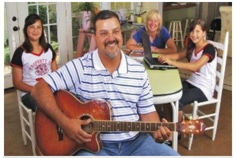
Lecture 14: Model Serving System

CSE599G1: Spring 2017

Deep Learning Applications

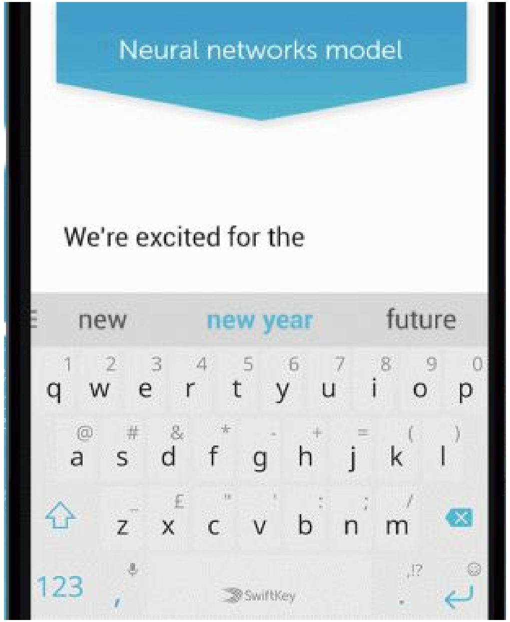
"That drink will get you to 2800 calories for today"

"I last saw your keys in the store room"



"Remind Tom of the party"

"You're on page 263 of this book"

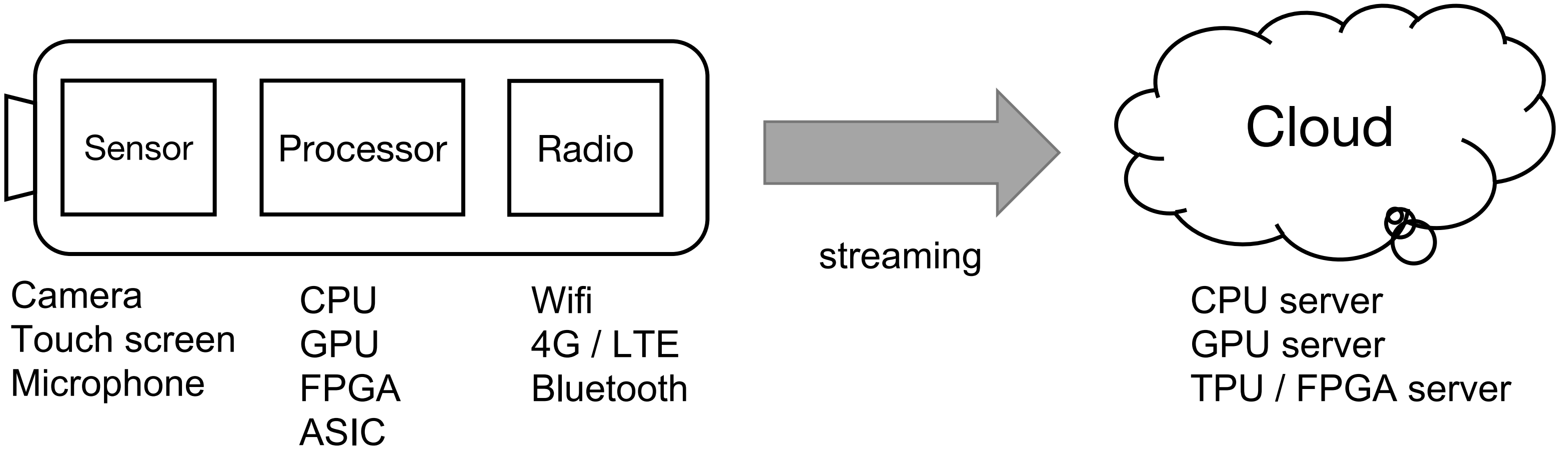


Intelligent assistant

Surveillance / Remote assistance

Input keyboard

Runtime Environment



Resource usage for a continuous vision app

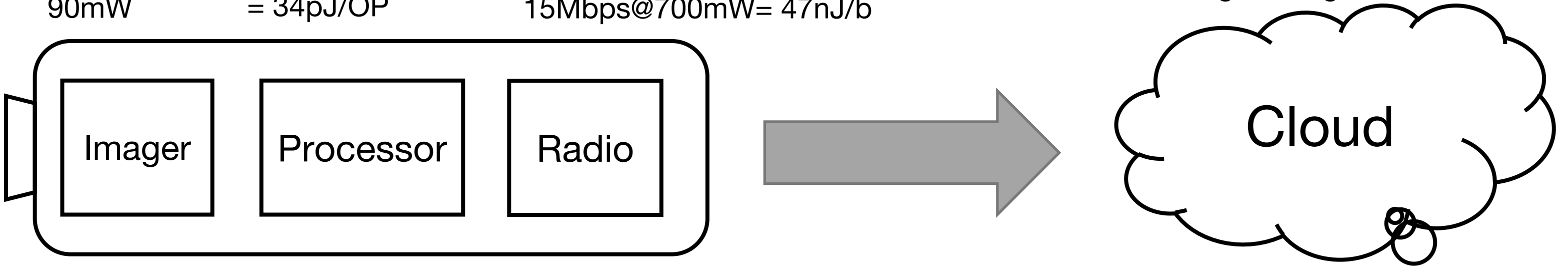
Omnivision
OV2740
90mW

Tegra K1 GPU
290GOPS@10W
= 34pJ/OP

Qualcomm SD810 LTE
>800mW
Atheros 802.11 a/g
15Mbps@700mW= 47nJ/b

Amazon EC2

CPU	c4.large	2x400GFLOPS	$0.1/h
GPU	g2.2xlarge	2.3TFLOPS	$0.65/h



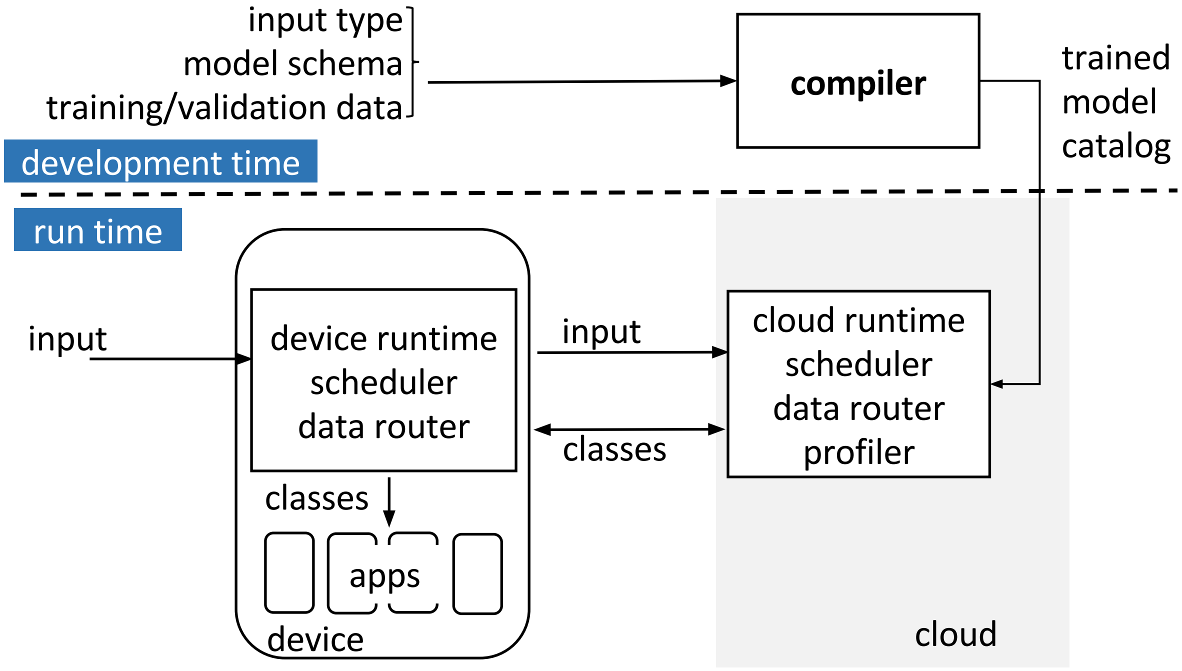
Workload	Deep learning **300GFLOPS** @ 30GFLOPs/frame, 10fps	
Budget	Device power 30% of 10Wh for 10h = 300mW	Cloud cost $10 person/year
Compute power	**9GFLOPS**	**3.5GFLOPS** (GPU) / **8GFLOPS** (CPU)

