Neurosurgeon

Collaborative Intelligence Between the Cloud and Mobile Edge

by Y. Kang, J.Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars and L. Tang

Stefanos Laskaridis
sl829@cam.ac.uk

UNIVERSITY OF CAMBRIDGE

R244: Large-Scale Data Processing and Optimisation

Summary

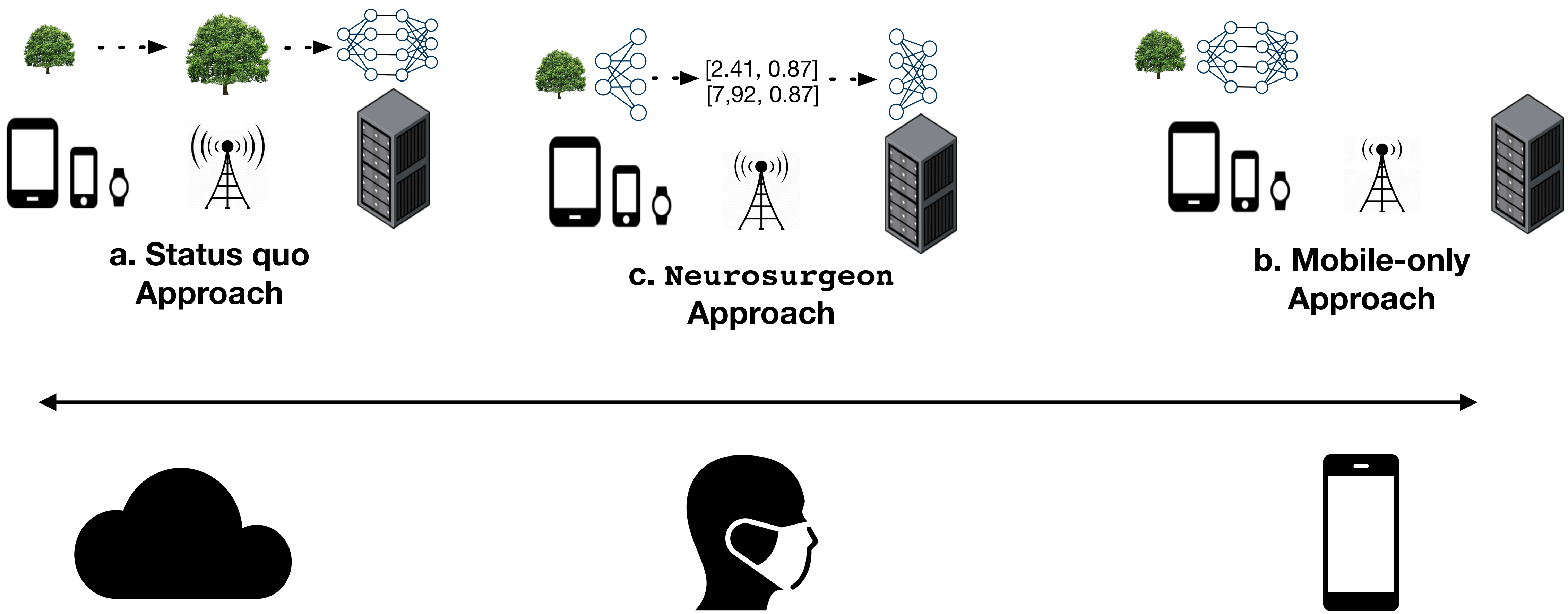


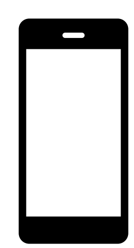
Image taken from [1]

Status Quo

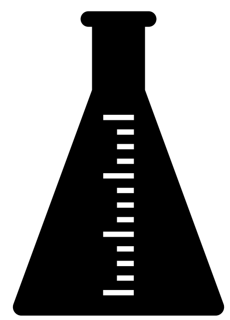
Status Quo

- Deep Neural Networks in “intelligent” applications
  - Apple Siri, Google Now, Microsoft Cortana
- Deep Neural Network applications are **mostly** offloaded to powerful private or public clouds for computation
  - Computer Vision
  - Natural Language Processing
  - Speech Recognition
- Large volume of data transfers cause **latency** and **energy consumption.**
- **However**, SoC advancements urged authors to revisit the problem.

The Mobile edge



Experiment Setup



Power Consumption
Watts Up? meter

Software
- Deep Learning: Caffe
- mCPU: OpenBLAS
- GPU: cuDNN

Mobile Platform

- Tegra K1 SoC
- 4+1 quad core ARM Cortex A15 CPU
- 2GB DDR3L 933MHz
- NVIDIA Kepler with 192 CUDA cores

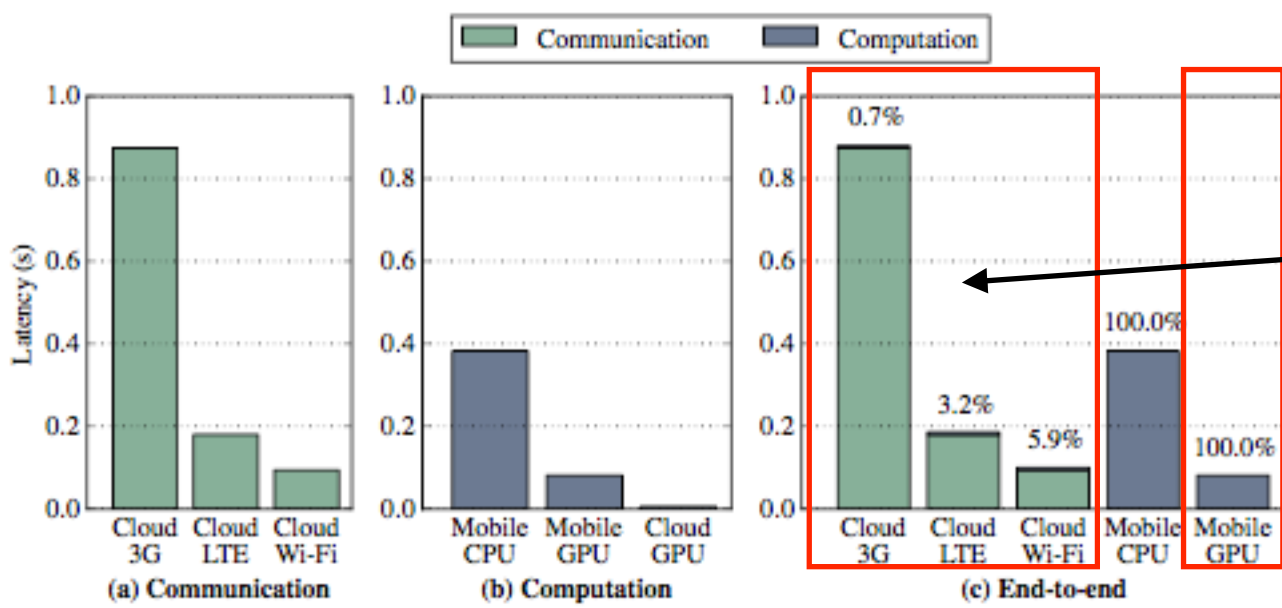
Server Platform

- 4U Intel Dual CPU Chassis, 8xPCIe 3.0 x 16 slots
- 2x Intel Xeon E5-2620, 2.1 GHz
- 1TB HDD
- 16x16GB DDR3 1866MHz ECC
- NVIDIA Tesla K40 M-class 12GB PCIe

Testing the Mobile Edge

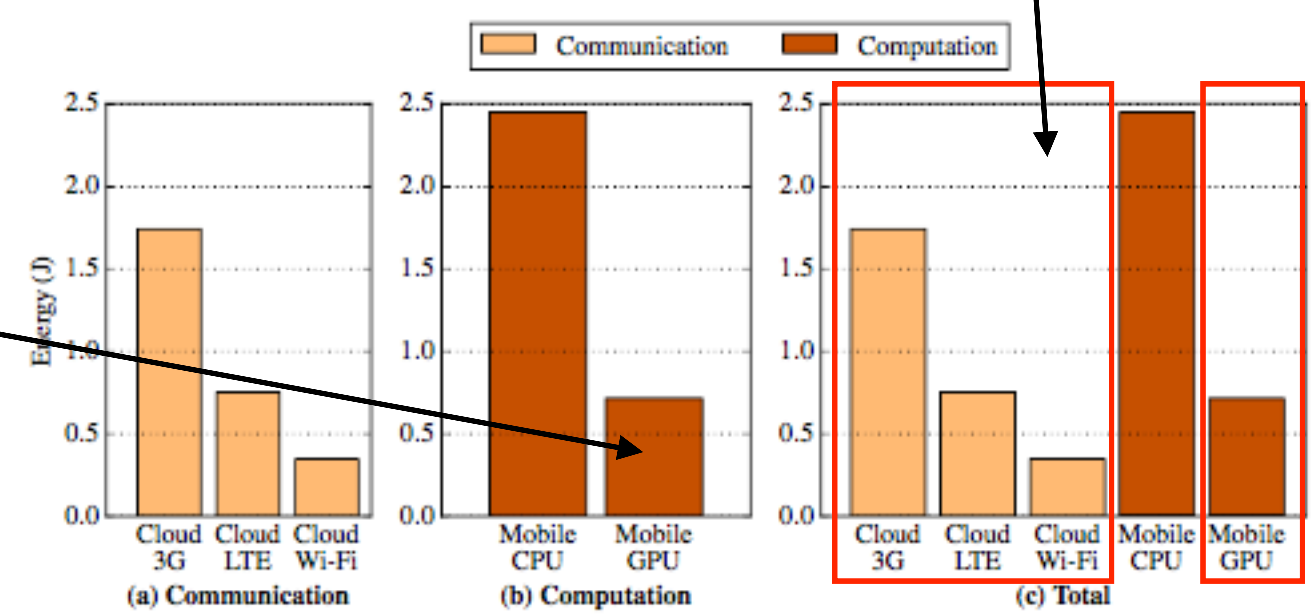
- Experiment running an Image of 152KB image through AlexNet [3]
- Measuring:
  - **Communication Latency**: 3G, LTE, WiFi
  - **Computation Latency**: mCPU, mGPU, cloud GPU
  - **End-to-end Latency**
  - **Energy Consumption**

Testing the Mobile Edge



Transmission has the dominating cost

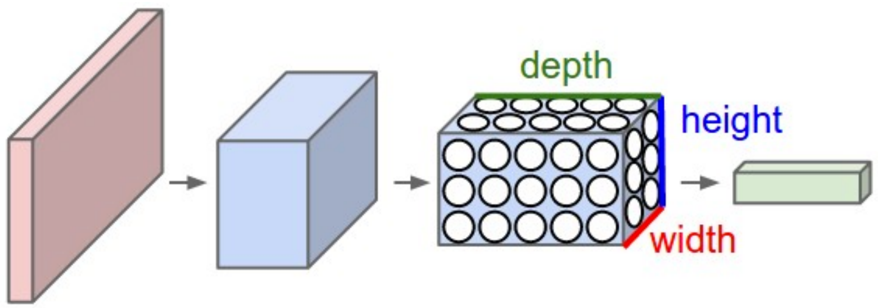
More power but shorter bursts

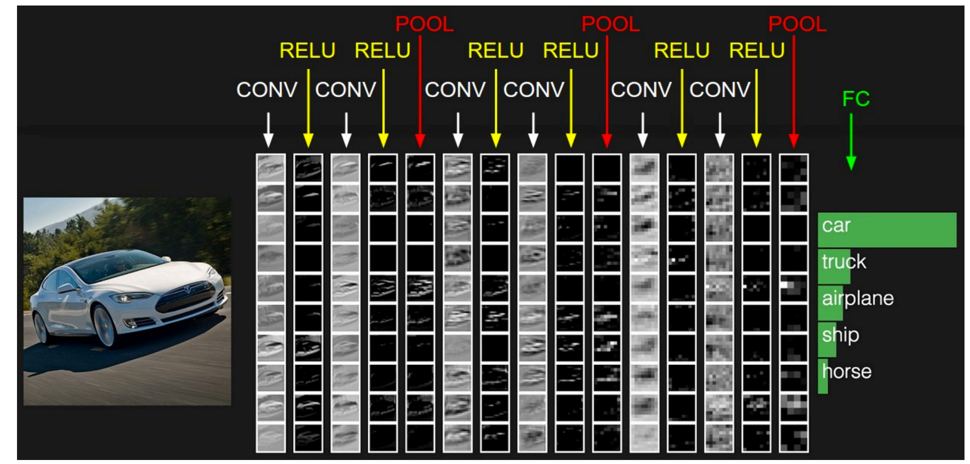


Images taken from [1]

