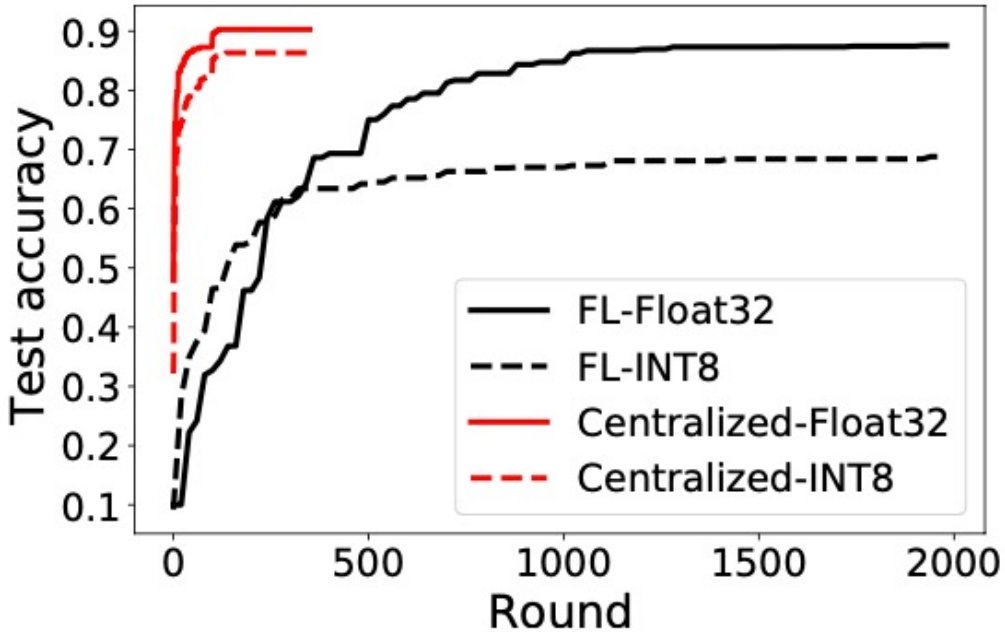
Preliminaries: memory wall



(a) Centralized le

Preliminaries: existing techniques

- Model & gradients compression



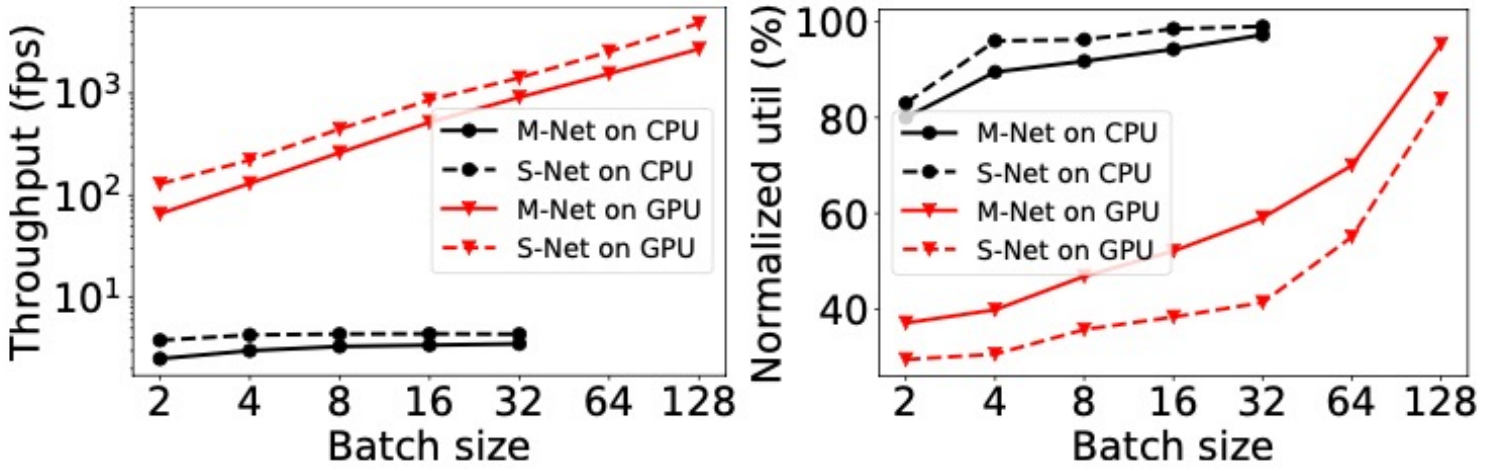
Significant accuracy drop

Preliminaries: existing techniques

- Model & gradients compression
- Host-device memory swapping
  - Performance is blocked significantly by I/O speed of mobile devices

Preliminaries: existing techniques

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



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

Preliminaries: existing techniques

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

Preliminaries: existing techniques

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

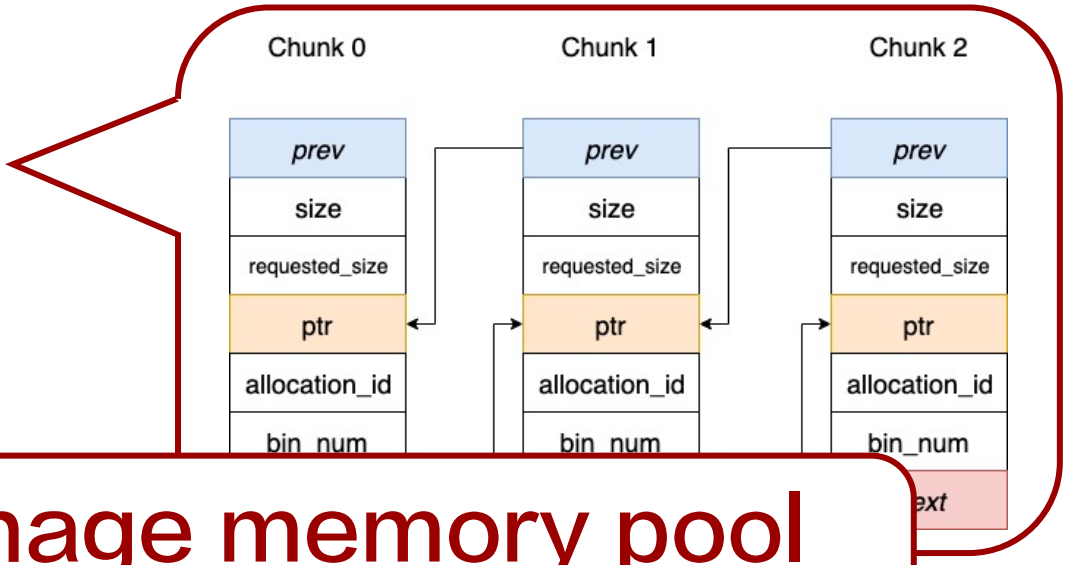
How to break the memory wall for on-device learning?