Huge gap between workload and budget

Model Serving System Constraints

- Latency constraint
  - Batch size cannot be as large as possible when executing in the cloud
  - Can only run lightweight model in the device
- Resource constraint
  - Battery limit for the device
  - Memory limit for the device
  - Cost limit for using cloud
- Accuracy constraint
  - Some loss is acceptable by using approximate models
  - Multi-level QoS

System overview



Outline

- Model compression
- Serving Backend
- Runtime scheduling between device and cloud

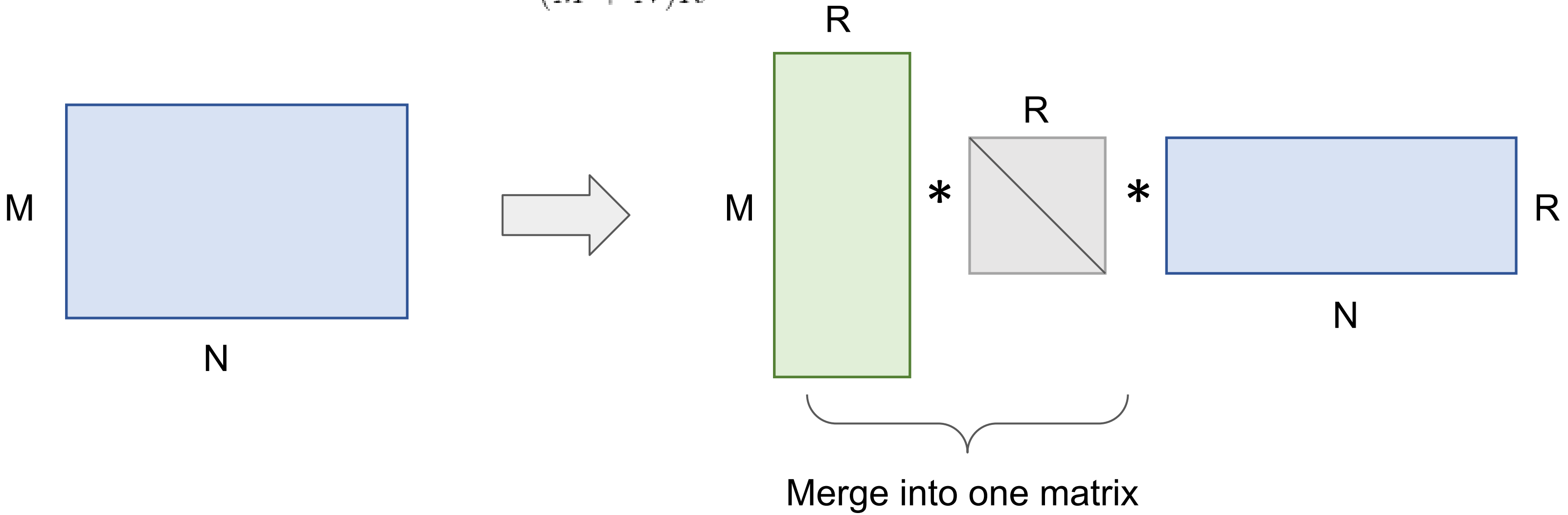
Model Compression

- Tensor decomposition
- Quantization
- Smaller model

Matrix Decomposition

Fully-connected layer

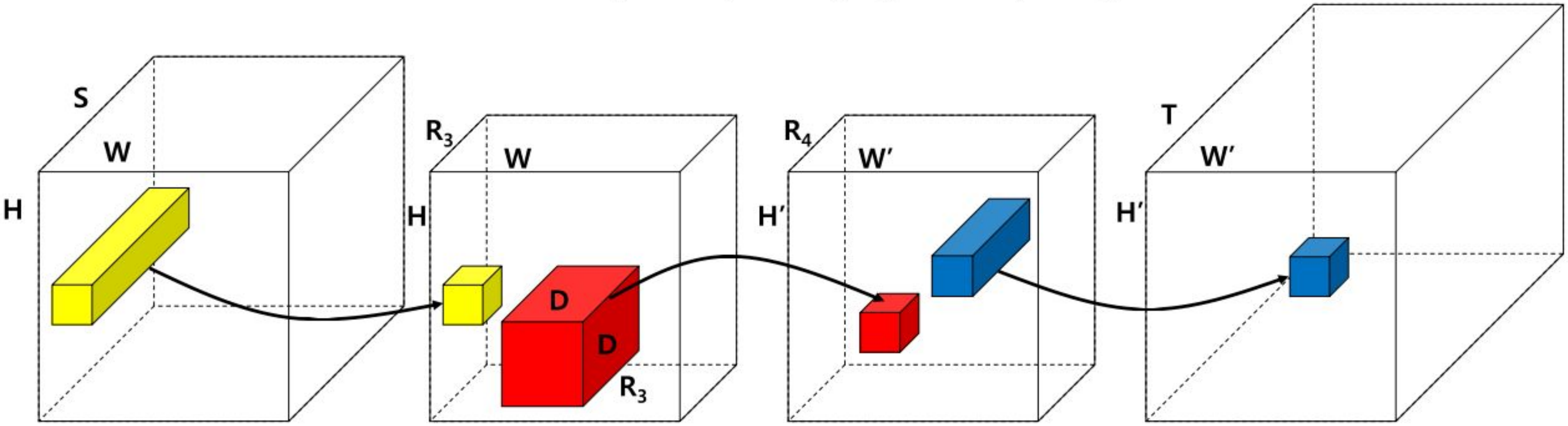
- Memory reduction: $\frac{MN}{(M+N)R}$
- Computation reduction: $\frac{MN}{(M+N)R}$