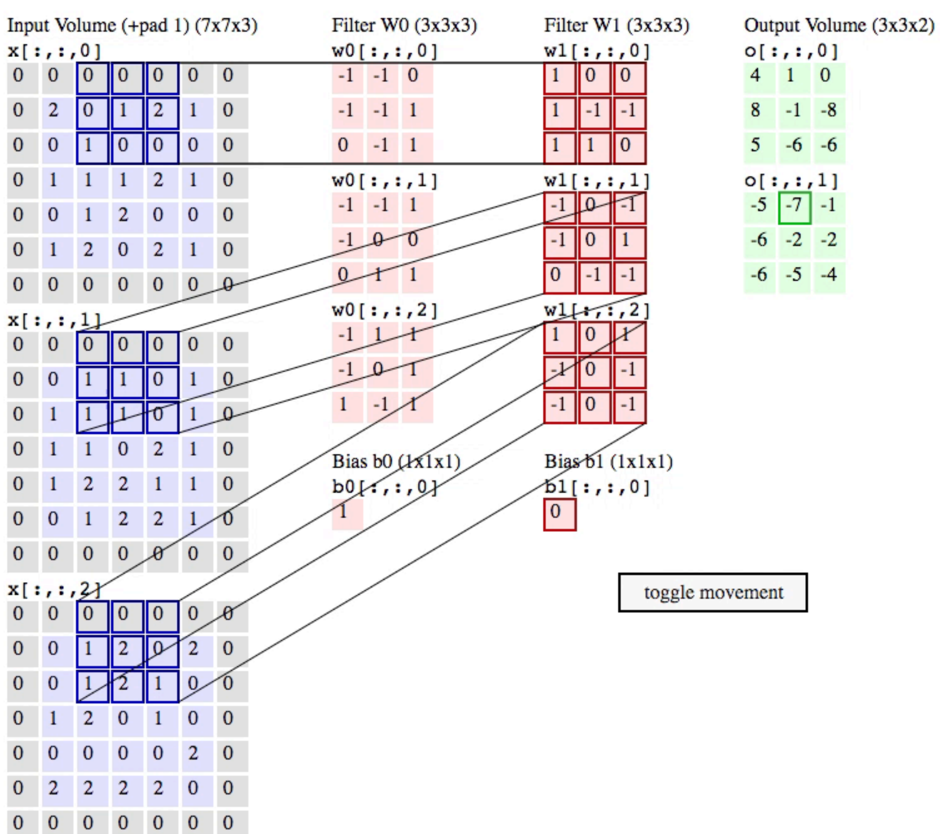
Neurosurgeon: Partitioning between Cloud and Mobile

cNN

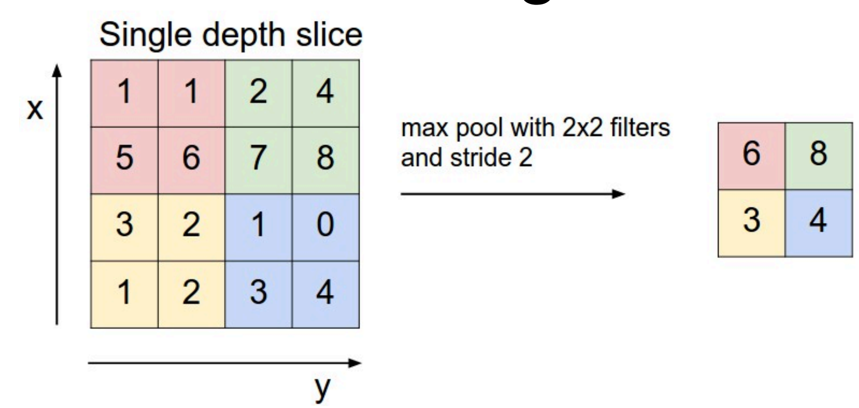


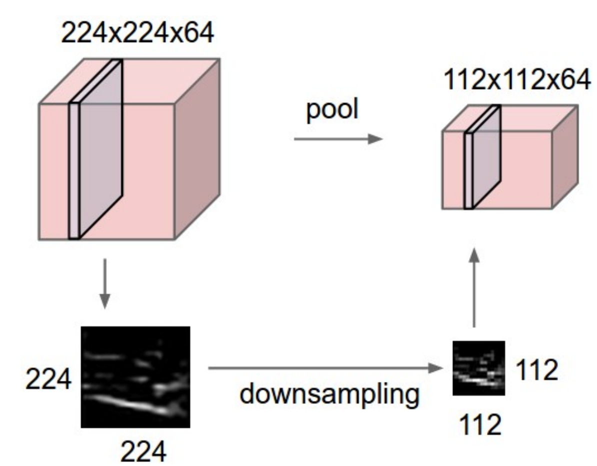


Convolution



Pooling





Images taken from [2]

DNN Layer types

- **Fully Connected Layer** (`fc`)
  All neurons are connected with all the neurons of the previous layer. Depth is the number of filters. Stride is how much we slide the filter each time. [2]

- **Convolutional & Local Layer** (`conv, local`)
  Convolves an image with one or more filters to produce a set of maps.

- **Pooling Layer** (`pool`)
  Downsamples an image to simplify representation. Can be average, max, or L2. [2]

- **Activation Layer** (`sig, relu, htanh`)
  Applies non-linear function to its input (sigmoid, Rectified Linear Unit, Tanh)

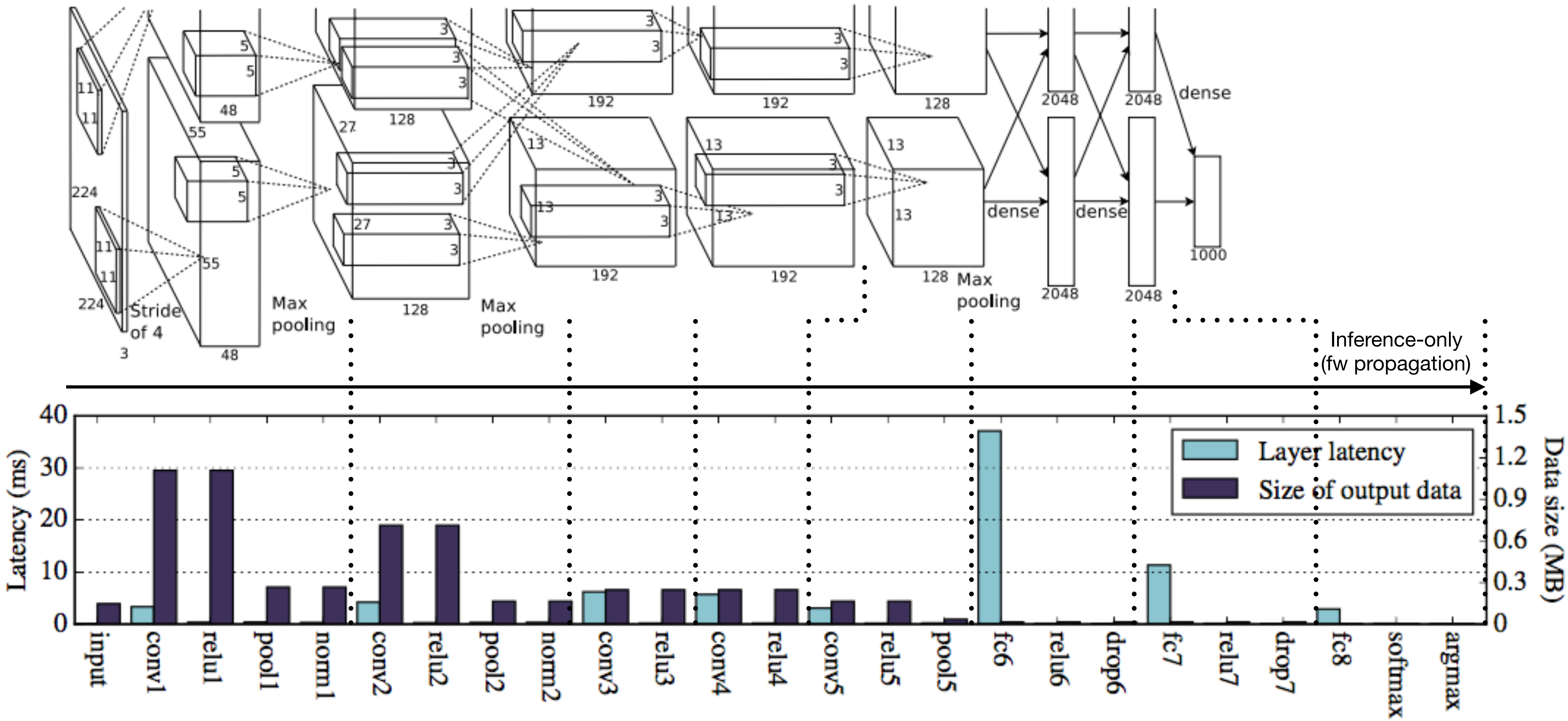
- **Normalisation layer** (`norm`)
  Normalises features across feature map.

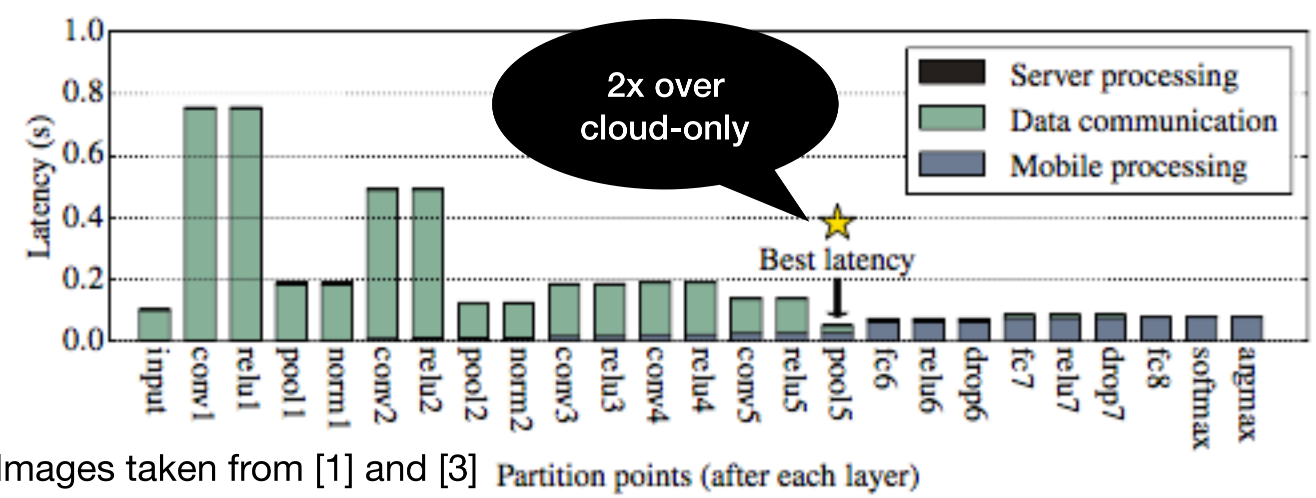
- **Softmax Layer** (`softmax`)
  Probability distribution over possible classes.