Challenge#1: efficient memory management

Memory pool is widely adopted

PyTorch TensorFlow

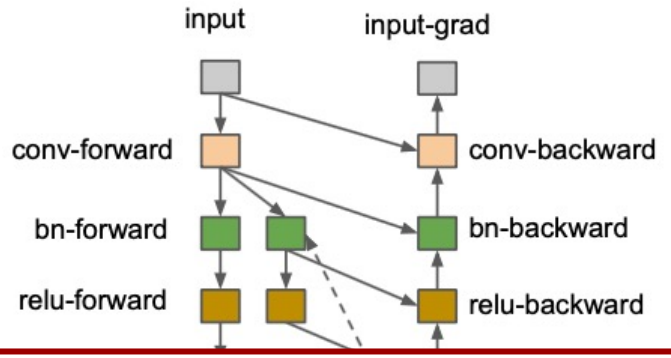
mxnet MNN Mobile Neural Network

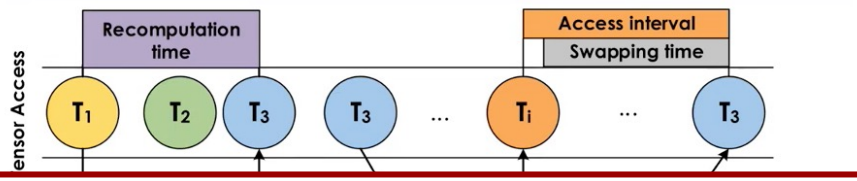


How to manage memory pool efficiently for DNN training?

Challenge#2: efficient recomputation

Current recomputation ignores impact of memory pool





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

Challenge#3: efficient training context switch

Resource competition due to other apps

Available memory

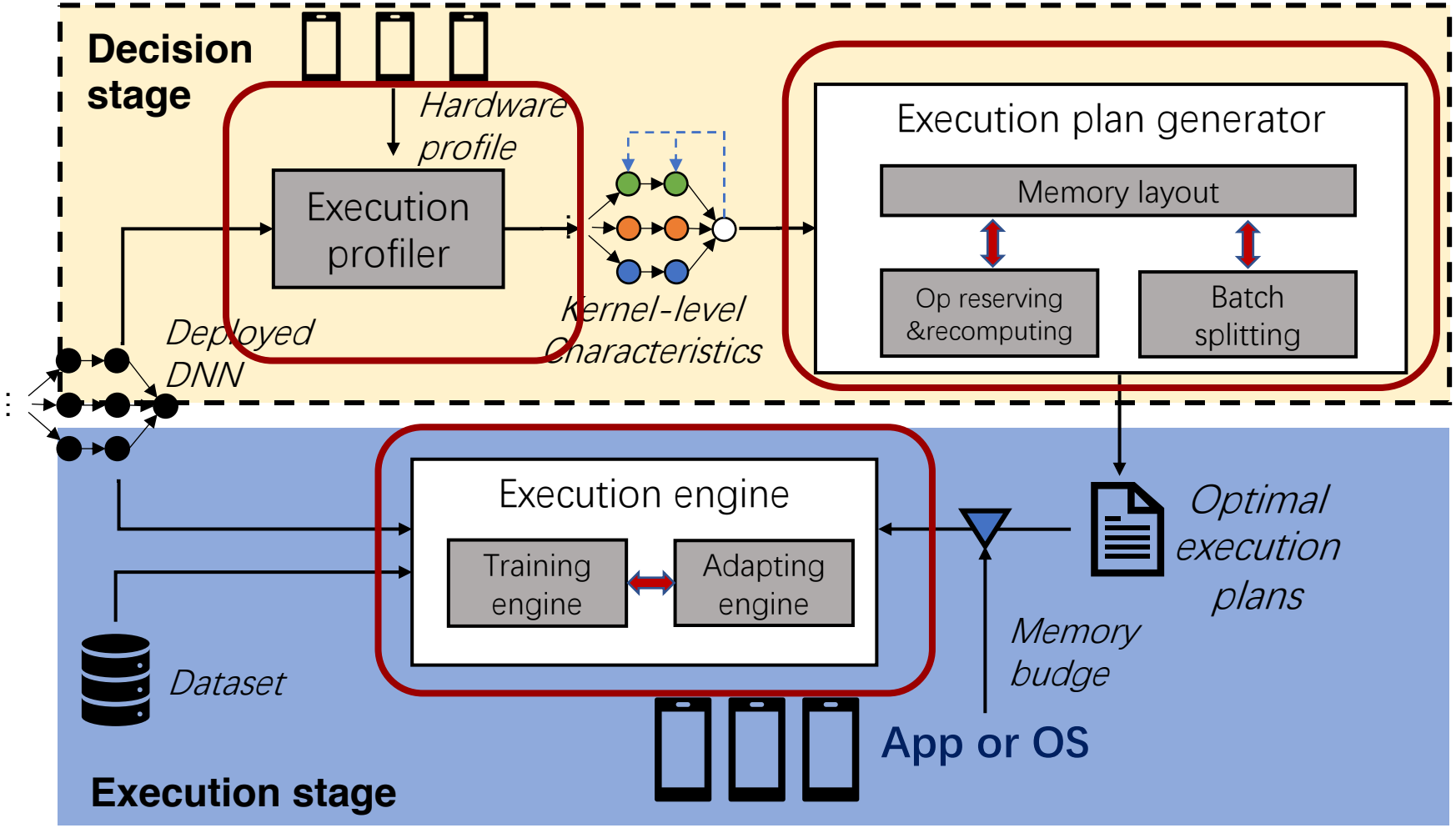
shrinks

Old training progress

Reserve previous results rather than dropping them

How to switch context efficiently with less resource waste?

