DNNTune: Automatic Benchmarking DNN Models for Mobile-cloud Computing

Chunwei Xia, Jiacheng Zhao, Huimin Cui, Jingling Xue, Xiaobing Feng

Institute of Computing Technology, Chinese Academy of Sciences

Work in conjunction with Prof. Jingling Xue, UNSW, Australia

HiPEAC, Bologna, Italy, Jan 20, 2020

Outline

- Introduction and Backgrounds
- DNNTune Framework
- Experimental Setup
- Mobile-only Optimal Deployment
- Analyzing Model Partitioning
- Analyzing Influence Factors
- Conclusion

Introduction and Backgrounds

- Deep Neural Networks (DNNs) are increasingly adopted in AI applications.
  - Image processing
  - Language translation
  - Speech recognition
  - …


- Traditionally DNNs are deployed in the cloud, now moving towards the edge.
- HW Accelerator：
  - Qualcomm: Snapdragon 855+(Hexagon 690+Adreno 640)
  - HiSilicon: Kirin 980(NPUx2, Cambricon)
  - Samsung: Exynos 9825(NPU+Mali-G76 MP12)
  - MediaTek: Helio P90(APU 2.0)

Andrey Ignatov etc, *AI Benchmark: All About Deep Learning on Smartphones in 2019*

Introduction and Backgrounds

DL is used as core building blocks in 171/221 mobile apps

usage	detailed usage	as core feature
image: 149	photo beauty: 97	94 (96.9%)
	face detection: 52	44 (84.6%)
	augmented reality: 19	5 (26.3%)
	face identification: 8	7 (87.5%)
	image classification: 11	6 (54.5%)
	object recognition: 10	9 (90%)
	text recognition:11	4 (36.3%)
text:26	word&emoji prediction: 15	15 (100%)
	auto-correct: 10	10 (100%)
	translation: 7	3 (42.8%)
	text classification: 4	2 (50%)
	smart reply: 2	0 (0%)
audio: 24	speech recognition: 18	7 (38.9%)
	sound recognition: 8	8 (100%)
other: 19	recommendation: 11	2 (18.1%)
	movement tracking: 9	4 (44.4%)
	simulation: 4	4 (100%)
	abnormal detection: 4	4 (100%)
	video segment: 2	1 (50%)
	action detection: 2	0 (0%)
total: 211		171 (81.0%)

Table 1: The DL usage in different apps. Note: as one app may have multiple DL uses, the sum of detailed usage (column 2) might exceed the corresponding coarse usage (column 1).

Mobile DL frameworks are gaining traction

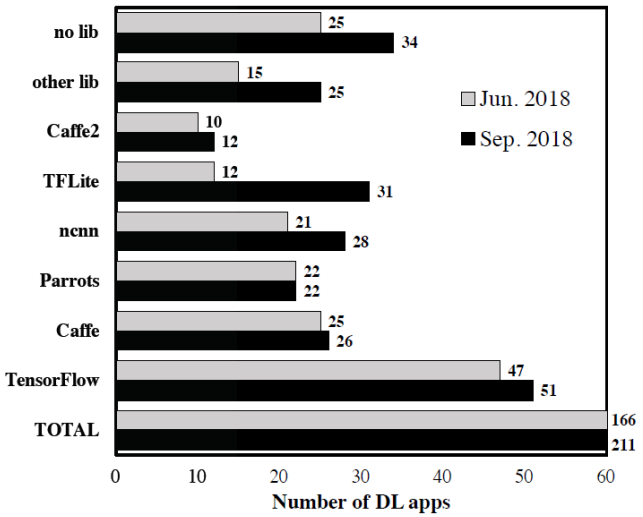
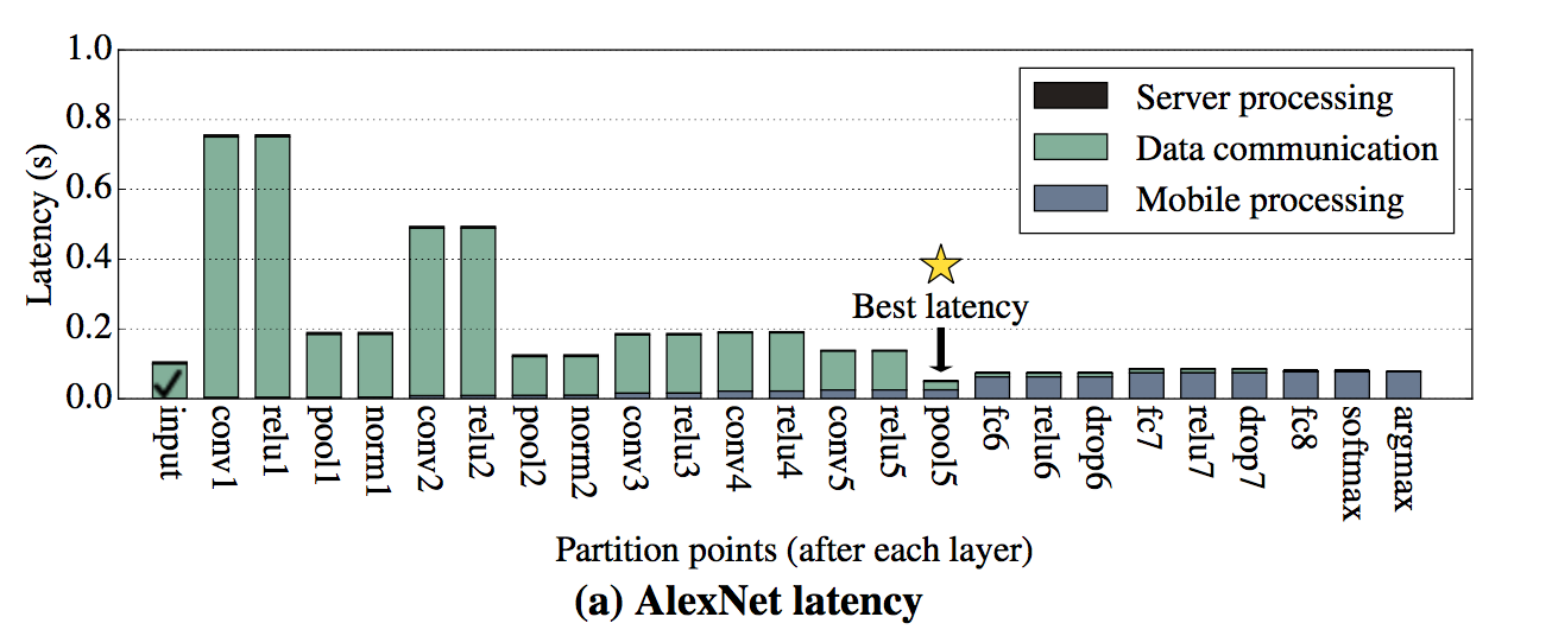