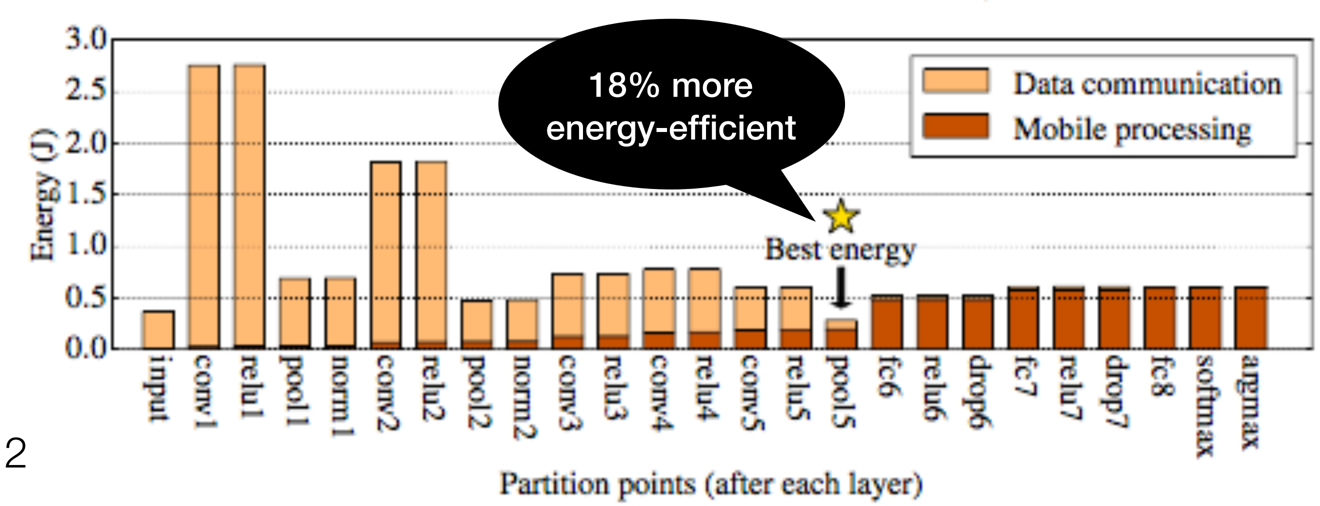
- **Argmax Layer** (`argmax`)
  Chooses class with higher probability.

- **Dropout Layer** (`dropout`)
  Randomly ignores neurons as regularisation to prevent overfitting.

AlexNet

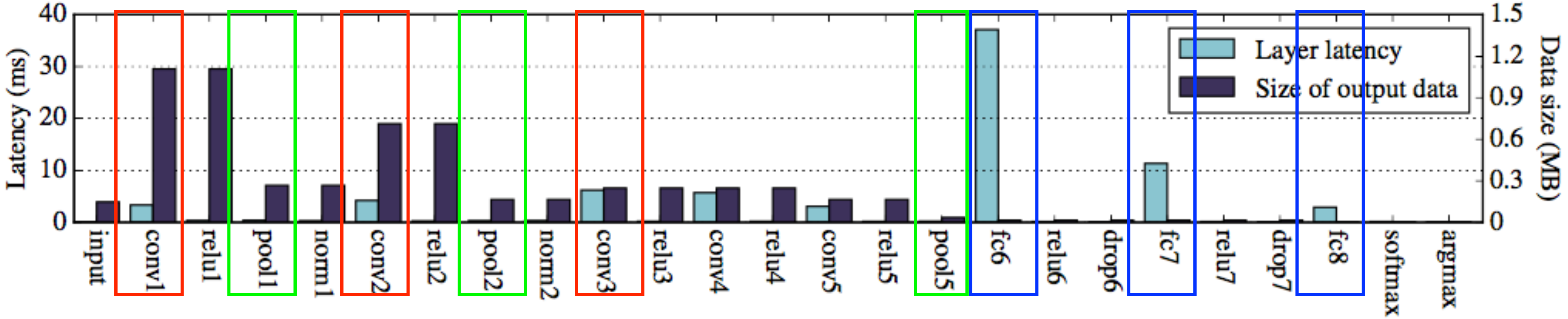






Images taken from [1] and [3]

AlexNet



- **Convolutional layers** produce a lot of data.
- **Pooling layers** reduce a lot of data.
- **Fully connected layers** operate on few data but have high latency.

Images taken from [3]

Partitioning

- First layers have most of the data (convolutions and pooling)

- Later layers have most of the latency (fully connected layers)

- **Key idea**: Compute locally until the point it make sense and then offload to cloud.

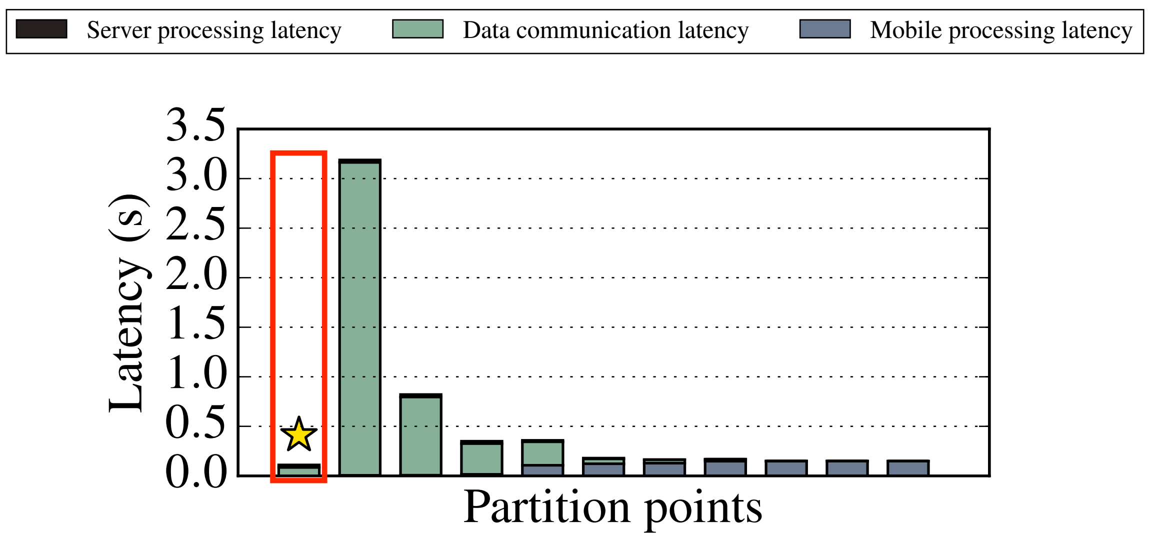
More Applications

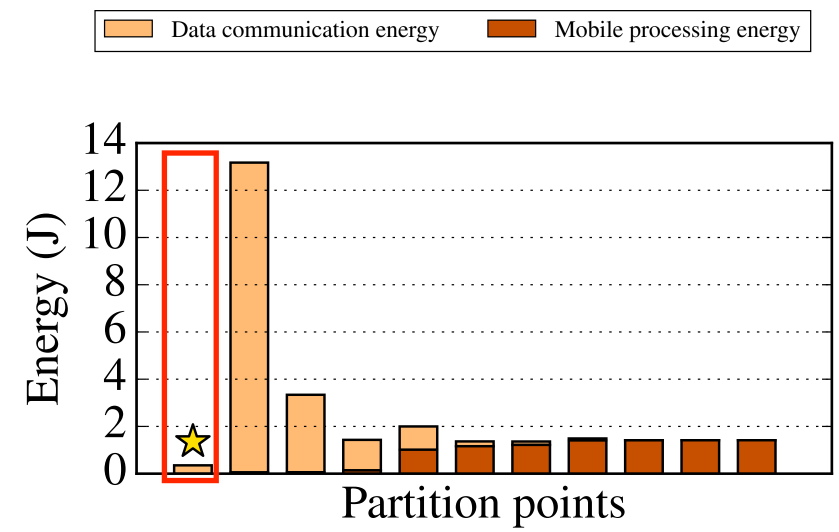
	Abbreviation	Network	Input	Layers
Image Classification	IMC	AlexNet	Image	24
	VGG	VGG	Image	46
Facial Recognition	FACE	DeepFace	Image	10
Digit Recognition	DIG	MNIST	Image	9
Speech Recognition	ASR	Kaldi	Speech	13
Part-of-speech Tagging	POS	SENNA	Word vectors	3
Named Entity Recognition	NER	SENNA	Word vectors	3
Word Chunking	CHK	SENNA	Word vectors	3

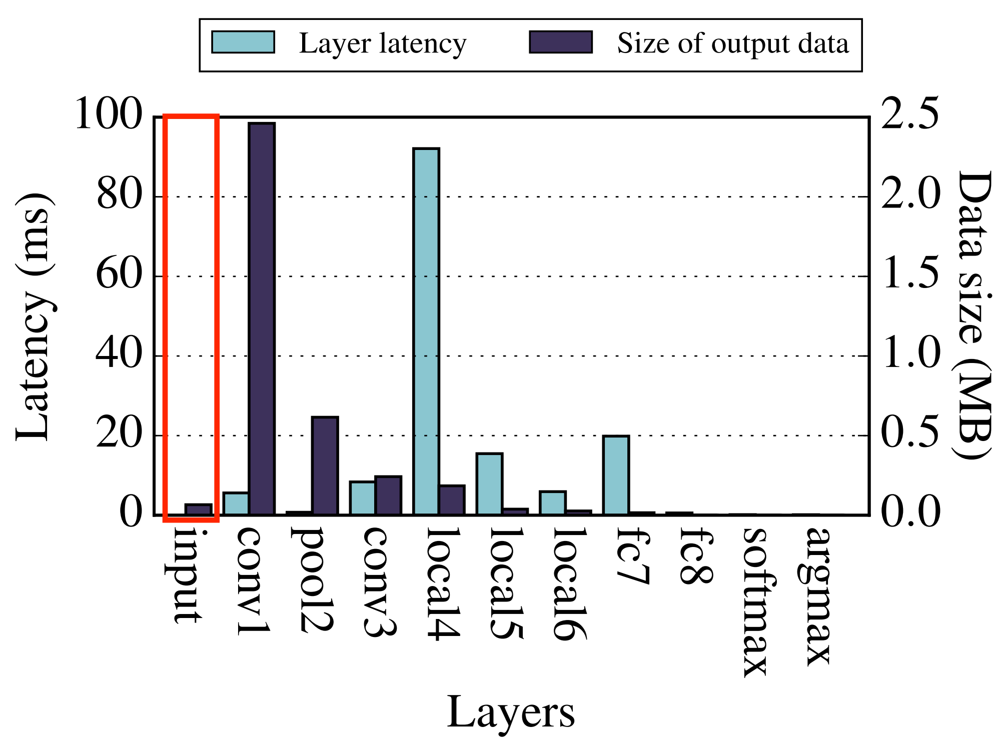
VGG

Images taken from [1]