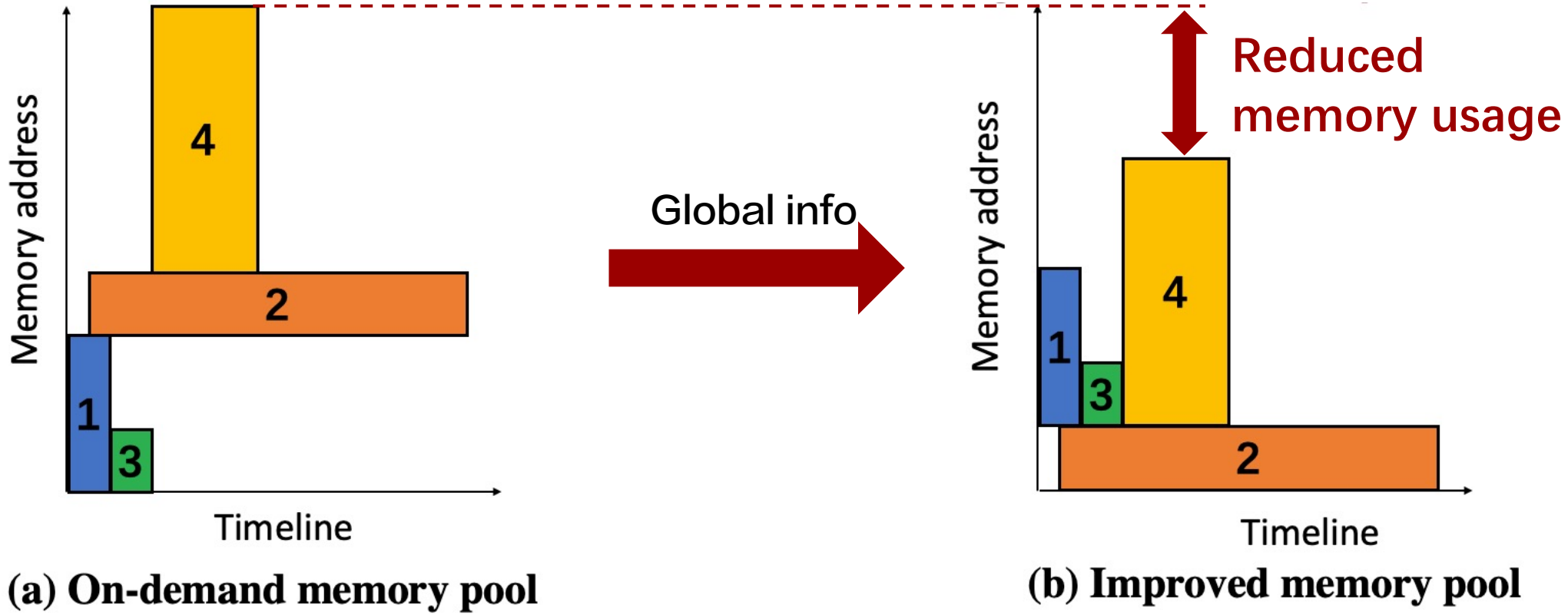
Melon: design overview



Tensor lifetime-aware memory pool

Heuristic

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



(a) On-demand memory pool

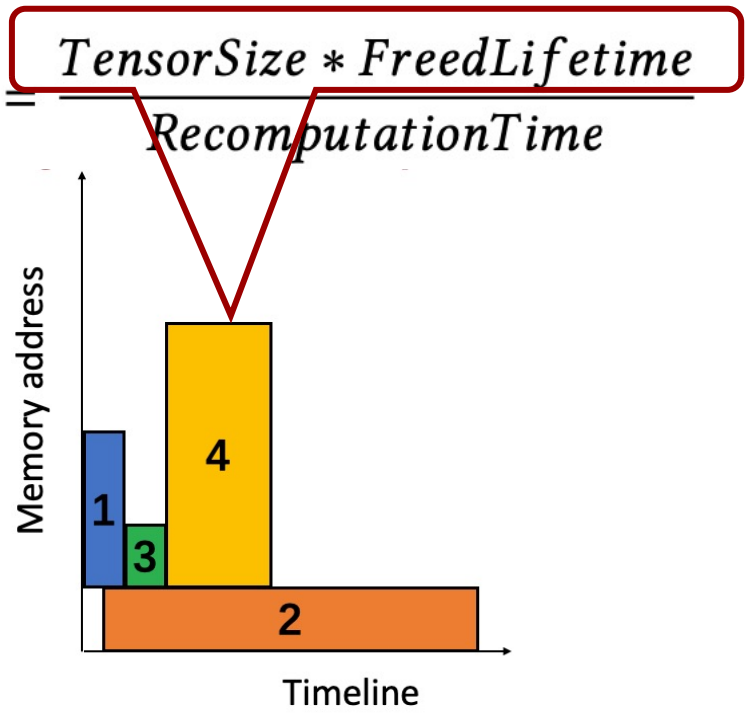
(b) Improved memory pool