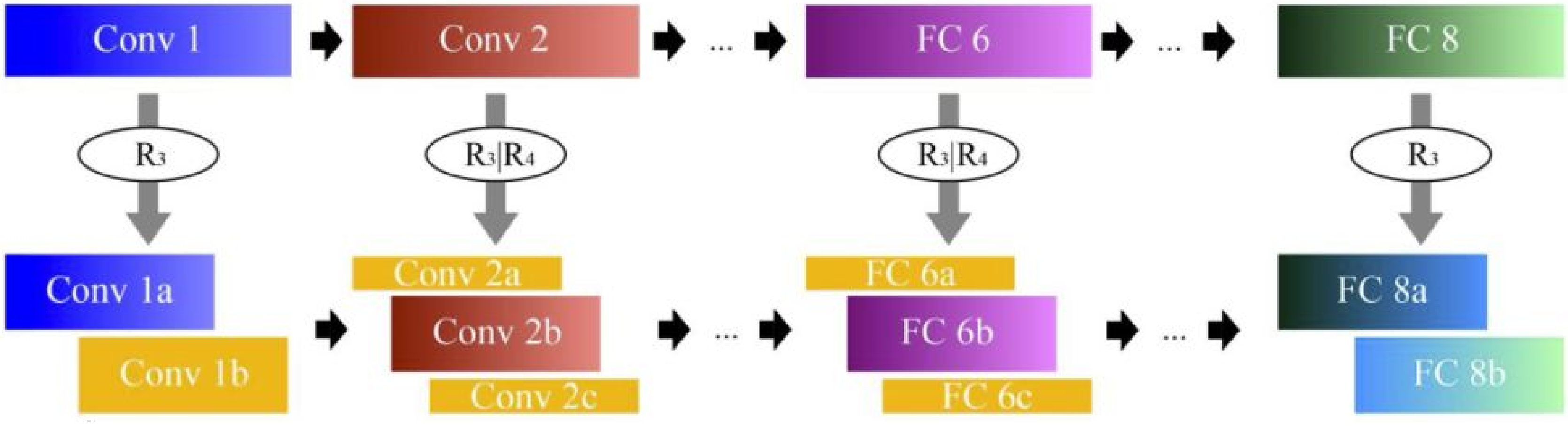
Tensor Decomposition

Convolutional layer

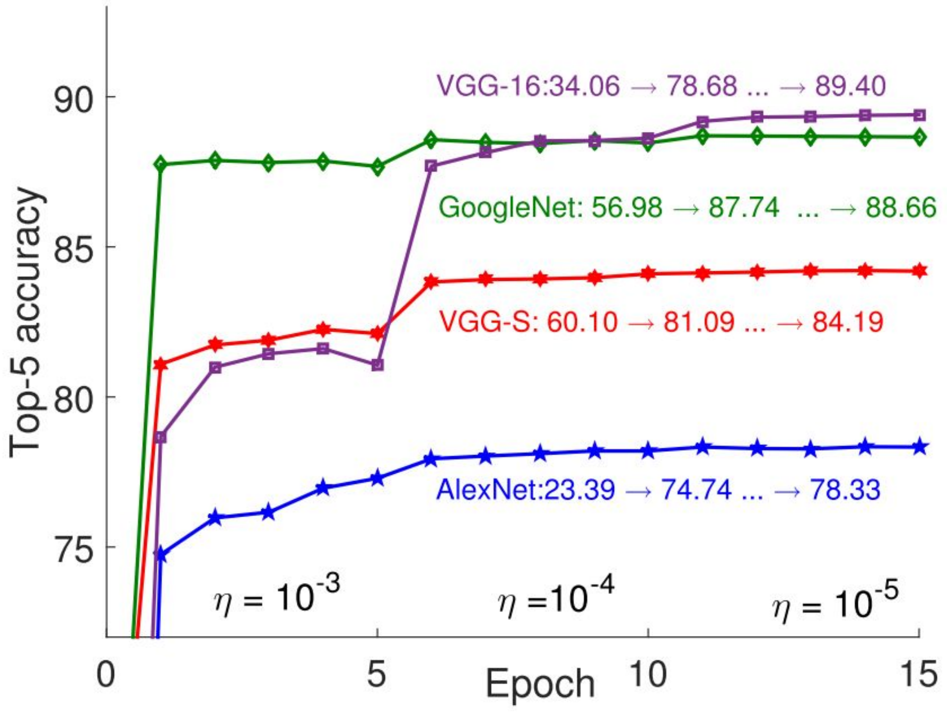
- Memory reduction: $\dfrac{D^2ST}{SR_3 + D^2R_3R_4 + TR_4}$
- Computation reduction: $\dfrac{D^2STH'W'}{SR_3HW + D^2R_3R_4H'W' + TR_4H'W'}$