FACE

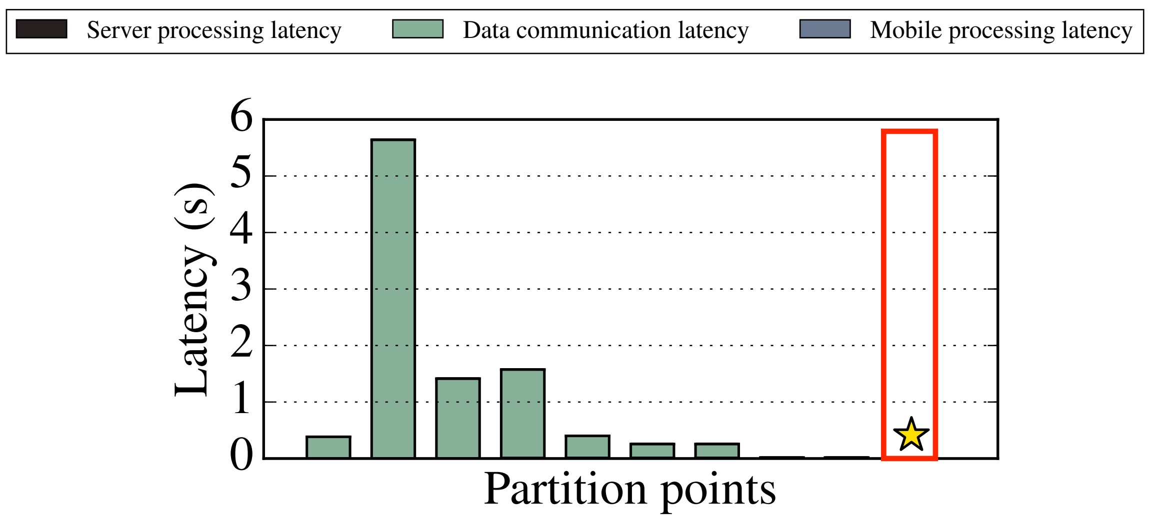


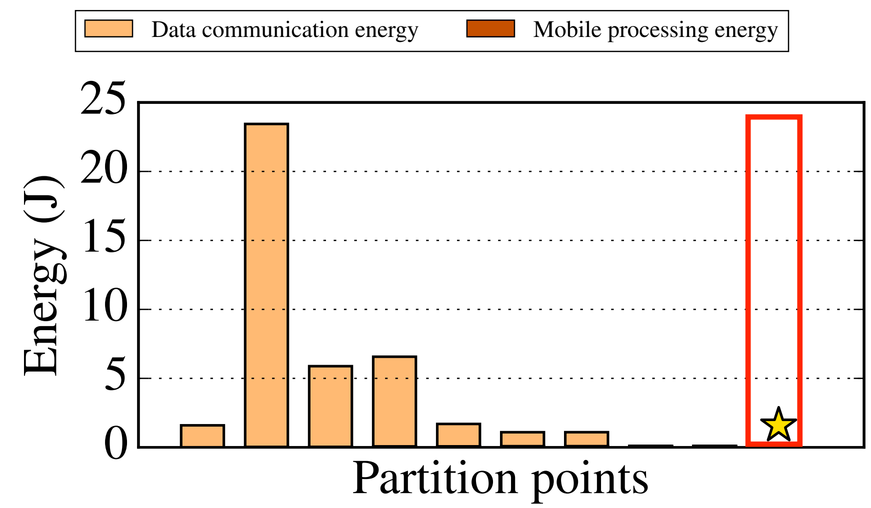


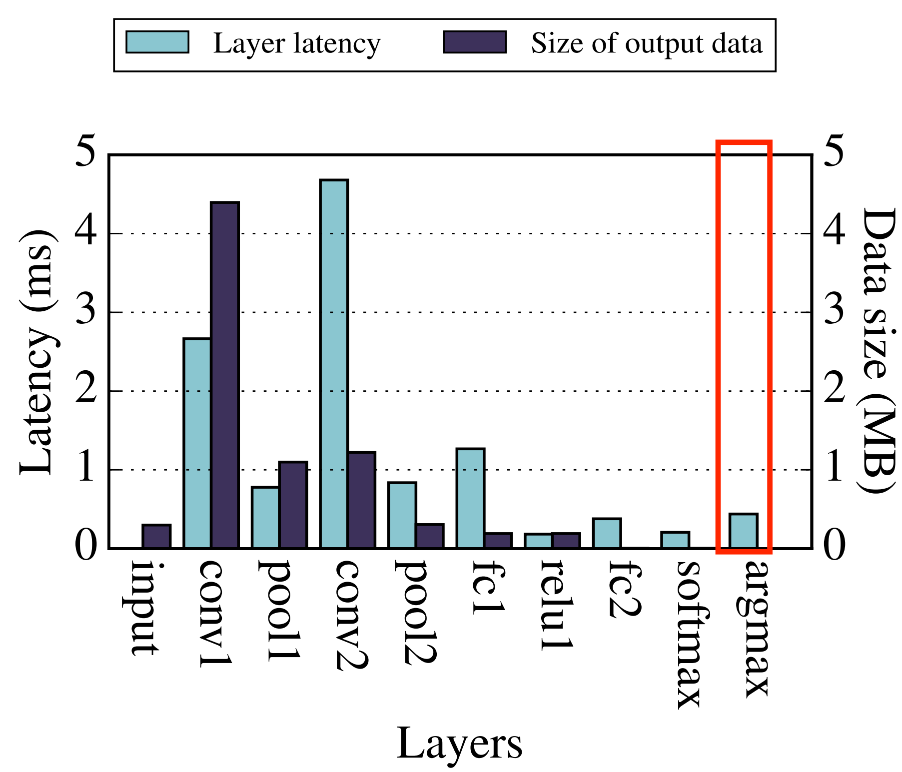


Images taken from [1]

DIG

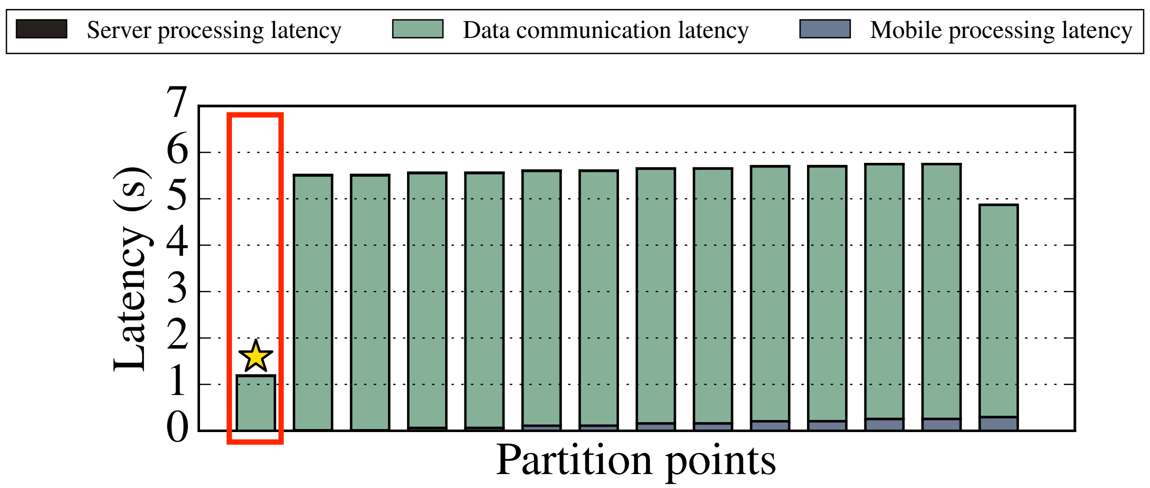


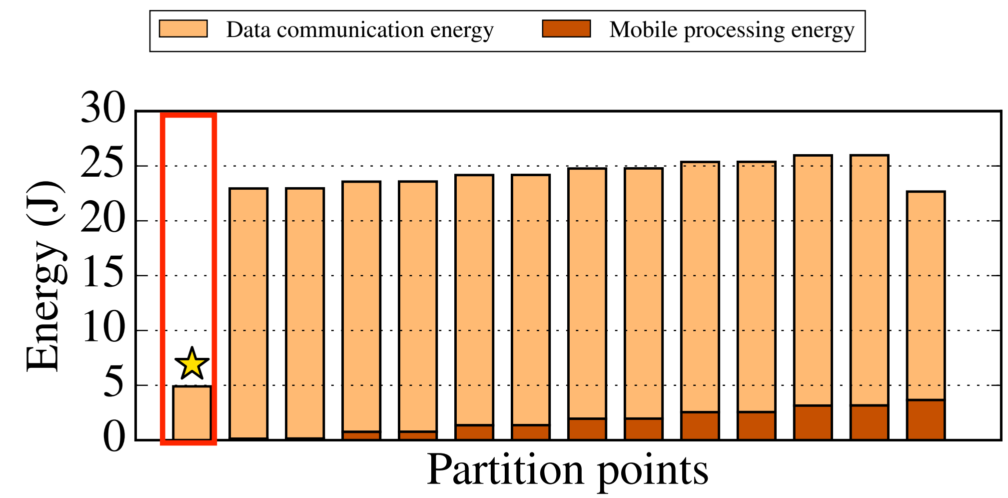


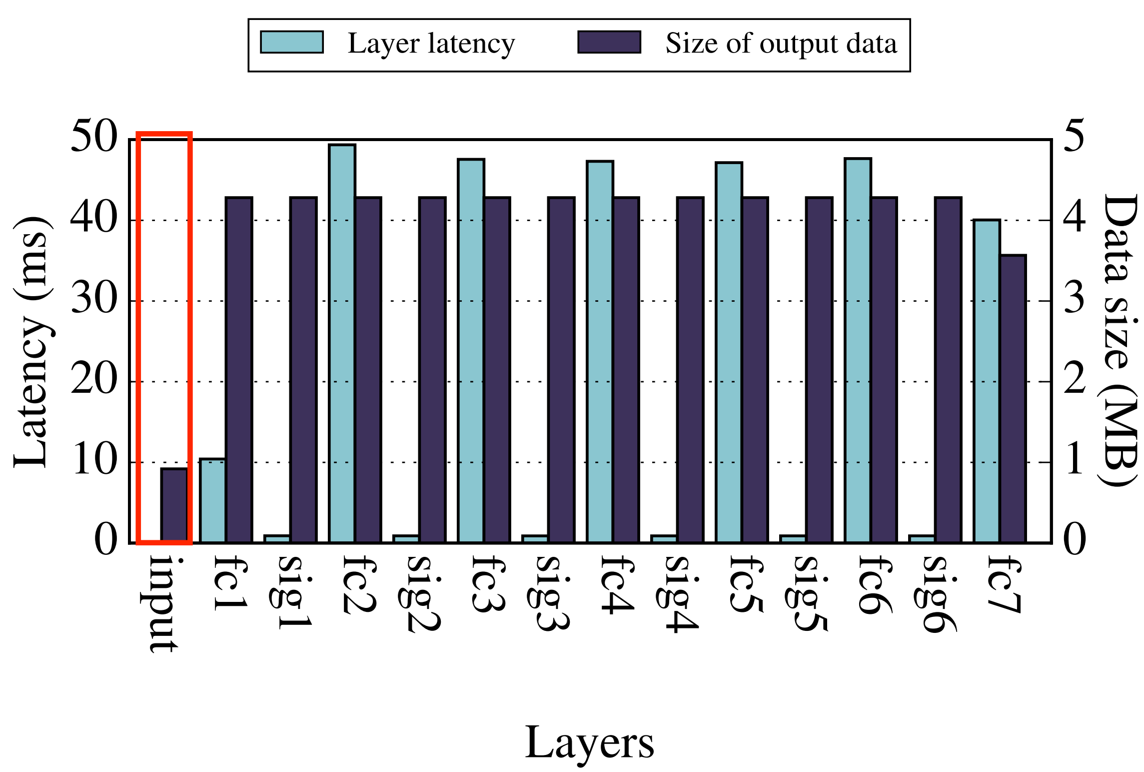


Images taken from [1]