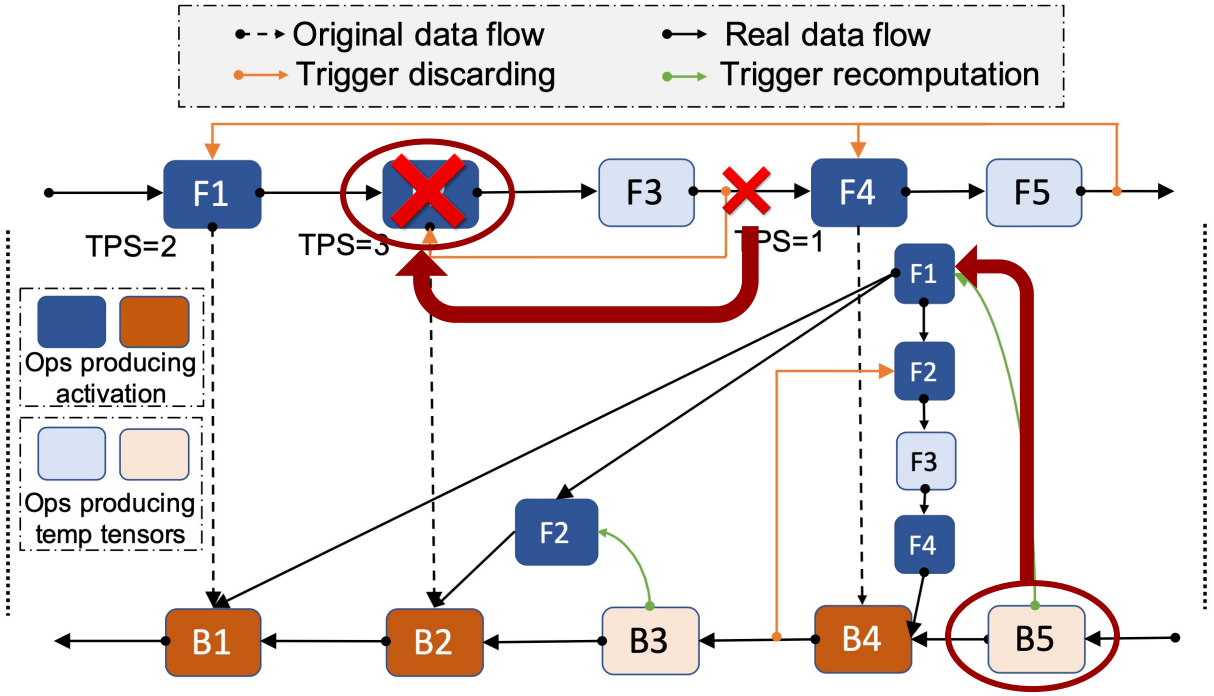
Memory-calibrated progressive recomputation

Recomputation mechanism

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

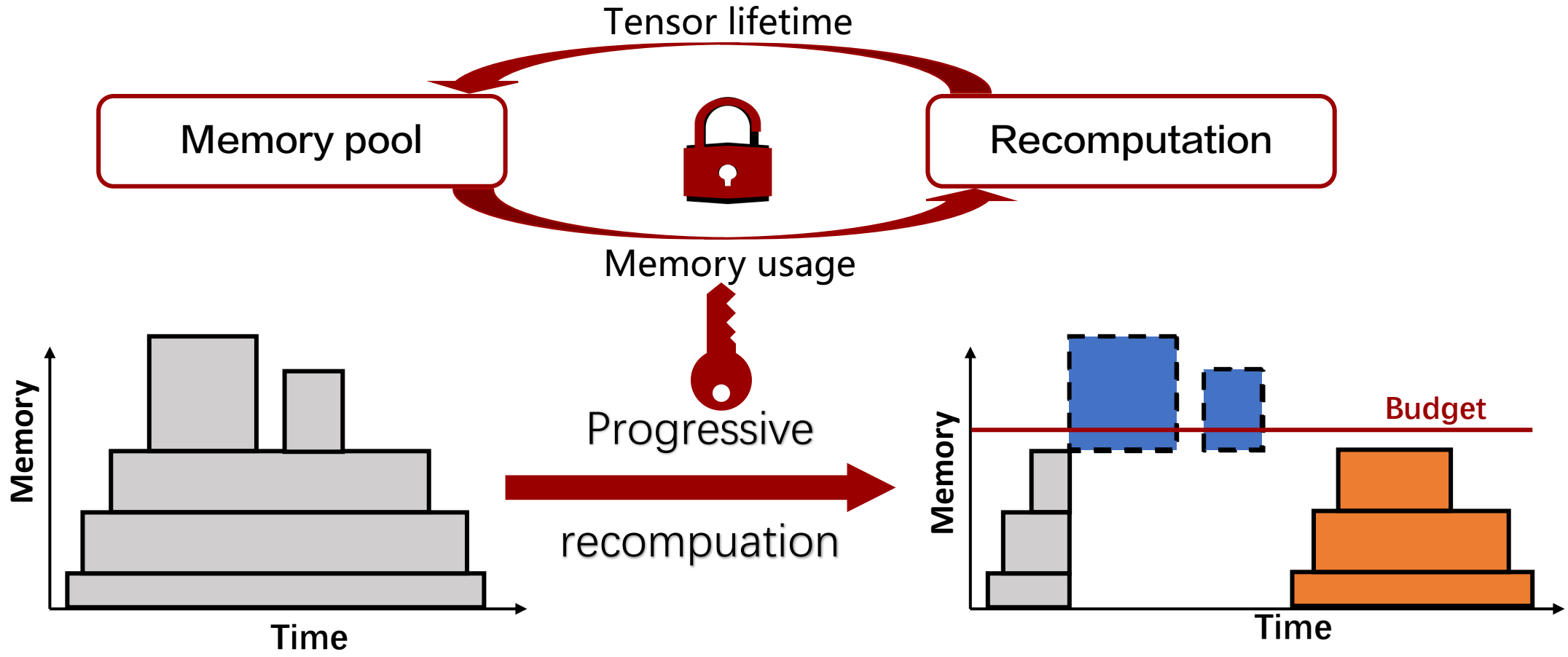
$$TPS = \frac{TensorSize * FreedLifetime}{RecomputationTime}$$