Decompose the entire model



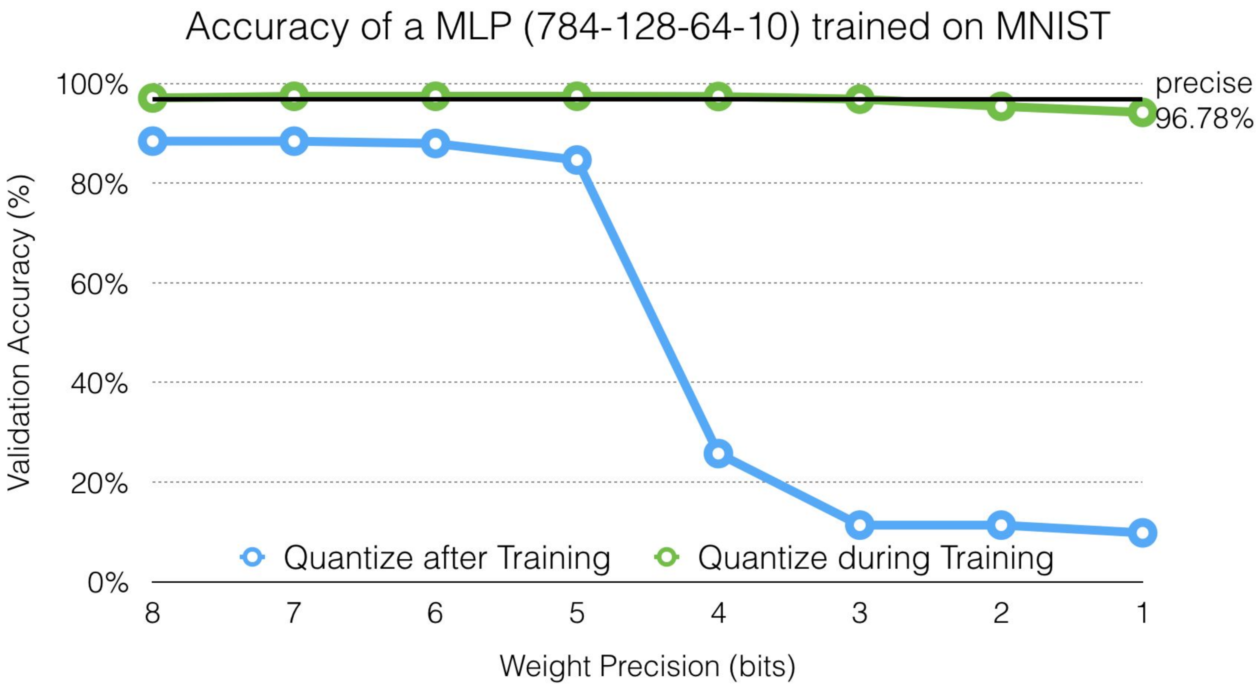
Fine-tuning after decomposition



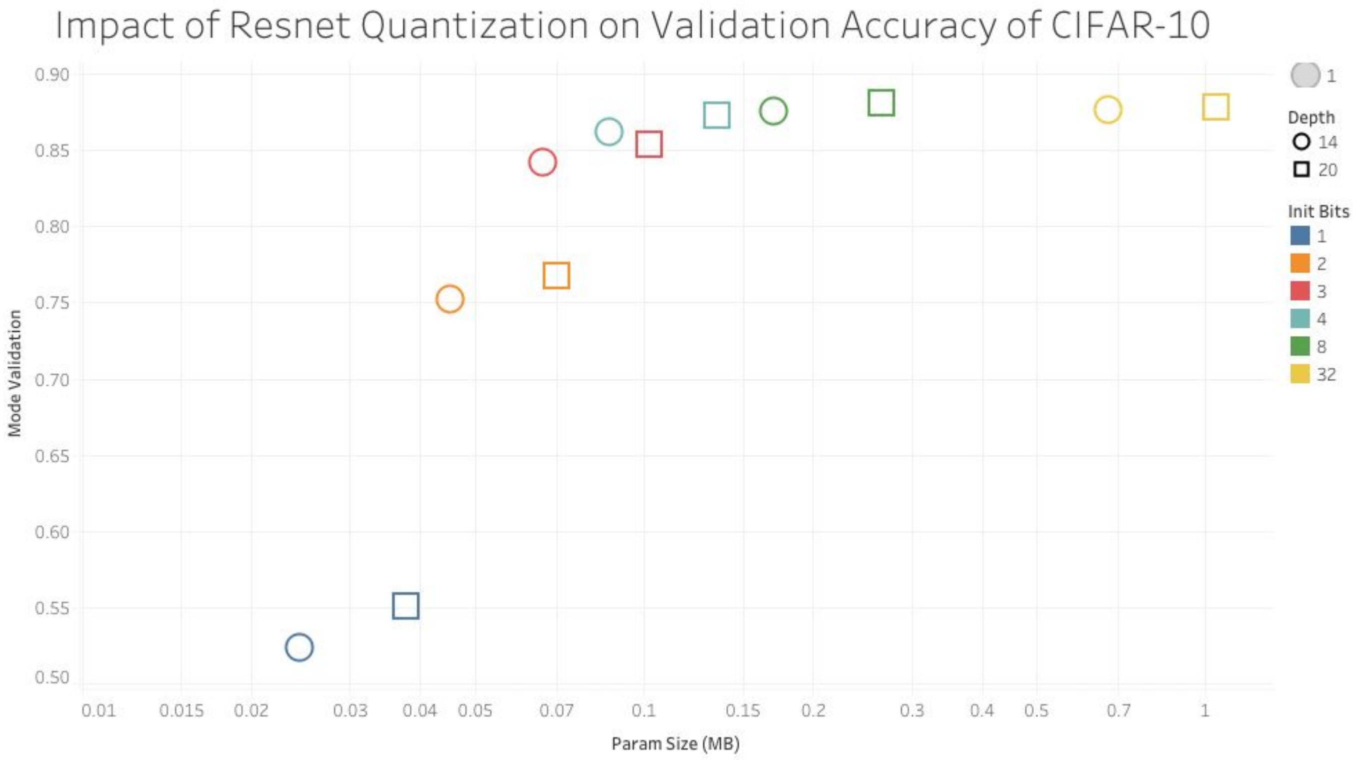
Accuracy & Latency after Decomposition

Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

Quantization



Quantization



Param Size (MB) vs. Mode Validation. Color shows details about sum of Init Bits. Size shows sum of Width Factor. Shape shows details about sum of Depth. The data is filtered on Weight Mode, which keeps tanh. The view is filtered on sum of Width Factor, which ranges from 1 to 1.

Train smaller model

- Knowledge distillation: use a teacher model (large model) to train a student model (small model)

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**91.61%**
Teacher	~9M	90.18%
Mimic single	~54M	84.6%
Mimic single	~70M	84.9%
Mimic ensemble	~70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Table 1: Accuracy on CIFAR-10

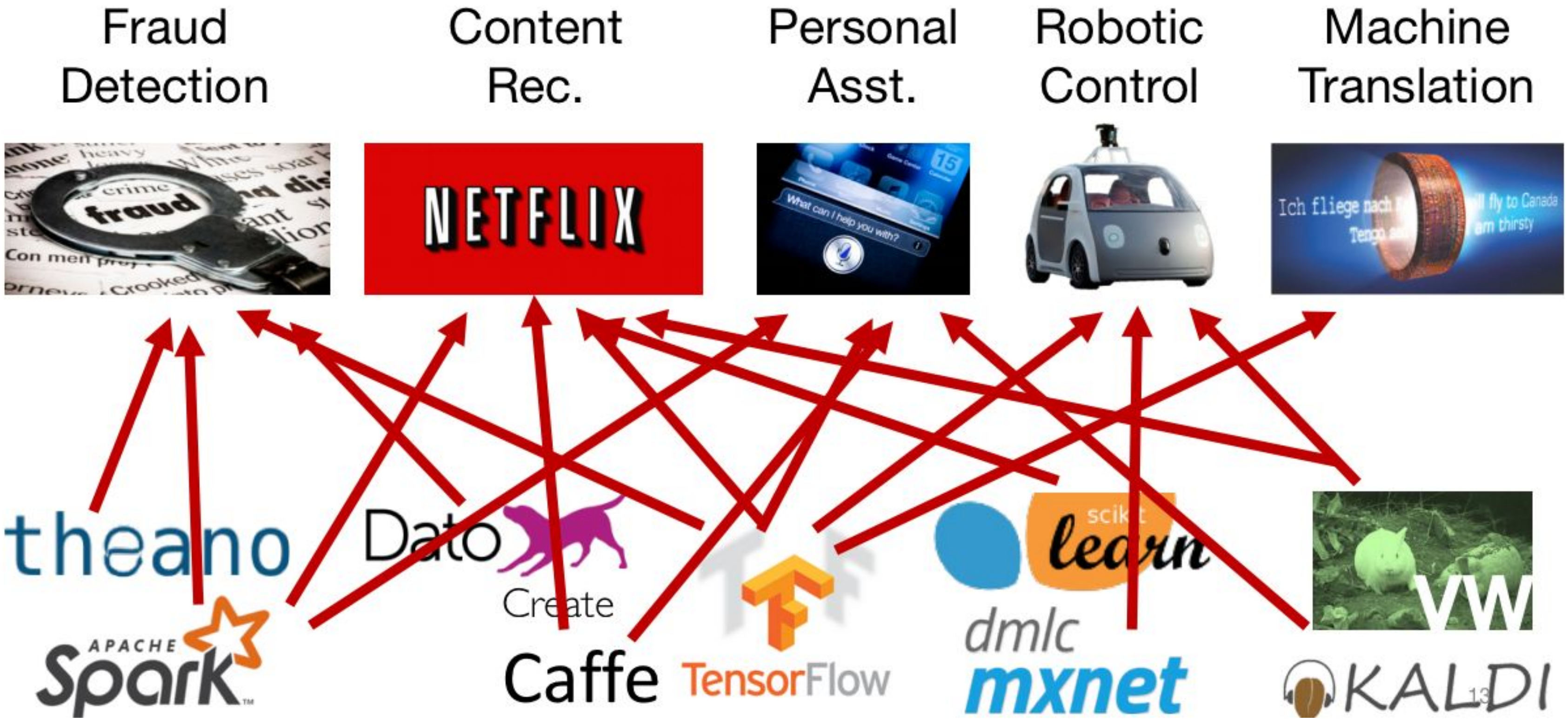
Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**64.96%**
Teacher	~9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Table 2: Accuracy on CIFAR-100