Figure 4: Numbers of DL apps using various mobile DL frameworks. "other lib": the DL apps developed on the frameworks in Table 2 but not itemized here, e.g. *mace, SNPE, and xnn*. "no lib": apps with DL functions but using no DL frameworks from Table 2. Note that the number in "TOTAL" is lower than the sum of others since some DL apps have integrated multiple frameworks.

Mengwei Xu etc, *A First Look at Deep Learning Apps on Smartphones*, at WWW'19

Introduction and Backgrounds

- New paradigm for DNN inferencing
- Neurosurgeon: Partition the DNN between the edge and cloud
- Reduce **latency** and **energy**



Yiping Kang etc, Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge, at ASPLOS'17

Introduction and Backgrounds

- Diversity of edge devices and DNN models,
  - HW: CPU, memory, ASIC for DNN, battery life, …
  - SW: frameworks/libraries, # of threads …
  - Inference latency, model size, memory usage, network communication…

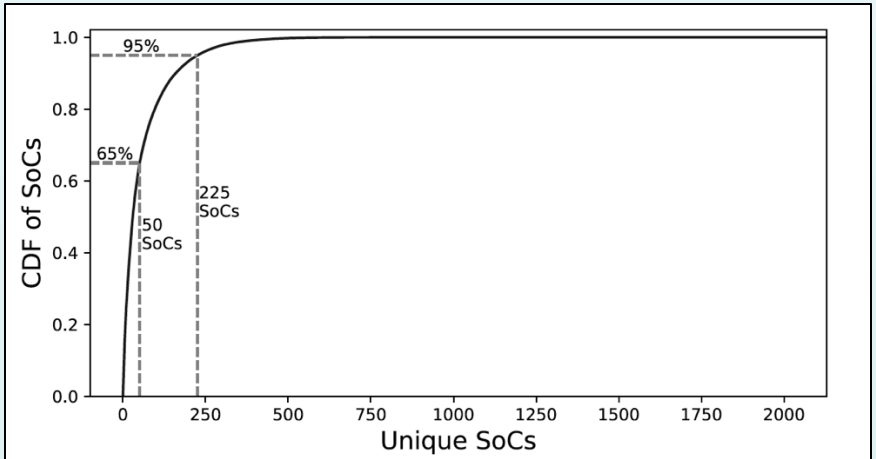


Figure 2: There is no standard mobile SoC to optimize for. The top 50 most common SoCs account for only 65% of the smartphone market.