ASR

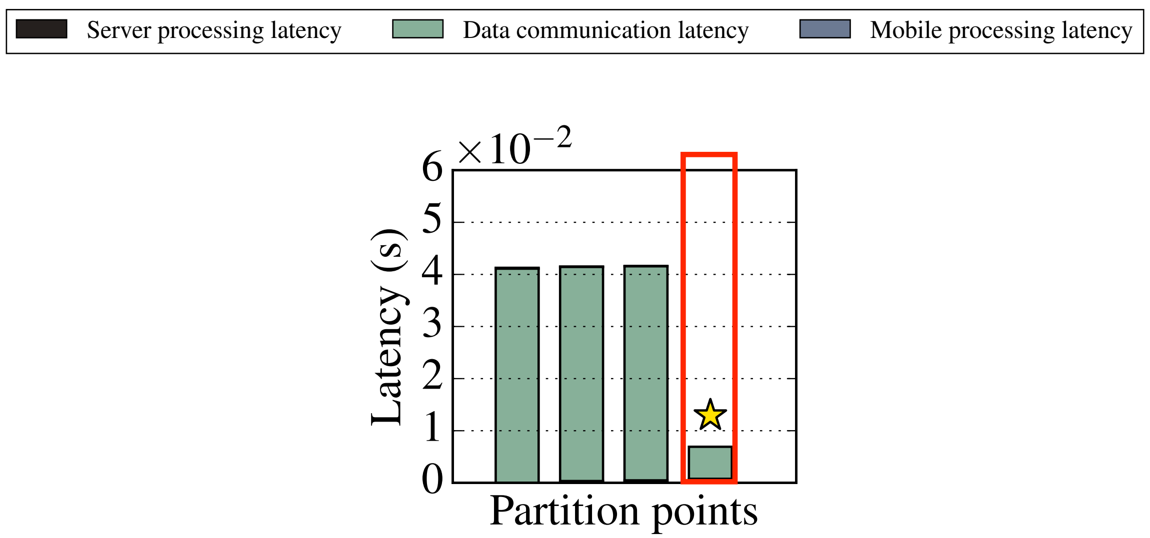


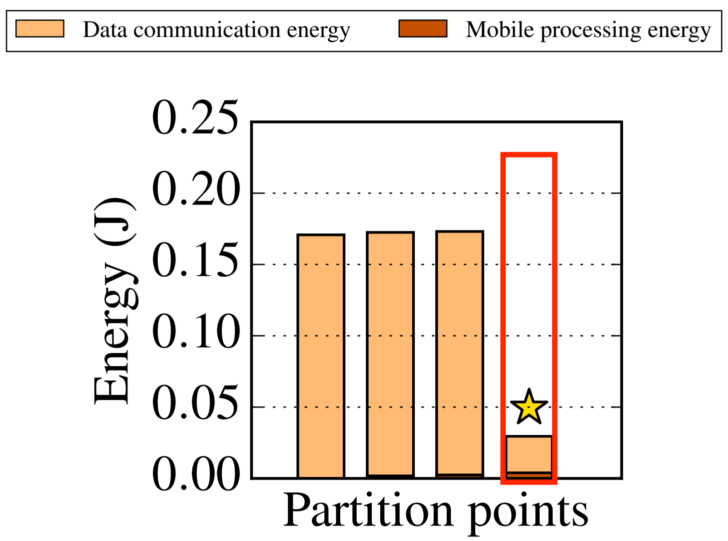


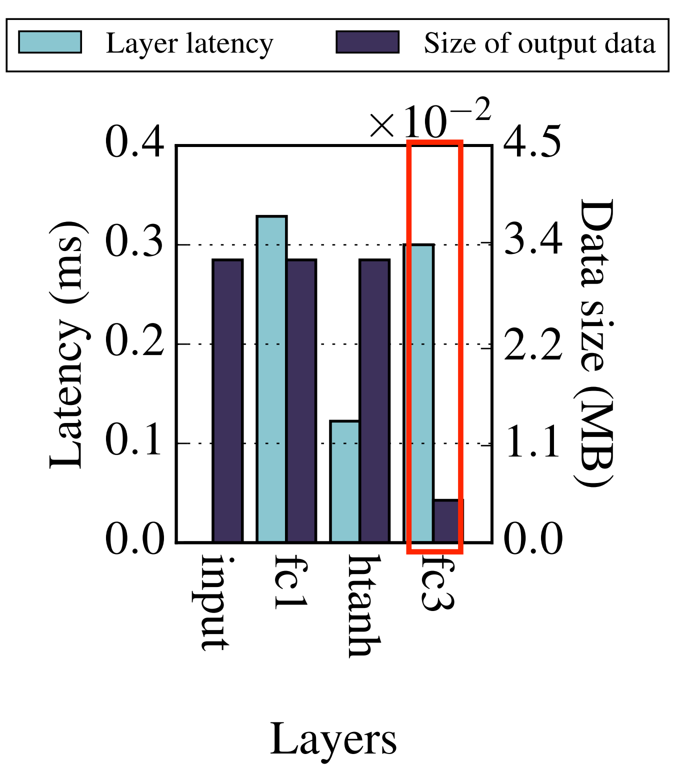


Images taken from [1]

POS

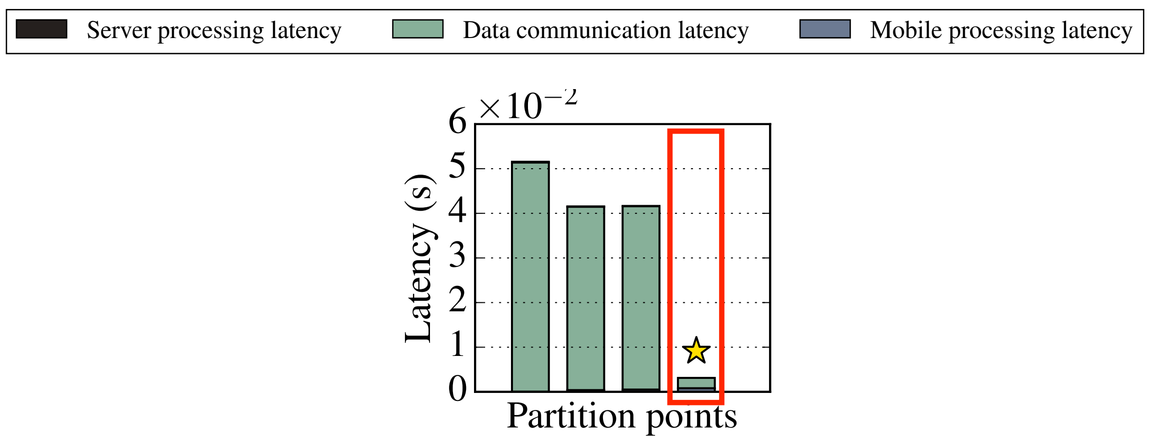


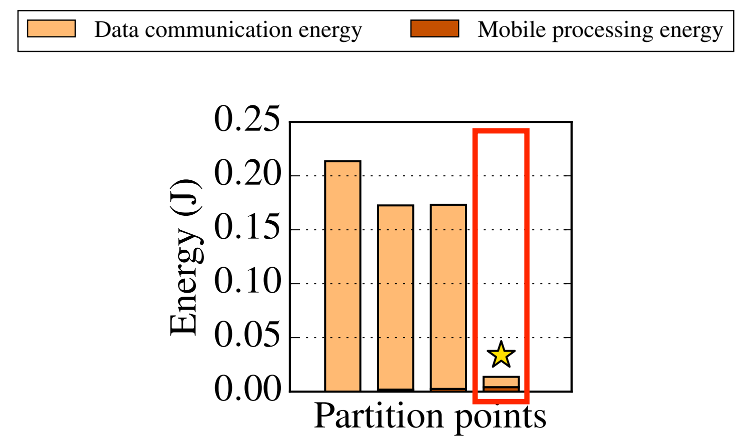


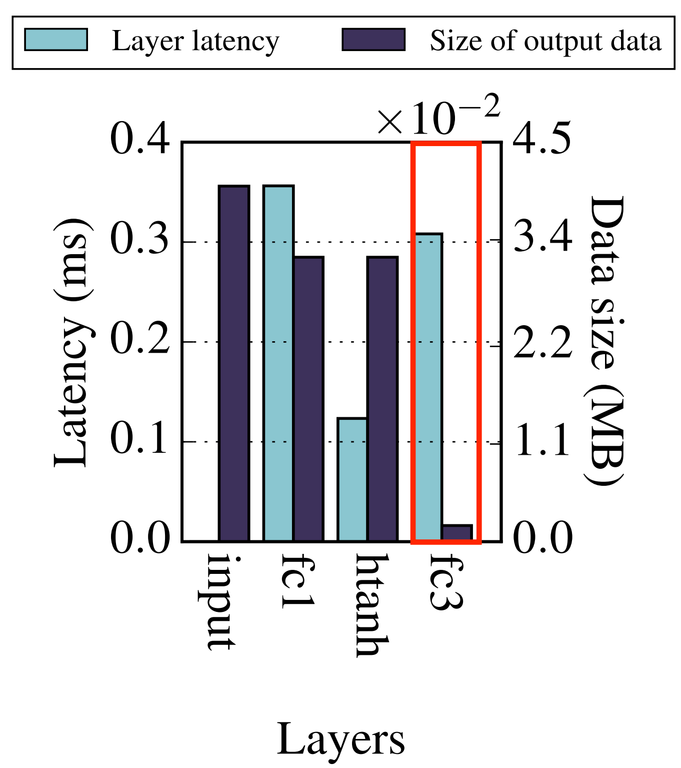


Images taken from [1]

NER

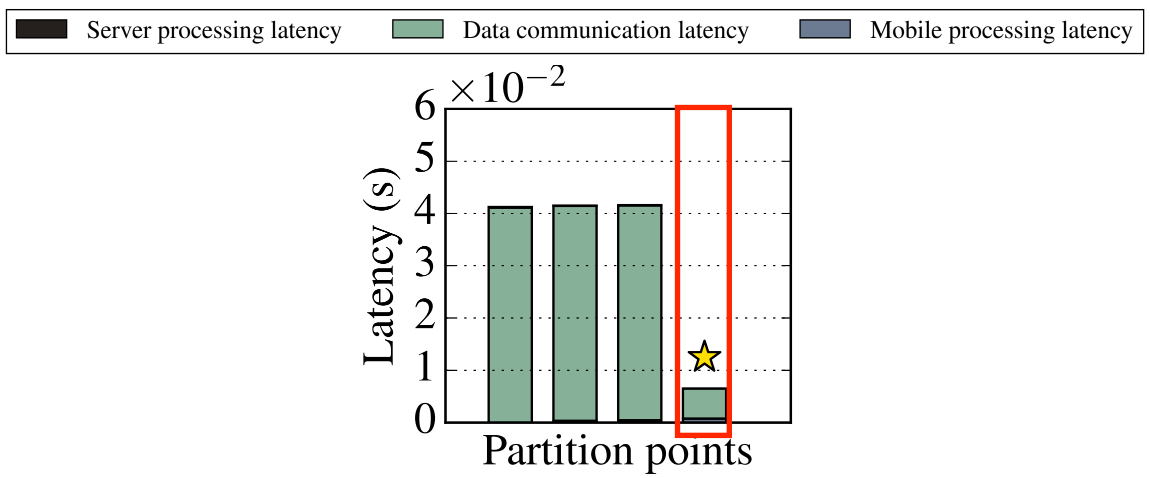


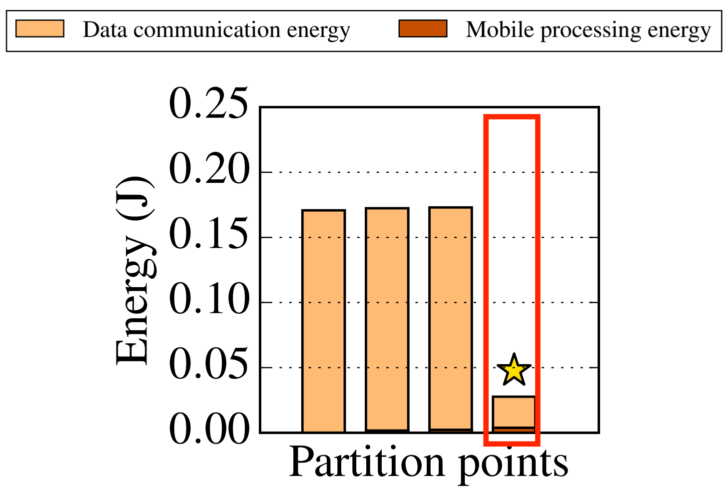


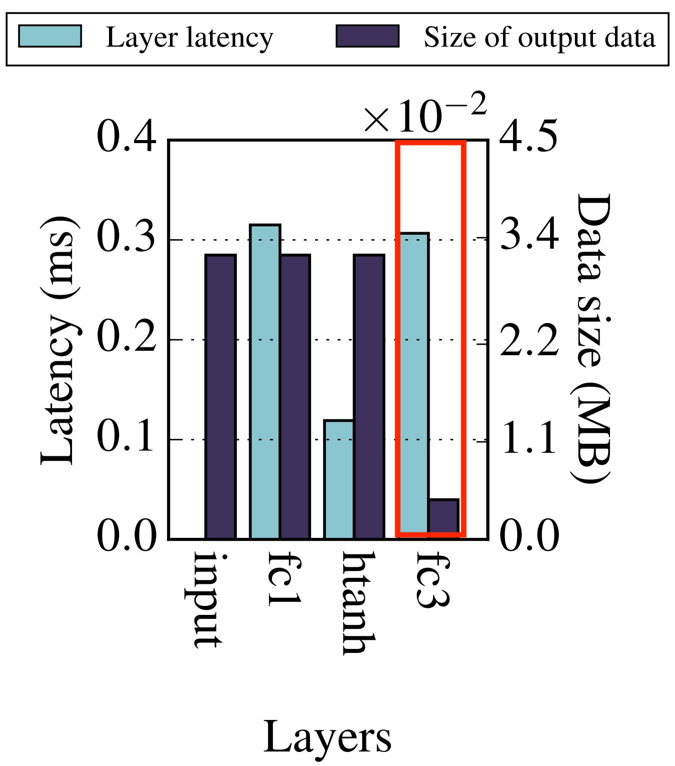


Images taken from [1]