Serving backend

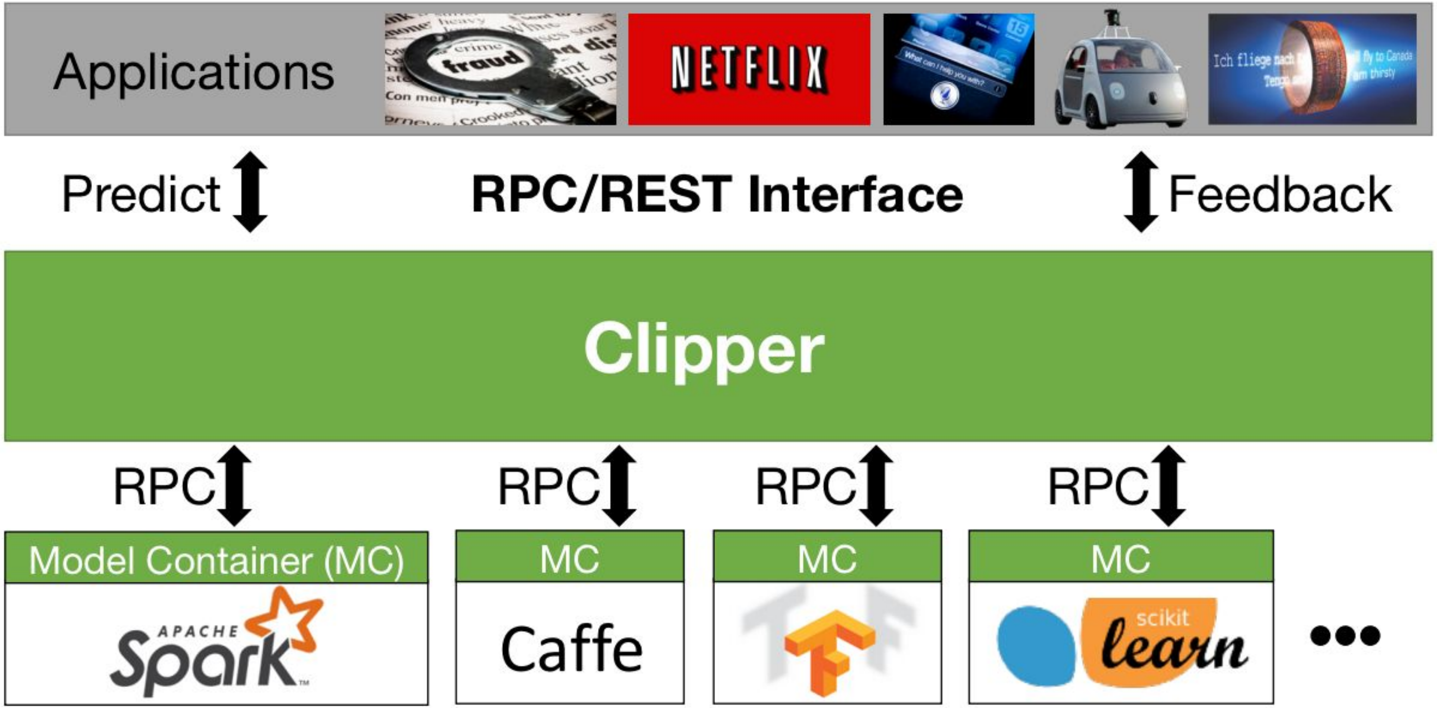
- Provide common abstraction for different frameworks
- Decide the batch size
- Load balancing and scheduling (on-going research)

Miscellaneous DL frameworks and models



* Crankshaw, Daniel, et al. "Clipper: A Low-Latency Online Prediction Serving System." presentation for NSDI (2017). https://www.usenix.org/sites/default/files/conference/protected-files/nsdi17_slides_crankshaw.pdf

Clipper Architecture



* Crankshaw, Daniel, et al. "Clipper: A Low-Latency Online Prediction Serving System." presentation for NSDI (2017). https://www.usenix.org/sites/default/files/conference/protected-files/nsdi17_slides_crankshaw.pdf

Model Abstraction

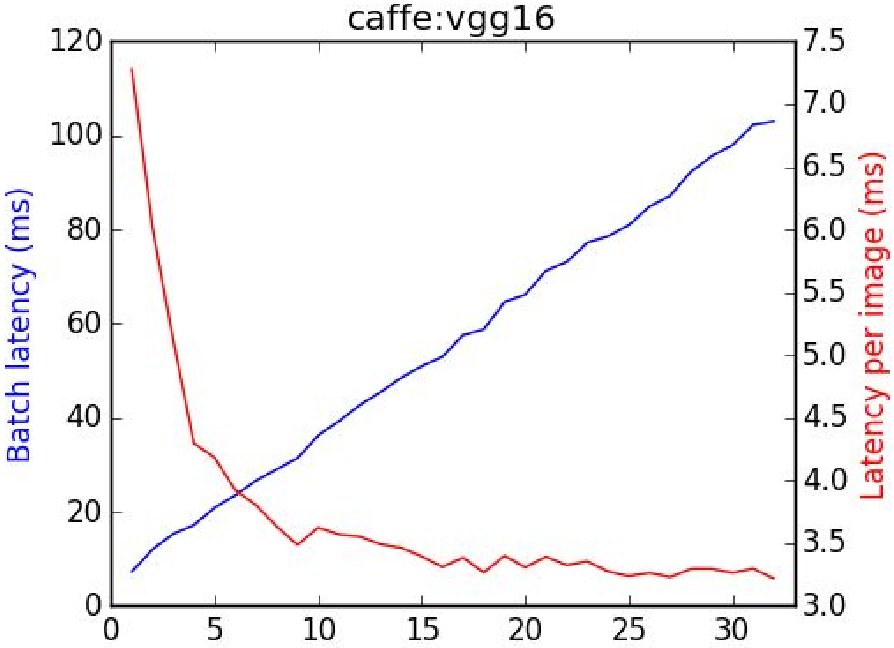
- Model container

```
class ModelContainer:
    def __init__(model_data)
    def predict_batch(inputs)
```

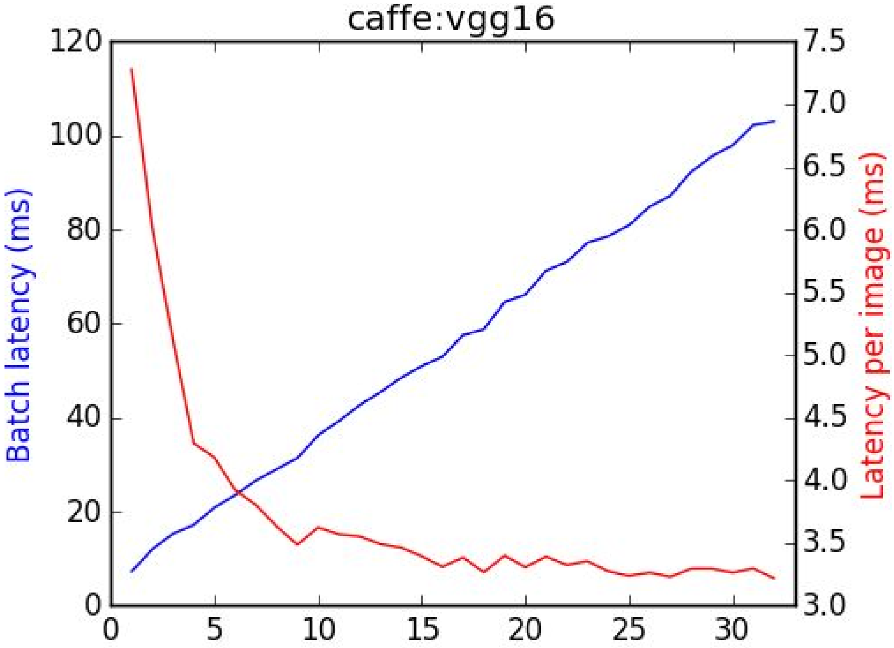
- Evaluate models using original framework
- Model run in separate process as Docker containers