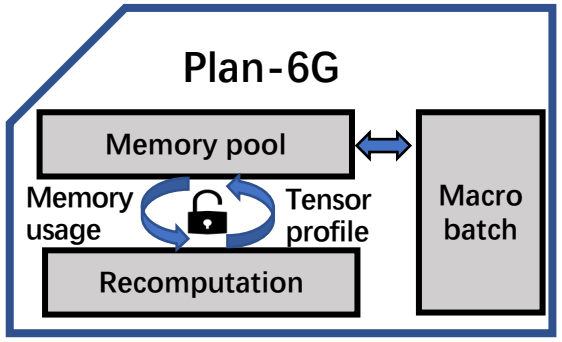
Memory-calibrated progressive recomputation

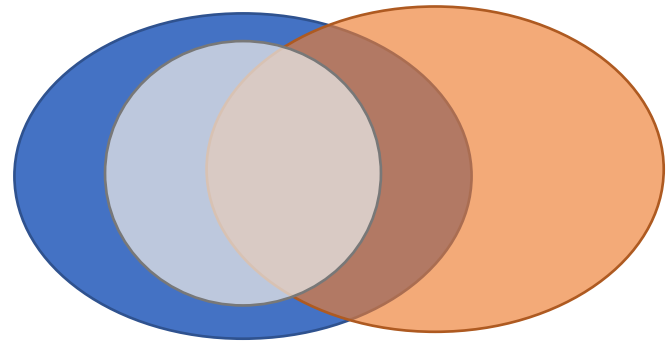
Take memory pool into consideration

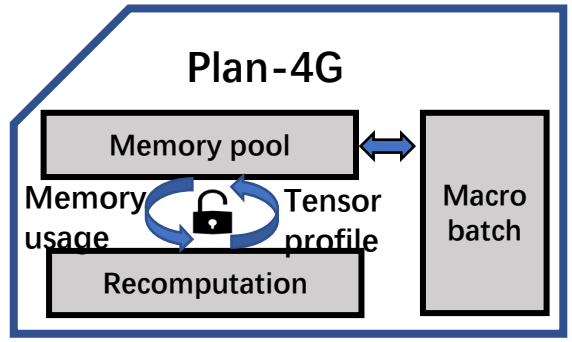


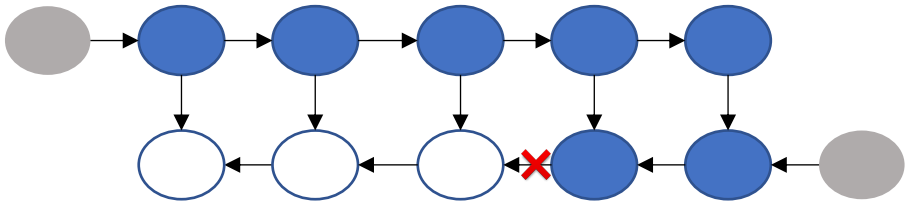
On-the-fly memory budget adapting

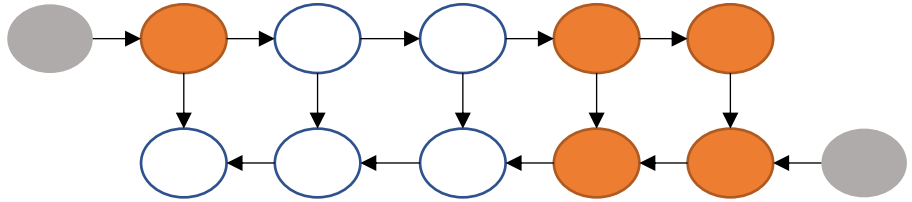
To avoid resource waste when memory budget changes











A In-memory tensors of Plan-6G

B Reserved tensors after realloc

C Needed tensors of Plan-4G

reserve B ∩ C

recompute C − B

Evaluation: setting

Devices and models:

Device	SoC	Memory	Model	Parameters
Samsung Note10	SnapDragon 855	8GB	MobileNetV1	3.3M
Meizu 16T	SnapDragon 855	6GB	MobileNetV2	2.4M
Redmi Note9 Pro	SnapDragon 720	6GB	SqueezeNet	0.8M
Redmi Note8	SnapDragon 655	4GB	ResNet50	23.8M

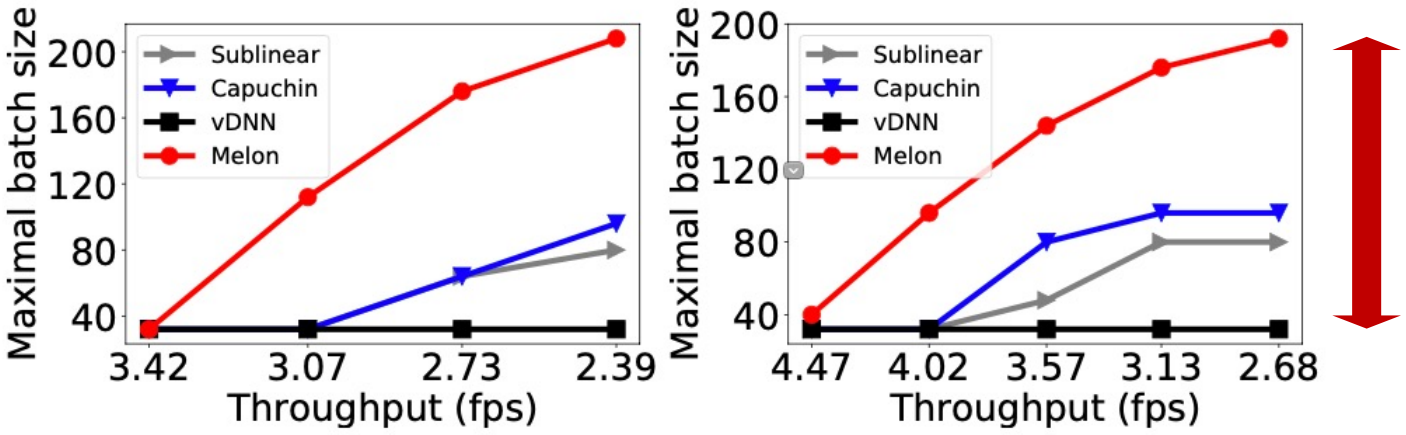
Baselines:

- vDNN[1]: host-device memory swapping
- Sublinear[2]: checkpoint-based recomputation
- Capuchin[3]: tensor-based swapping and recomputation
- Ideal: ideal case that supposes memory is infinite

[1] Rhu et al. vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design. MICRO' 16.
[2] Chen et al. Training deep nets with sublinear memory cost. arXiv' 16.
[3] Peng et al. Capuchin: Tensor-based gpu memory management for deep learning. ASPLOS' 20

Evaluation: end2end

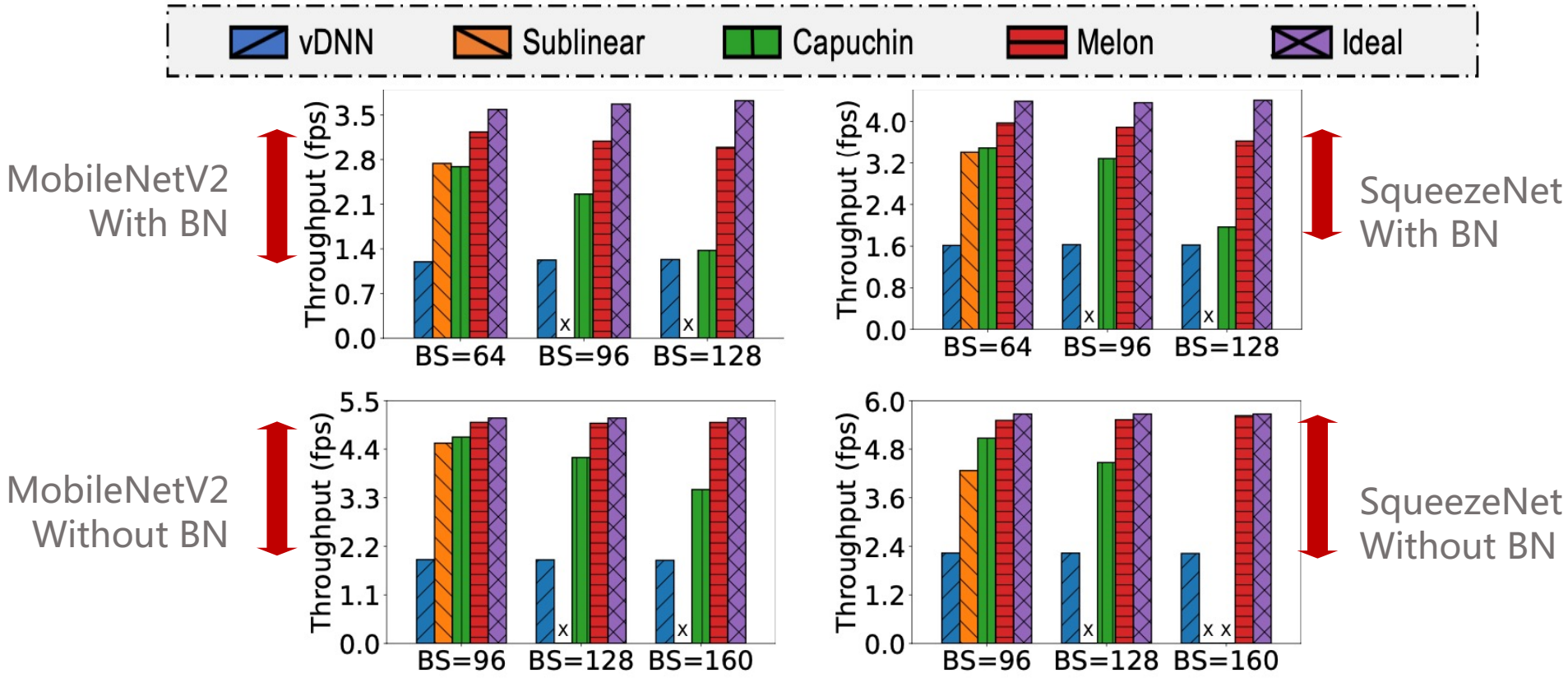
- Achieve larger batch size compared with baselines with same throughput



(a) MobileNetV2 (b) SqueezeNet

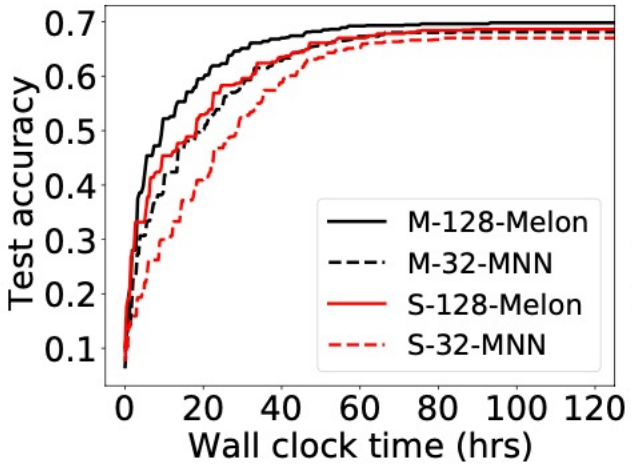
Evaluation: end2end

- Achieve much larger batch size with same throughput
- Achieve much higher throughput with same batch size