CHK







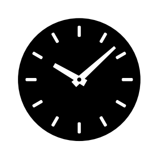
Images taken from [1]

Neurosurgeon

Neurosurgeon

- Partitions DNN based on:
  - **DNN Topology**
    - Computation latency
    - Data size output
  - **Dynamic factors**
    - Wireless network
    - Datacenter workload

Neurosurgeon

- **Profiles** device and cloud server
  - To generate prediction models
  - One time, in advance
  - Results stored in device for decision-making
- Two, distinct goals:
  - Latency minimisation 
  - Energy optimisation 

Neurosurgeon

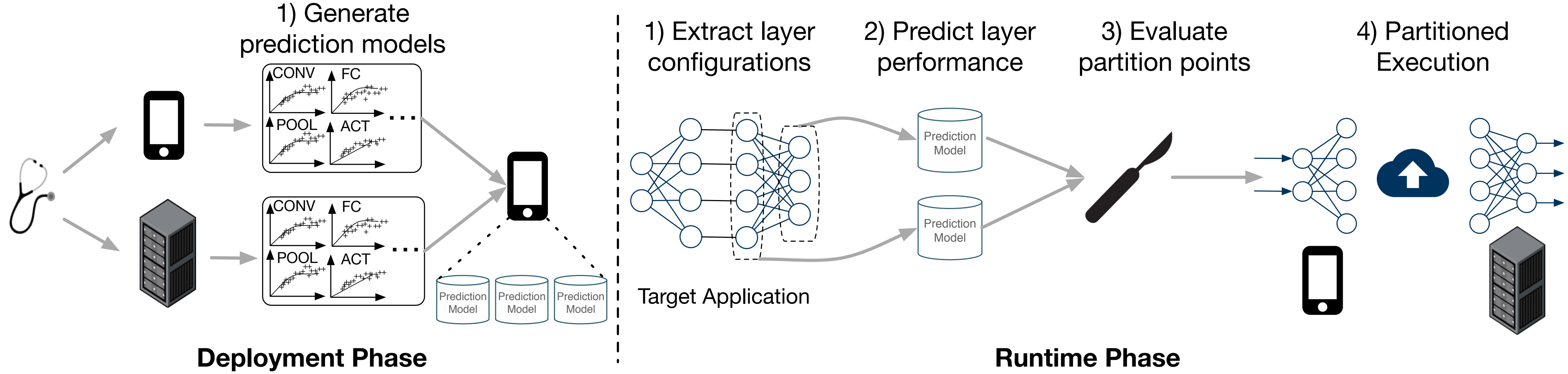


Image taken from [1]

Regression model per DNN Layer

Linear or logarithmic regression model. GFLOPS for performance.

Layer	Regression Variables
Convolution	(filter_size/stride)^2 * (# filters)
Local, Pooling	input, output feature maps
Fully Connected	# Input/Output neurons
Softmax, Argmax	# Input/Output neurons
Activation, Normalisation	# neurons

Partitioning Algorithm

Input:
N: number of layers in the DNN
$\{L_i | i = 1 \cdots N\}$: layers in the DNN
$\{D_i | i = 1 \cdots N\}$: data size at each layer
$f, g(L_i)$: regression models predicting the latency and power of executing L_i
K: current datacenter load level
B: current wireless network uplink bandwidth
PU: wireless network uplink power consumption
procedure PARTITIONDECISION
  for each i in $1 \cdots N$ **do**
    $TM_i \leftarrow f_{mobile}(L_i)$ // Latency of mobile execution
    $TC_i \leftarrow f_{cloud}(L_i, K)$ // Latency of cloud execution
    $PM_i \leftarrow g_{mobile}(L_i)$ // Power of mobile execution
    $TU_i \leftarrow D_i / B$ // Transfer latency
  if $OptTarget == latency$ **then**
    return $\arg\min_{j=1 \cdots N} (\boxed{\sum_{i=1}^{j} TM_i +} \sum_{k=j+1}^{N} \boxed{TC_k + TU_j})$
  else if $OptTarget == energy$ **then**
    return $\arg\min_{j=1 \cdots N} (\sum_{i=1}^{j} \boxed{TM_i \times PM_i} + \boxed{TU_j \times PU})$

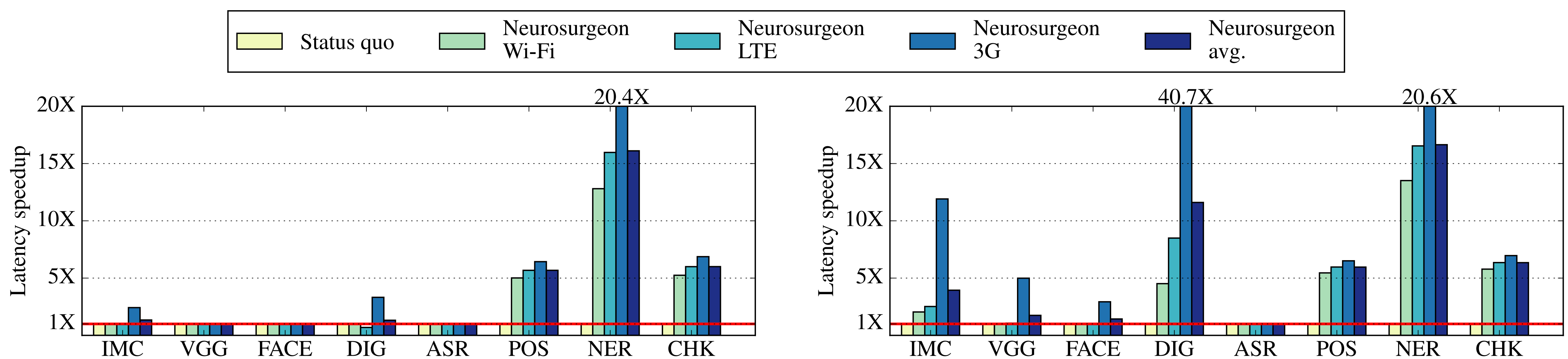
Device calculation
Cloud Calculation
Power consumed for local calculations
Uplink power consumption

Image taken from [1]

Benchmark Results - Latency Optimisation

Mispredicts when performance is close to one another.

Mobile	Wireless network	Benchmarks							
		IMC	VGG	FACE	DIG	ASR	POS	NER	CHK
CPU	Wi-Fi	input	input	input	input	input		fc3	
	LTE	input	input	input	argmax	input		fc3	
	3G	argmax	input	input	argmax	input		fc3	
GPU	Wi-Fi	pool5	input	input	argmax	input		fc3	
	LTE	argmax	argmax	input	argmax	input		fc3	
	3G	argmax	argmax	argmax	argmax	input		fc3	



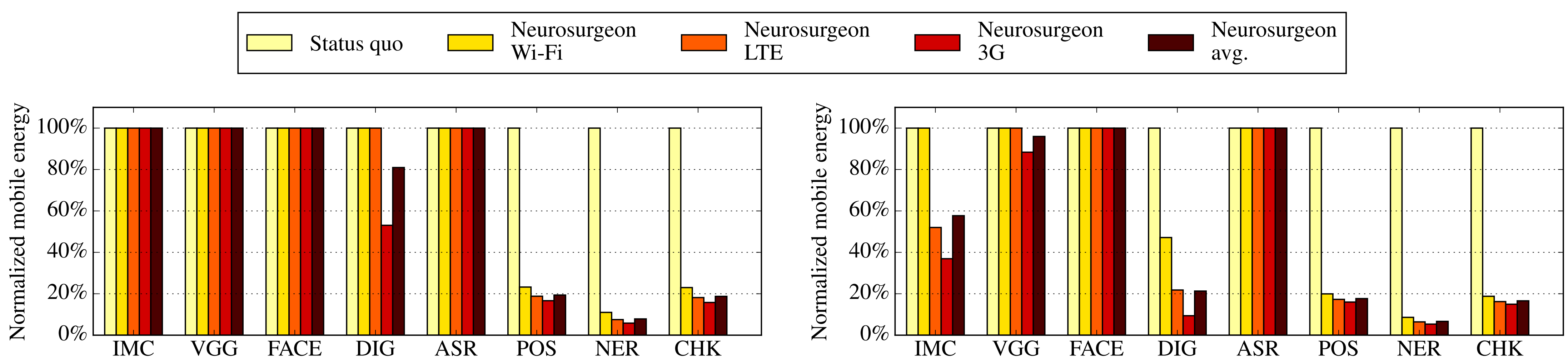
(a) Neurosurgeon using the mobile CPU **(b) Neurosurgeon using the mobile GPU**

Images taken from [1]

Benchmark Results - Power Optimisation

Even in suboptimal cases, 24.2% less energy than status quo

Mobile	Wireless network	Benchmarks							
		IMC	VGG	FACE	DIG	ASR	POS	NER	CHK
CPU	Wi-Fi	input	input	input	input	input	fc3		
	LTE	input	input	input	input	input	fc3		
	3G	input	input	input	argmax	input	fc3		
GPU	Wi-Fi	input	input	input	argmax	input	fc3		
	LTE	pool5	input	input	argmax	input	fc3		
	3G	argmax	argmax	input	argmax	input	fc3		