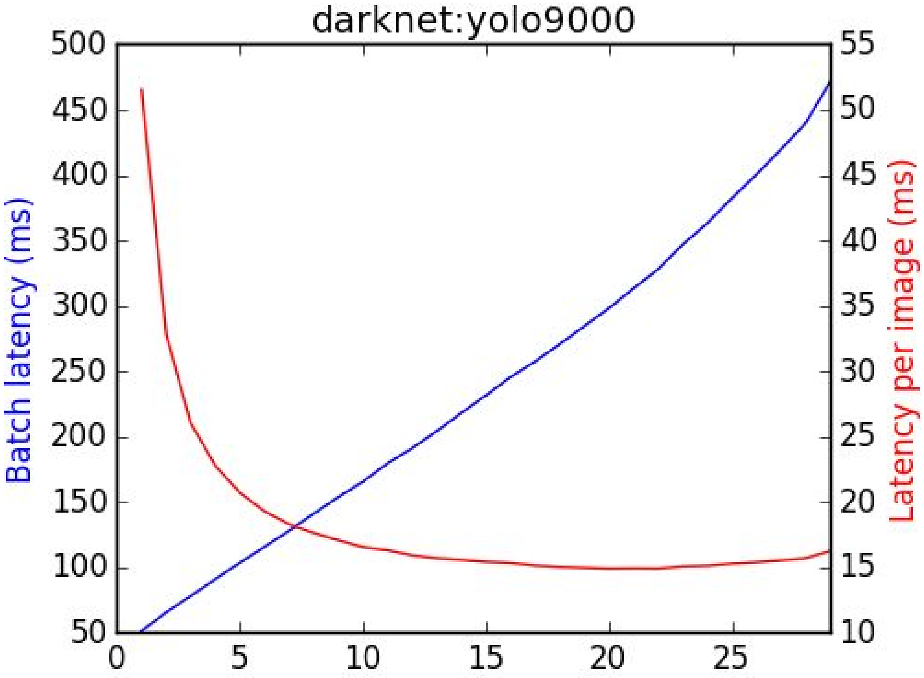
* Crankshaw, Daniel, et al. "Clipper: A Low-Latency Online Prediction Serving System." presentation for NSDI (2017). https://www.usenix.org/sites/default/files/conference/protected-files/nsdi17_slides_crankshaw.pdf

Batch / latency trade-off



Batch / latency trade-off





Runtime scheduling between device and cloud

- Schedule execution between device and cloud, and select approximate models
- Manage the battery and memory constraints for the device
- Manage the cost constraint for the cloud
- Goal: Maximize the overall accuracy

Resource usage for a continuous vision app

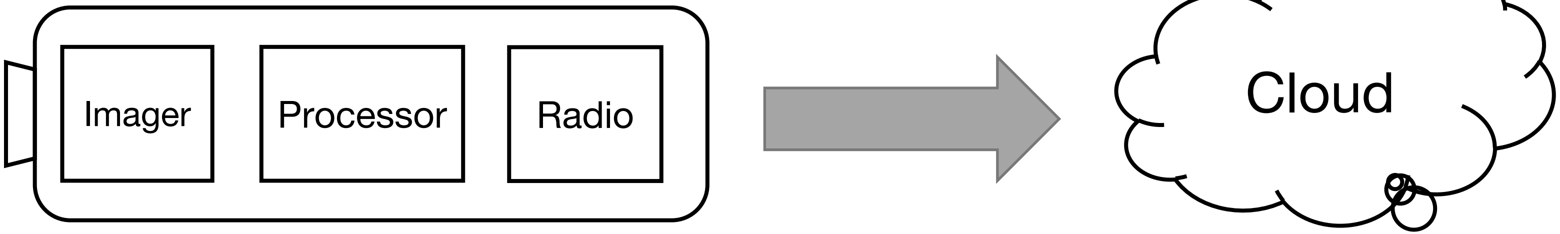
Omnivision
OV2740
90mW

Tegra K1 GPU
290GOPS@10W
= 34pJ/OP

Qualcomm SD810 LTE
>800mW
Atheros 802.11 a/g
15Mbps@700mW= 47nJ/b

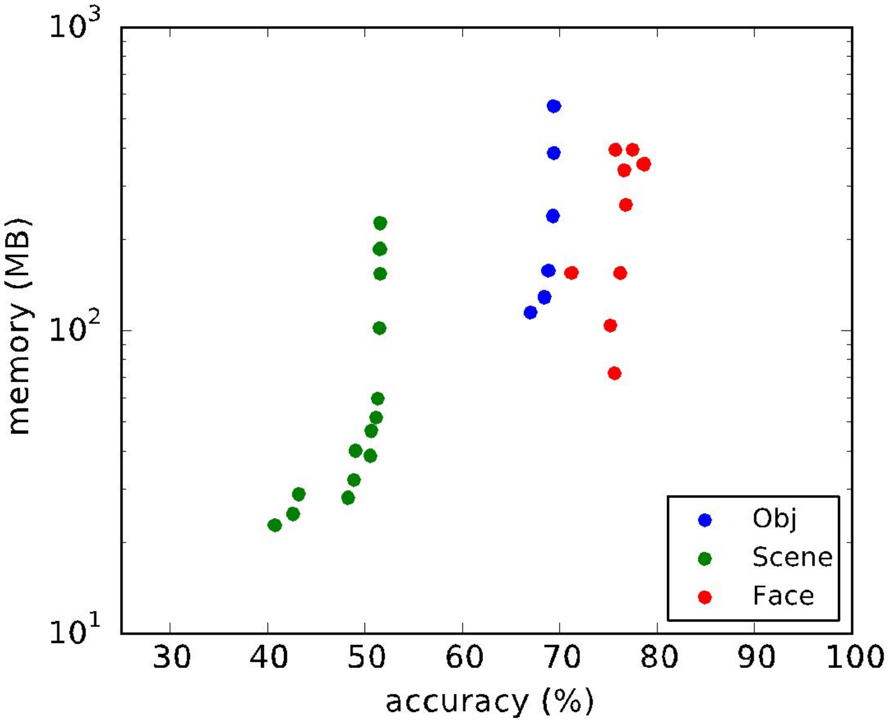
Amazon EC2

CPU	c4.large	2x400GFLOPS	$0.1/h
GPU	g2.2xlarge	2.3TFLOPS	$0.65/h



Workload	Deep learning **300GFLOPS** @ 30GFLOPs/frame, 10fps	
Budget	Device power 30% of 10Wh for 10h = 300mW	Cloud cost $10 person/year
Compute power	**9GFLOPS**	**3.5GFLOPS** (GPU) / **8GFLOPS** (CPU)