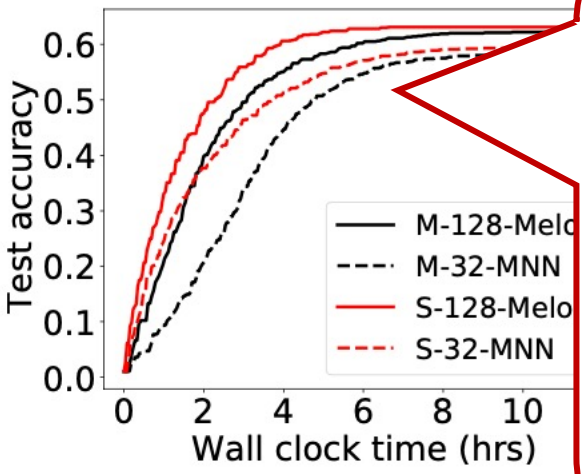


Evaluation: end2end

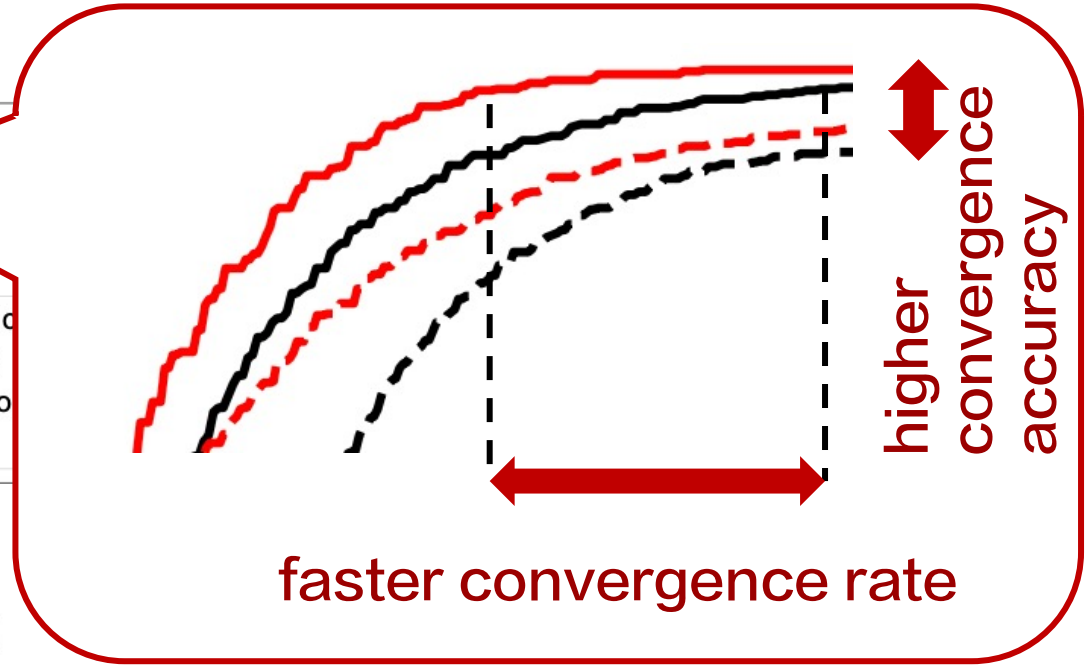
- Achieve much larger batch size with same throughput
- Achieve much higher throughput with same batch size
- Achieve faster convergence rate and higher convergence accuracy