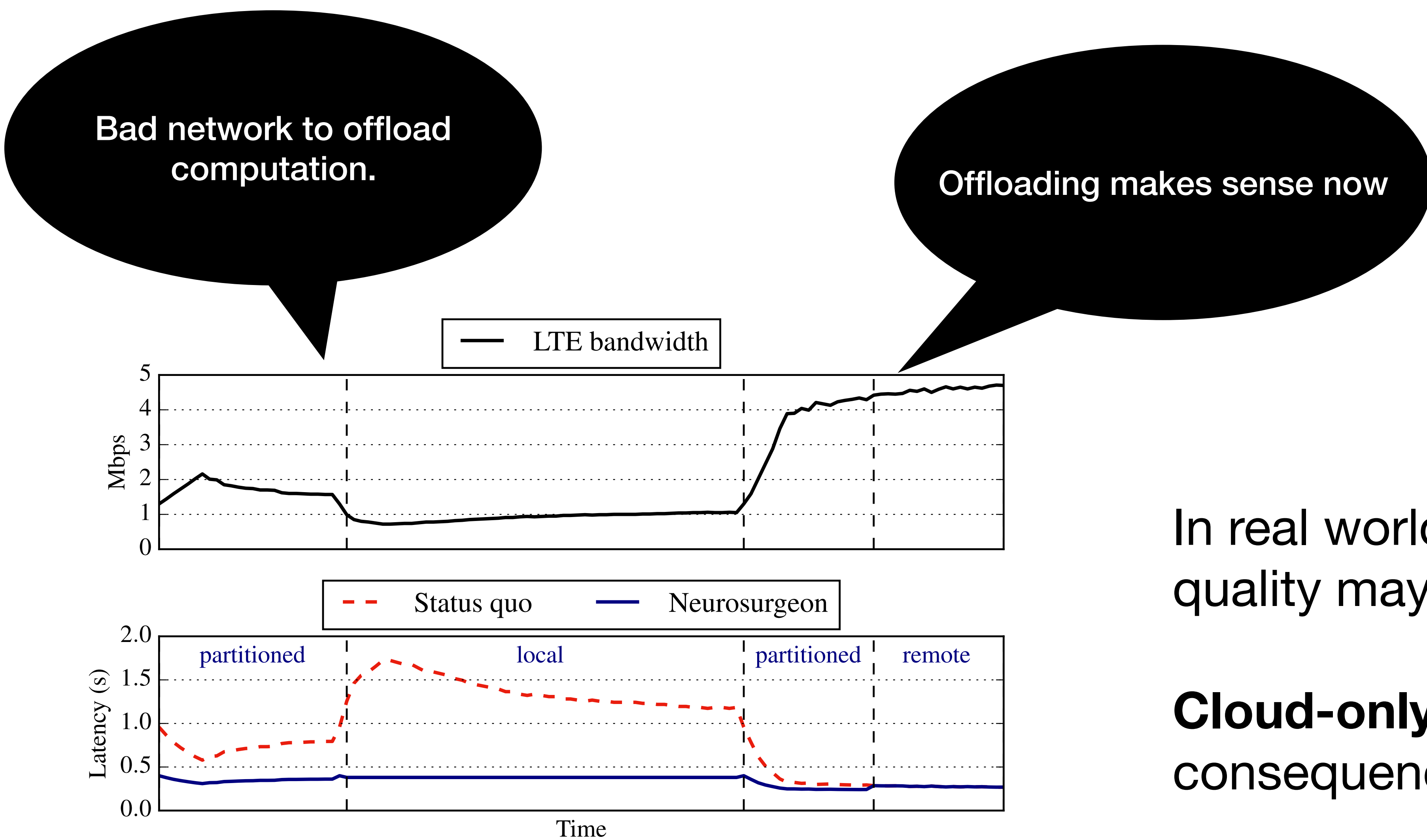


(a) Neurosurgeon using the mobile CPU

(b) Neurosurgeon using the mobile GPU

Images taken from [1]

Testing under Network Variation

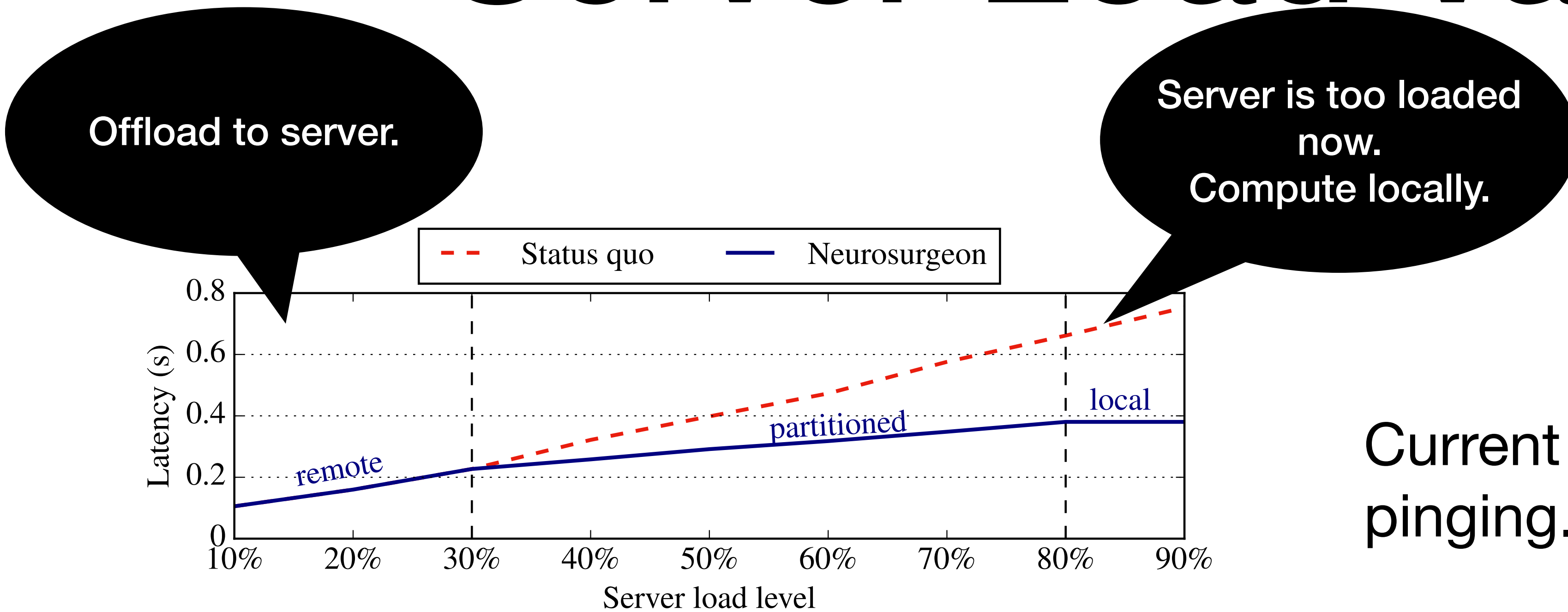


In real world scenarios, network quality may vary.

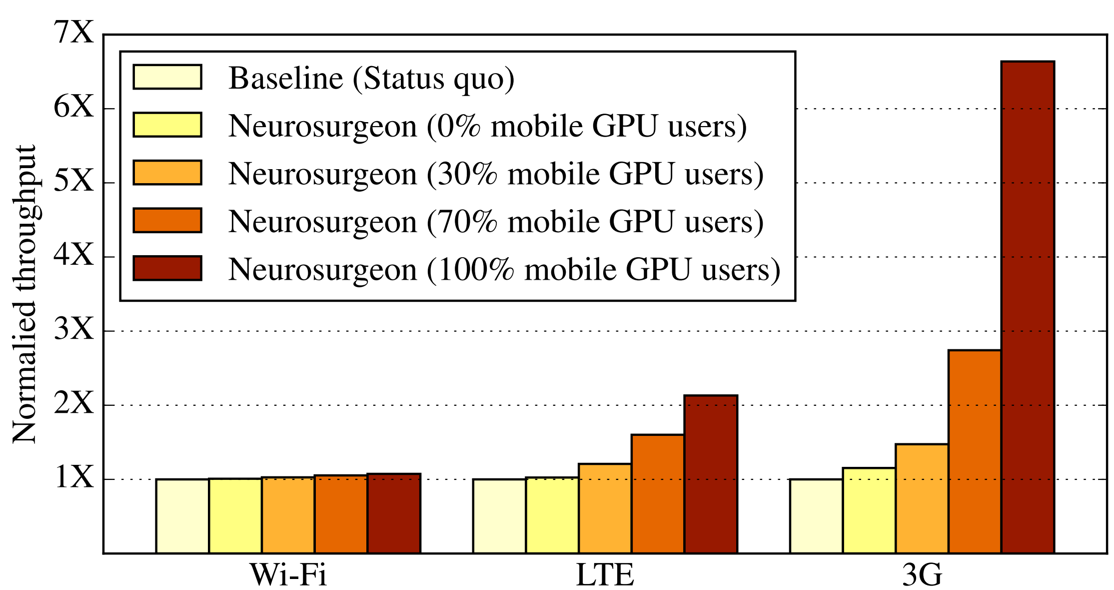
Cloud-only will suffer the consequences.

Neurosurgeon dynamically adapts offloading to mitigate the problem.

Testing under Server Load Variation



End-to-end latency of AlexNet
Mobile CPU-only, transfers via Wi-Fi



Current server load is determined by pinging.

Avoid latency, by taking server load into consideration.

This strategy is further **dropping server load**, allowing for more user queries to be served.

Results

- **End-to-end latency** improvement:
  - average: 3.1x
  - up to: 40.7x
- **Energy consumption** improvement:
  - average: 59.5%
  - up to: 94.7%
- **Datacenter throughput** improvement:
  - average: 1.5x
  - up to: 6.7x

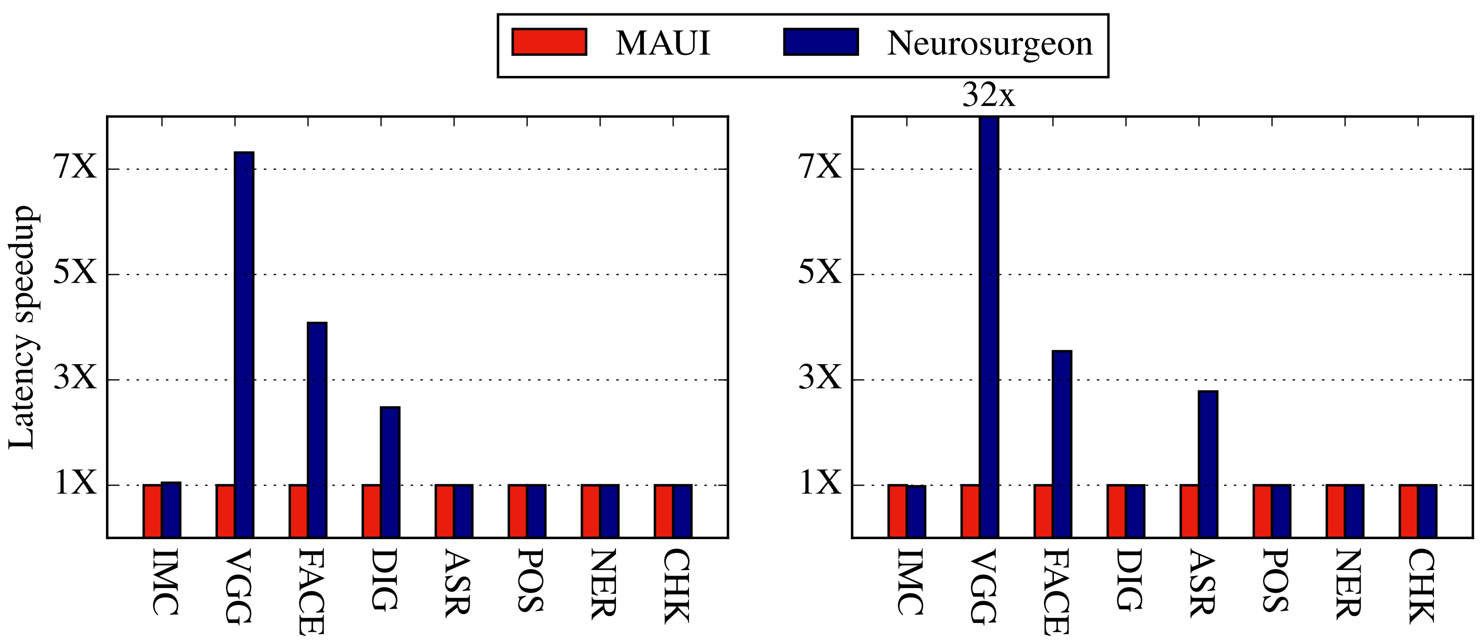
Relevant work

Popular computation offloading solutions

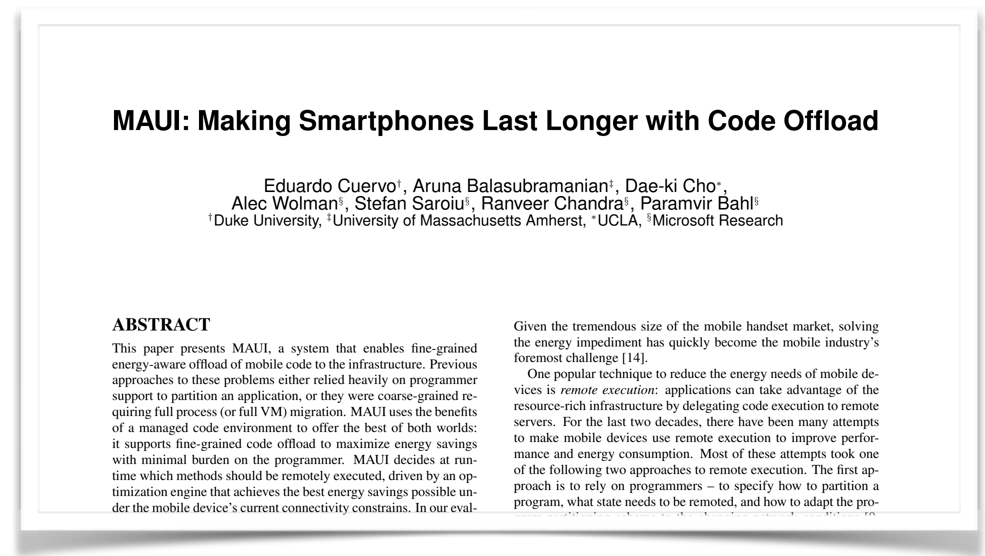
	MAUI [34]	Comet [35]	Odessa [36]	CloneCloud [37]	Neurosurgeon
No need to transfer program state			✓		✓
Data-centric compute partitioning					✓
Low/no runtime overhead	✓		✓	✓	✓
Requires no application-specific profiling		✓			✓
No programmer annotation needed		✓	✓	✓	✓
Server load sensitive			✓		✓