Memory / accuracy trade-off

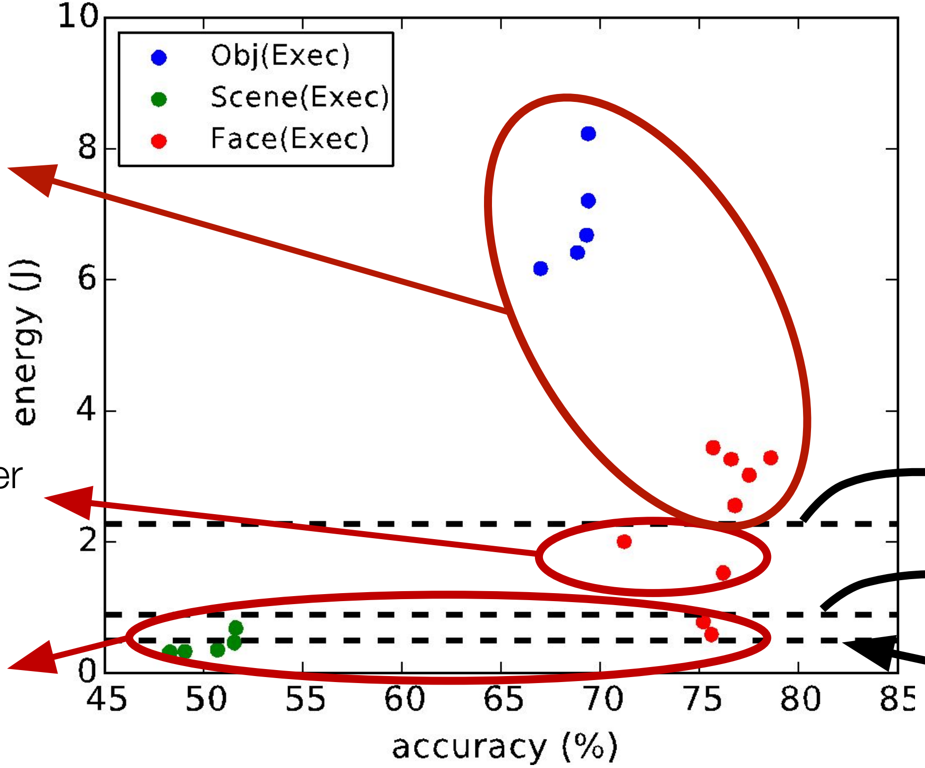


Energy / accuracy trade-off

Exceed energy budget when execute locally

Can execute locally under energy budget

Always execute locally



Compute energy budget (2.3J)

LTE xmit cost (0.9J)

Wifi xmit cost (0.5J)

Nvidia Jetson TK1

energy budget = total energy / total time(10h) / requests per second(1 req/sec)

Approximate model scheduling

memory | energy | cost

Model Pool

	Task 1		Task 2	
	model 1 (90%)	model 2 (80%)	model 3 (80%)	model 4 (70%)
memory	8	5	9	6
energy	2	1	3	2
cost	2	1	3	2