(a) Centralized learning

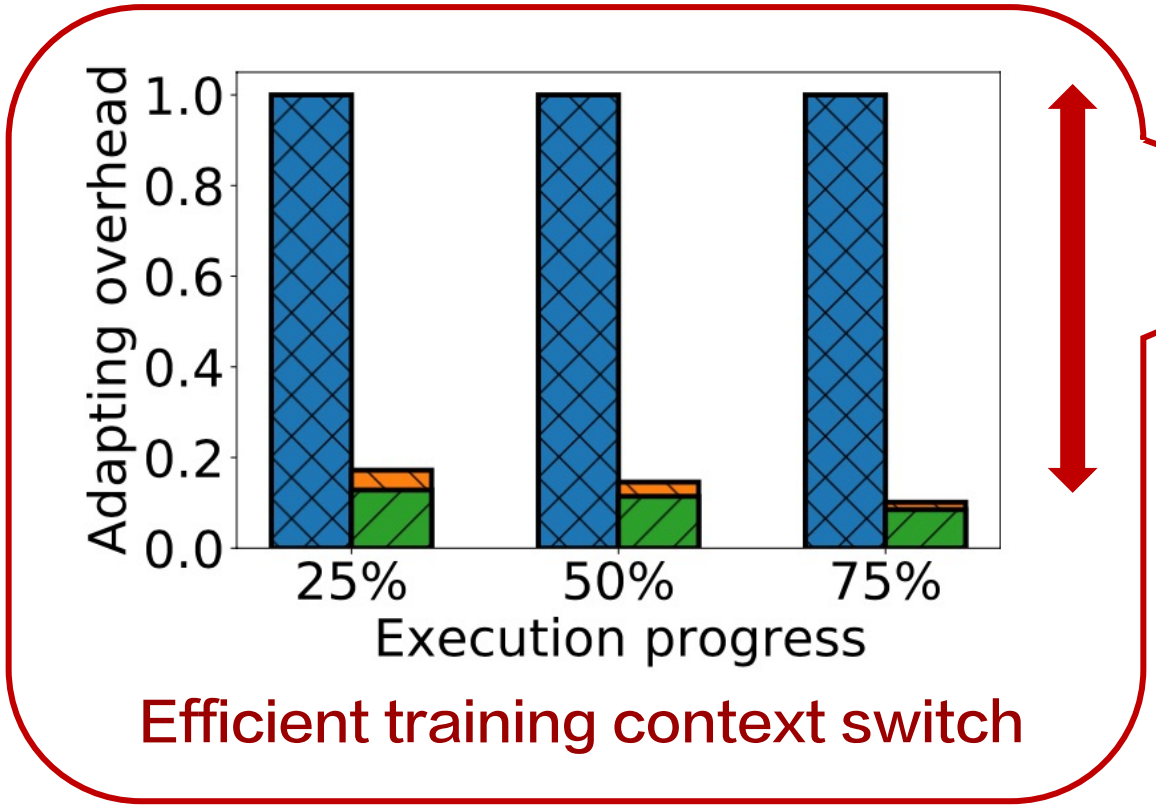


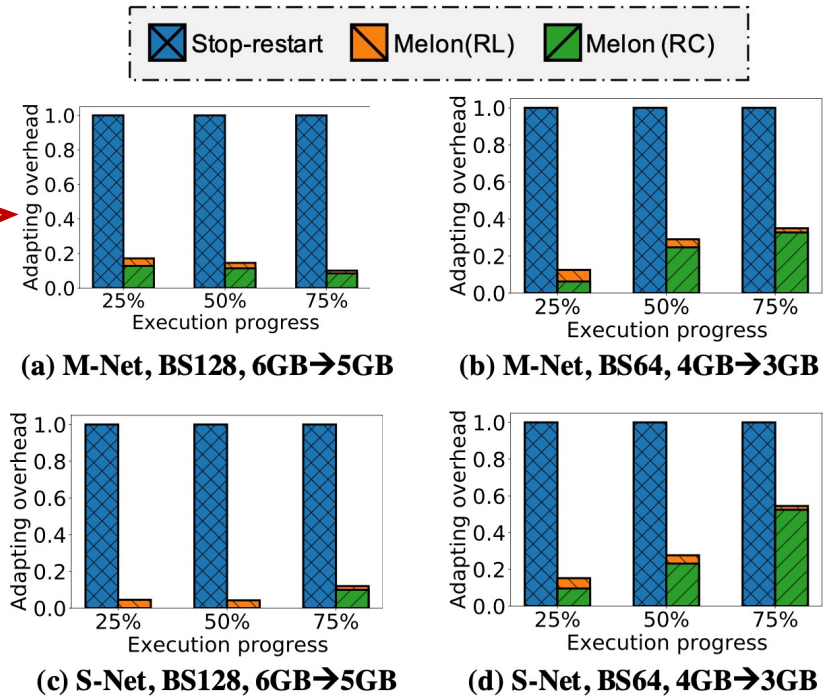
(b) Federated learning



Evaluation: adaptation

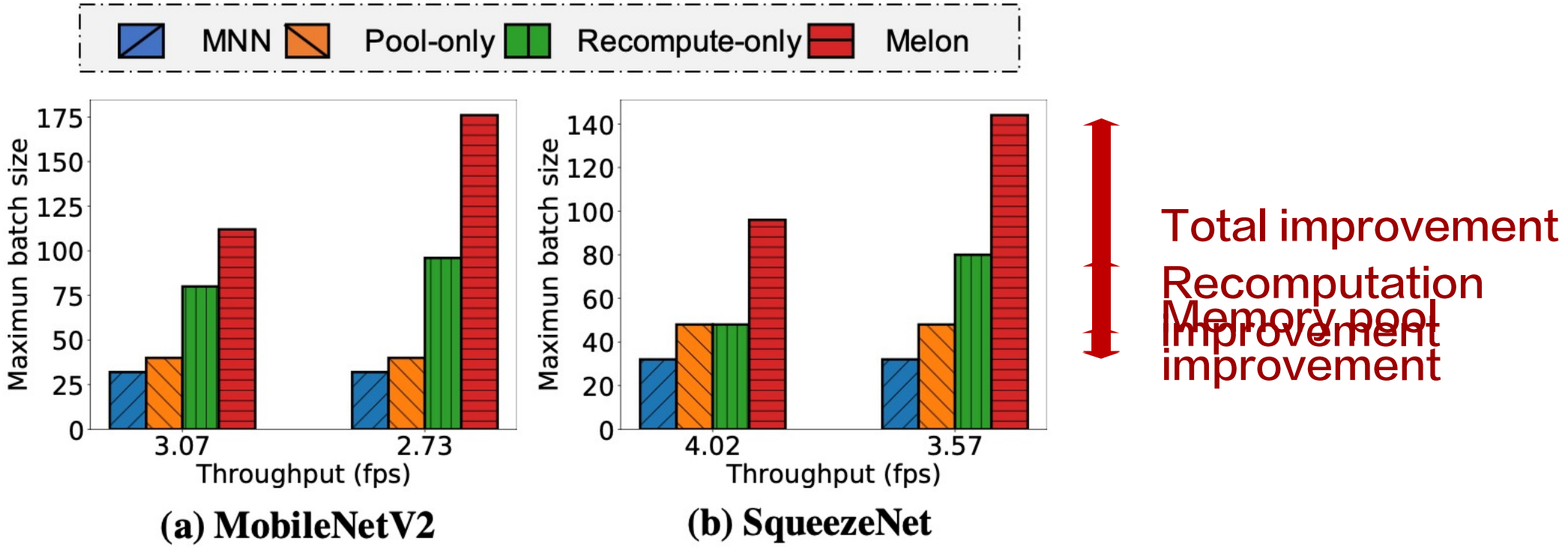
- Avoid less resource waste when memory budget changes





Evaluation: ablation

- Each component contributes to the improvement



Conclusion

- Melon: an efficient on-device learning framework to break the memory wall
  - Tensor lifetime-aware memory pool
  - Memory-calibrated progressive recomputation
  - On-the-fly memory budget adapting
- Implement a prototype and evaluate its efficiency
- Open source: https://github.com/qipengwang/Melon

Code:
https://github.com/qipengwang/Melon

wangqipeng@stu.pku.edu.cn

Artifact Available Artifact Evaluated-Functional Results Reproduced