Image taken from [1]

Benchmark Result - MAUI



(a) using the mobile CPU (b) using the mobile GPU



MAUI: Making Smartphones Last Longer with Code Offload

Eduardo Cuervo†, Aruna Balasubramanian‡, Dae-ki Cho*, Alec Wolman§, Stefan Saroiu§, Ranveer Chandra§, Paramvir Bahl§
†Duke University, ‡University of Massachusetts Amherst, *UCLA, §Microsoft Research

ABSTRACT

This paper presents MAUI, a system that enables fine-grained energy-aware offload of mobile code to the infrastructure. Previous approaches to these problems either relied heavily on programmer support to partition an application, or they were coarse-grained requiring full process (or full VM) migration. MAUI uses the benefits of a managed code environment to offer the best of both worlds: it supports fine-grained code offload to maximize energy savings with minimal burden on the programmer. MAUI decides at run-time which methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible under the mobile device's current connectivity constrains. In our eval-

Given the tremendous size of the mobile handset market, solving the energy impediment has quickly become the mobile industry's foremost challenge [14].

One popular technique to reduce the energy needs of mobile devices is *remote execution*: applications can take advantage of the resource-rich infrastructure by delegating code execution to remote servers. For the last two decades, there have been many attempts to make mobile devices use remote execution to improve performance and energy consumption. Most of these attempts took one of the following two approaches to remote execution. The first approach is to rely on programmers – to specify how to partition a program, what state needs to be remoted, and how to adapt the pro-

MAUI scheduling for a layer depends on previous invocations.

	MAUI	Neurosurgeon
Partitioning	Control-based	Data-centric
Profiling	Dynamic	Static
Partitioning Granularity	Per annotated function	Per layer
Optimises for	Power efficiency	Latency XOR Power Efficiency
Specificity	General	DNN Specific

Images taken from [1] and [4]

Privacy Preserving Shared Models

- Based on the edge computing paradigm.
- Models where a general model is trained in the cloud and online learning is supplementing this model on the device [8].

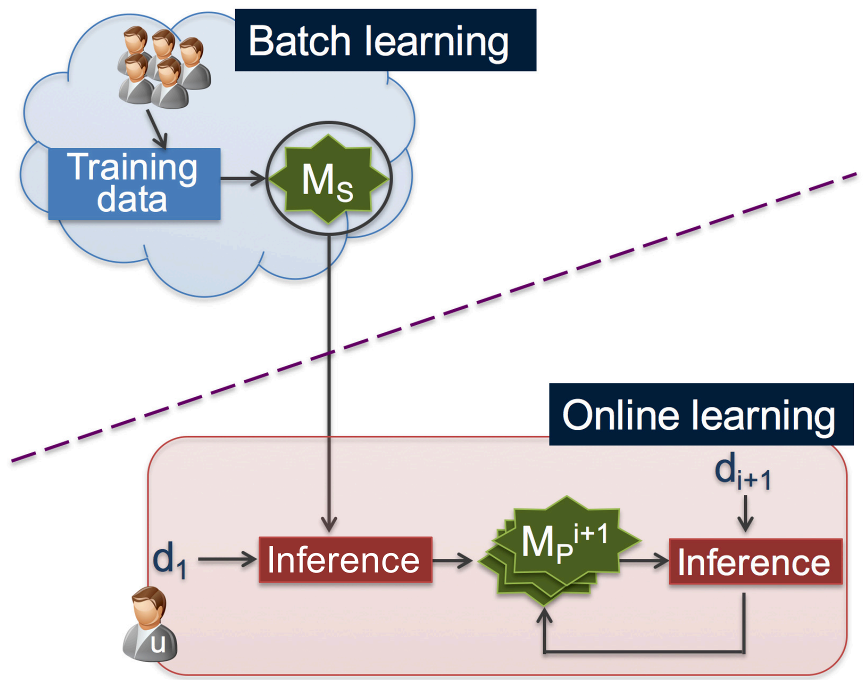


Image taken from [8]

Review & Critique

Review: The good parts

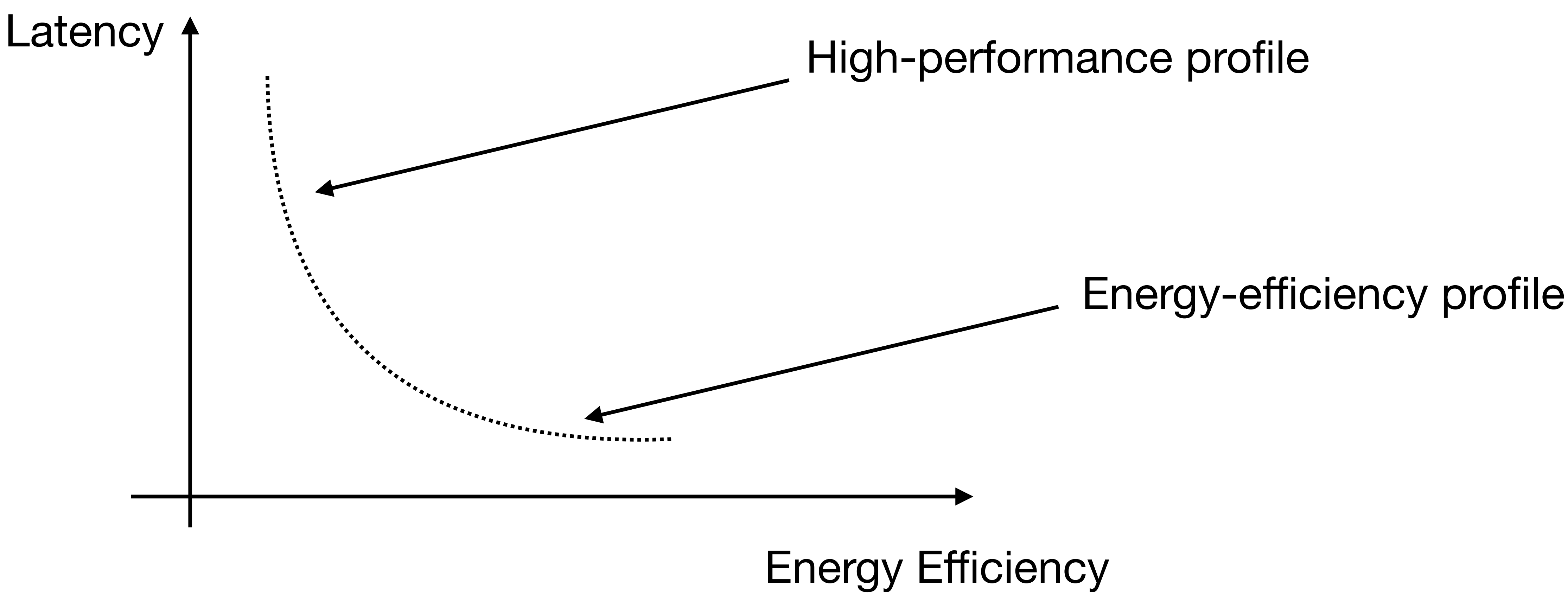
- The name! :)
- Brand new paper published in ASPLOS ’17 *(1 citation from Cambridge* [5]*)*
- Rational extension of current model of execution based on SoC developments.
- All around benchmarks, substantial speedups.
- Inclusive of GPU computation and different network setups.

Critique

- DNN specific (in contrast with MAUI)
- Profiling has hardcoded the regression models for each type of layer (difficult to expand, does not learn how to assess)
    - How would an rNN get split with Neurosurgeon?
- Profiler assumes one type of hardware server-side.
    - Different sized containers based on load.
    - Different datacenter forwarding behind load balancer.
- Adoption: NVIDIA Tegra K1 is a high-end SoC
    - Lower-end processors may shift offloading to the cloud.

Critique

- Distinct optimisation for latency, energy efficiency.
  - Why not offer a Pareto's optimality curve and pick point based on user profile?



Critique

Input:

N: number of layers in the DNN

$\{L_i | i = 1 \cdots N\}$: layers in the DNN

$\{D_i | i = 1 \cdots N\}$: data size at each layer

$f, g(L_i)$: regression models predicting the latency and power of executing L_i

K: current datacenter load level

B: current wireless network uplink bandwidth

PU: wireless network uplink power consumption

procedure PARTITIONDECISION

  for each $i \; in \; 1 \cdots N$ **do**

    $TM_i \leftarrow f_{mobile}(L_i)$

    $TC_i \leftarrow f_{cloud}(L_i, K)$

    $PM_i \leftarrow g_{mobile}(L_i)$

    $TU_i \leftarrow D_i / B$

  if $OptTarget == latency$ **then**

    return $\underset{j=1 \cdots N}{\arg\min}(\sum_{i=1}^{j} TM_i + \sum_{k=j+1}^{N} TC_k + TU_j)$

  else if $OptTarget == energy$ **then**

    return $\underset{j=1 \cdots N}{\arg\min}(\sum_{i=1}^{j} TM_i \times PM_i + TU_j \times PU)$

Smartphones support multitasking.
Why not include `K_mobile`?

Suggestions

- Work with model decomposition and compression algorithms to push more computation locally (such as DeepX [6])
- Other hardware could be taken into consideration (e.g. DSP) for further efficiency (such as DeepEar [7])
- Could Reinforcement Learning be of any help in learning how to partition instead of static profiler?
- Offloading to devices in local network. (MAUI [4])

Thank you

Q&A

Stefanos Laskaridis
sl829@cam.ac.uk

References

1. Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., & Tang, L. (2017). Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, 615–629. https://doi.org/10.1145/3037697.3037698

2. Stanford CS231n - Andrej Karpathy
   http://cs231n.github.io/convolutional-networks/

3. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems, 1–9. https://doi.org/http://dx.doi.org/10.1016/j.protcy.2014.09.007

4. Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010). Maui. Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services - MobiSys '10, 49. https://doi.org/10.1145/1814433.1814441

5. Zhao, J., Mortier, R., Crowcroft, J., & Wang, L. (n.d.). User-centric Composable Services : A New Generation of Personal Data Analytics Normalised Inference Time (%). Retrieved from https://arxiv.org/pdf/1710.09027.pdf

References

6. Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., & Kawsar, F. (2016). DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices. 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2016 - Proceedings, (1). https://doi.org/10.1109/IPSN.2016.7460664

7. Lane, N. D., Georgiev, P., & Qendro, L. (2015). DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments using Deep Learning. Ubicomp, 283–294. https://doi.org/10.1145/2750858.2804262

8. Servia-Rodriguez, S., Wang, L., Zhao, J. R., Mortier, R., & Haddadi, H. (2017). Personal Model Training under Privacy Constraints. Retrieved from http://arxiv.org/abs/1703.00380