Mobile device

Energy 14

Memory 10

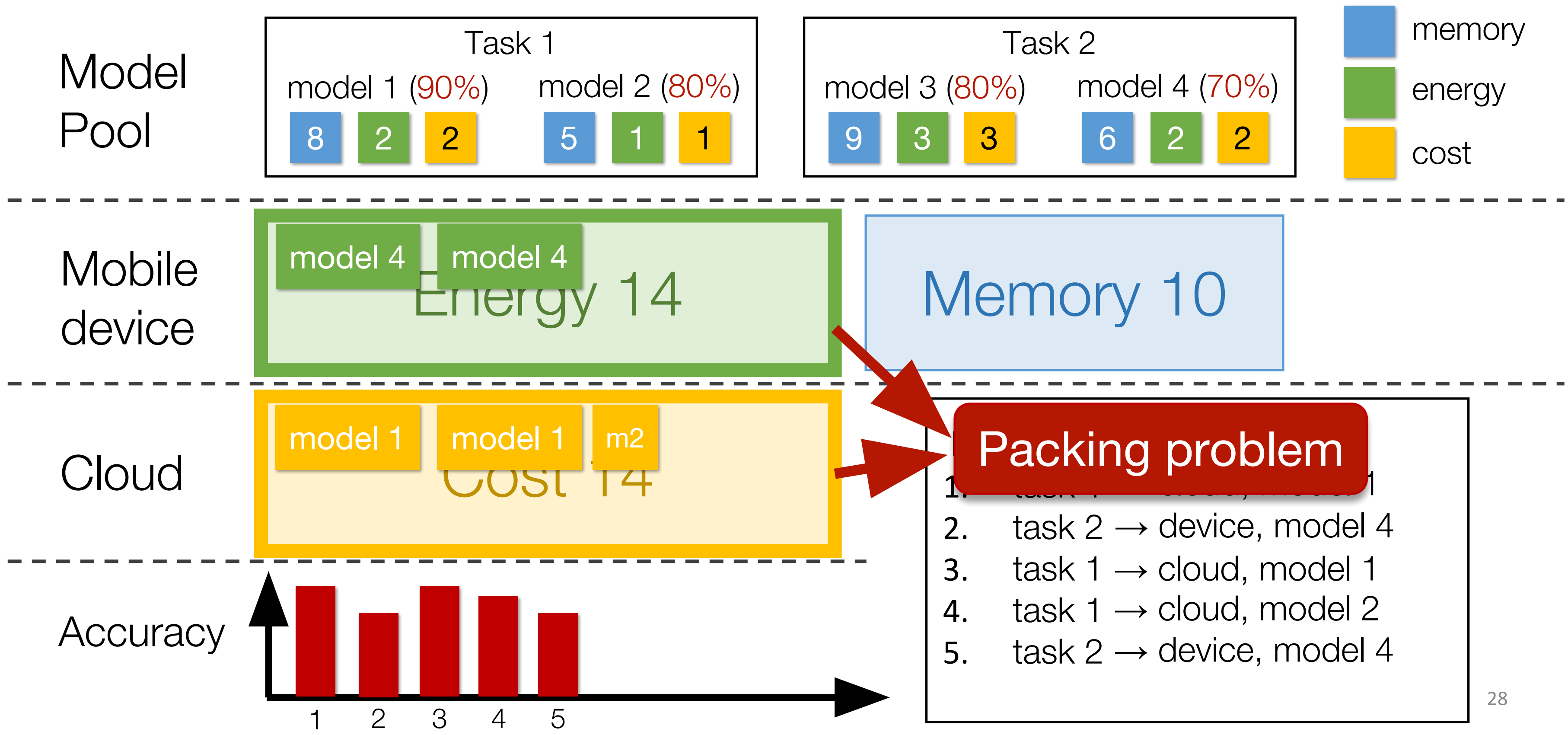
Cloud

Cost 14

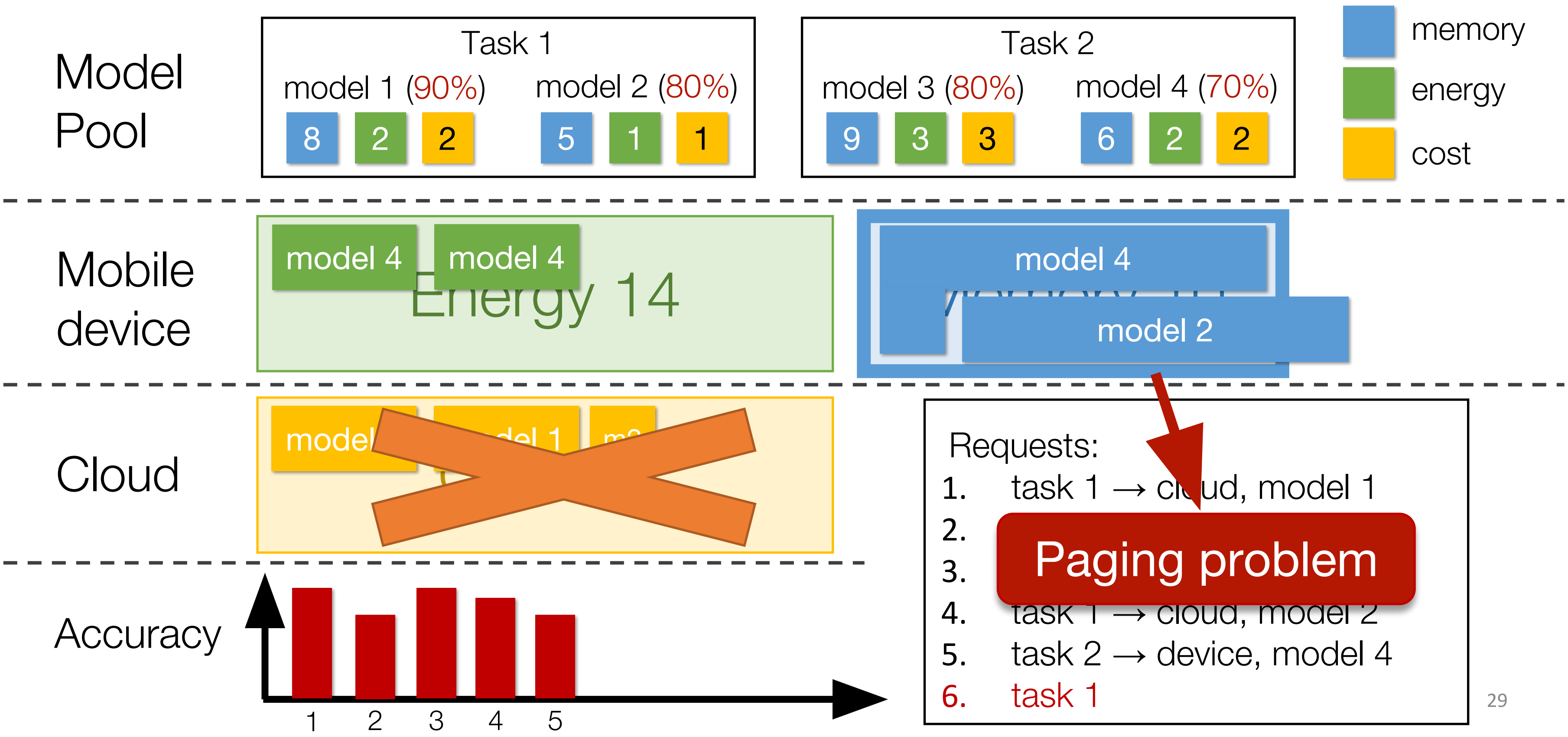
Accuracy

Goal: maximize the overall accuracy

Approximate model scheduling



Approximate model scheduling



Approximate model scheduling

- Packing problem: pick versions that satisfy energy/cost budgets

$$\sum_t e_i x_{it} \le E, \sum_t c_i x'_{it} \le C \quad (x_{it}, x'_{it} \in [0,1], x_{it} \cdot x'_{it} = 0)$$

- Paging problem: pick versions that fit in memory

$$\forall 1 \le t \le T, \sum_{i=1}^{n} s_i x_{it} \le S$$

- Goal: maximize the accuracy

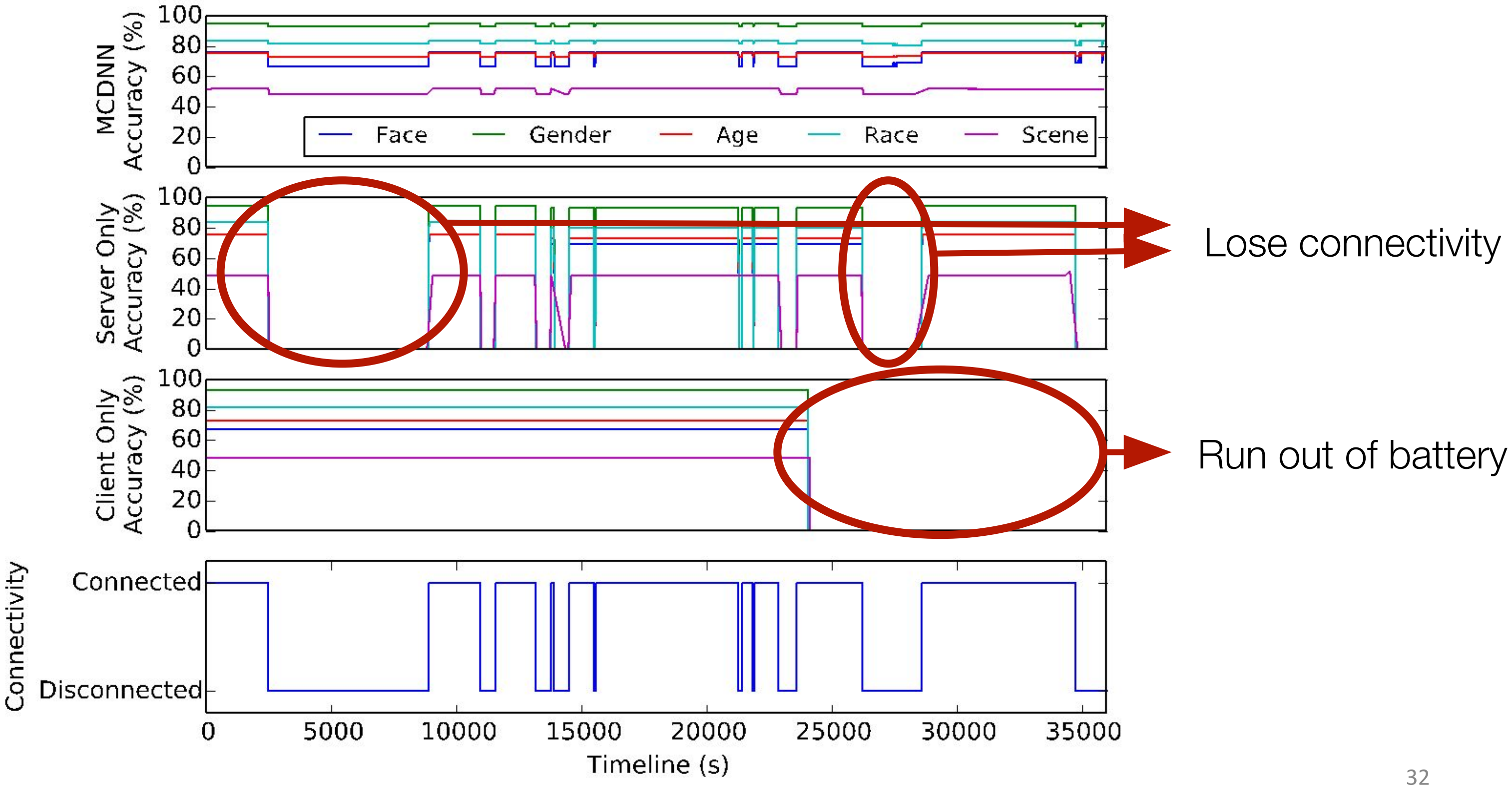
$$\max_x \sum_t \sum_i a_i (x_{it} + x'_{it})$$

No known optimal online algorithms

Heuristic scheduler

- Estimate future resource use and compute the budget for each request
- Account for paging cost to reduce oscillations
- Use increasingly more accurate versions of more heavily used models

Trace-driven evaluation



Reference

- Kim, Yong-Deok, et al. "Compression of deep convolutional neural networks for fast and low power mobile applications." ICLR (2016).
- Han, Seungyeop, et al. "MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints." MobiSys (2016).
- Romero, Adriana, et al. "Fitnets: Hints for thin deep nets." ICLR (2015).
- Crankshaw, Daniel, et al. "Clipper: A Low-Latency Online Prediction Serving System." NSDI (2017).