There are no standard mobile chipset to optimize for

Carole-Jean Wu etc, *A look at smartphones Facebook runs on*, Facebook, Inc

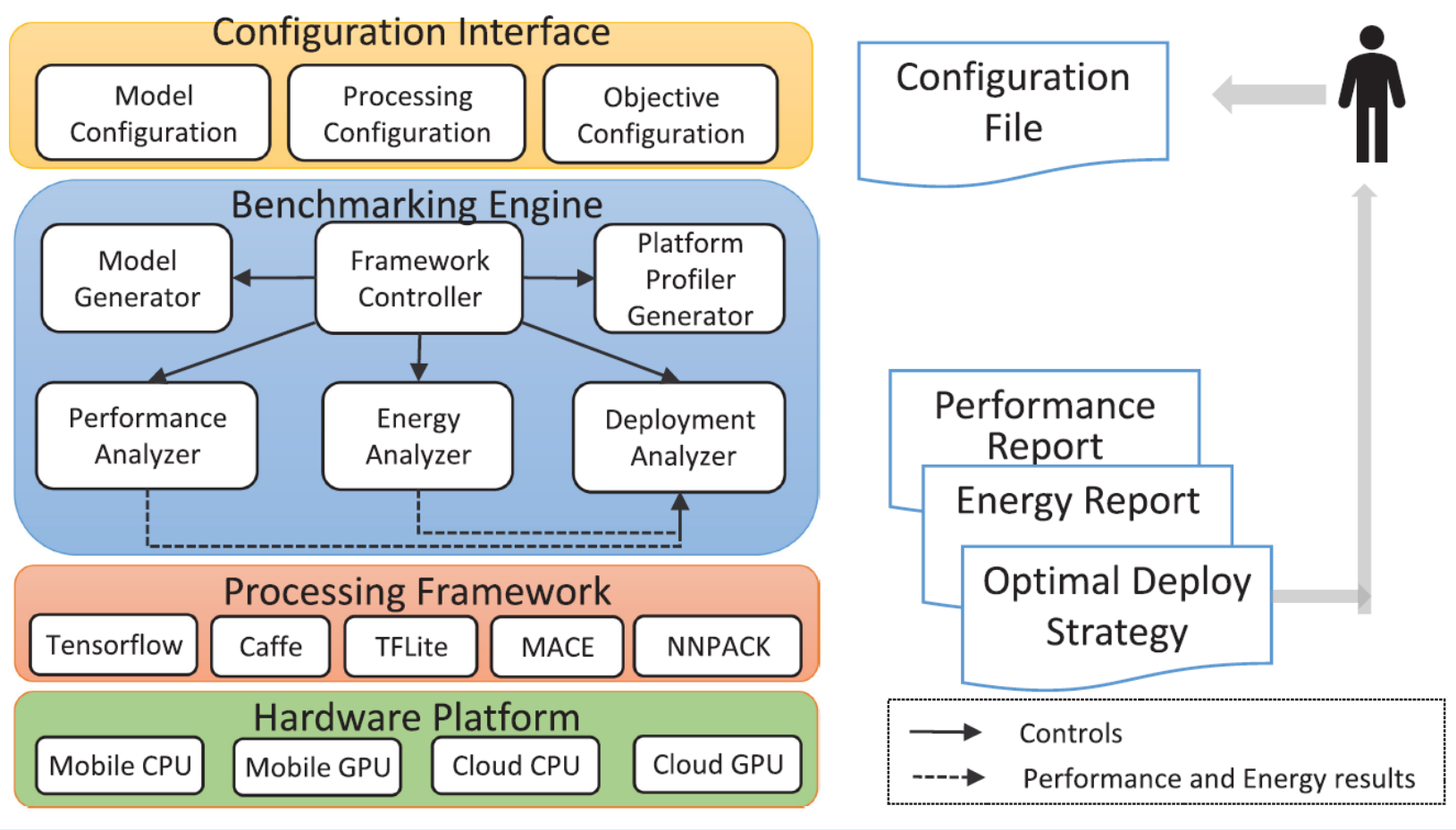
Outline

- Introduction and Backgrounds
- **DNNTune Framework**
- Experimental Setup
- Mobile-only Optimal Deployment
- Analyzing Model Partitioning
- Analyzing Influence Factors
- Conclusion

DNNTune Framework

- Goal
  - Characterize different DNN models on a number of mobile devices.
  - Automatically seek for a partitioning point for mobile-cloud coordinate computing
- Challenges
  - Diversity
  - Layer-wise profiling
    - Latency from several milliseconds to several seconds

DNNTune Framework



The DNNTune framework overview.

DNNTune Framework

- How to use the framework
  - Configure DNN model
  - Processing platform
  - Analysis objective

Model
name: mobilenet-v2
file: path/to/mobilenet-v2.pb
input_name: input
input_shape:[1,224,224,3]
output_name: mobilenetv2/predictions
data_type: float

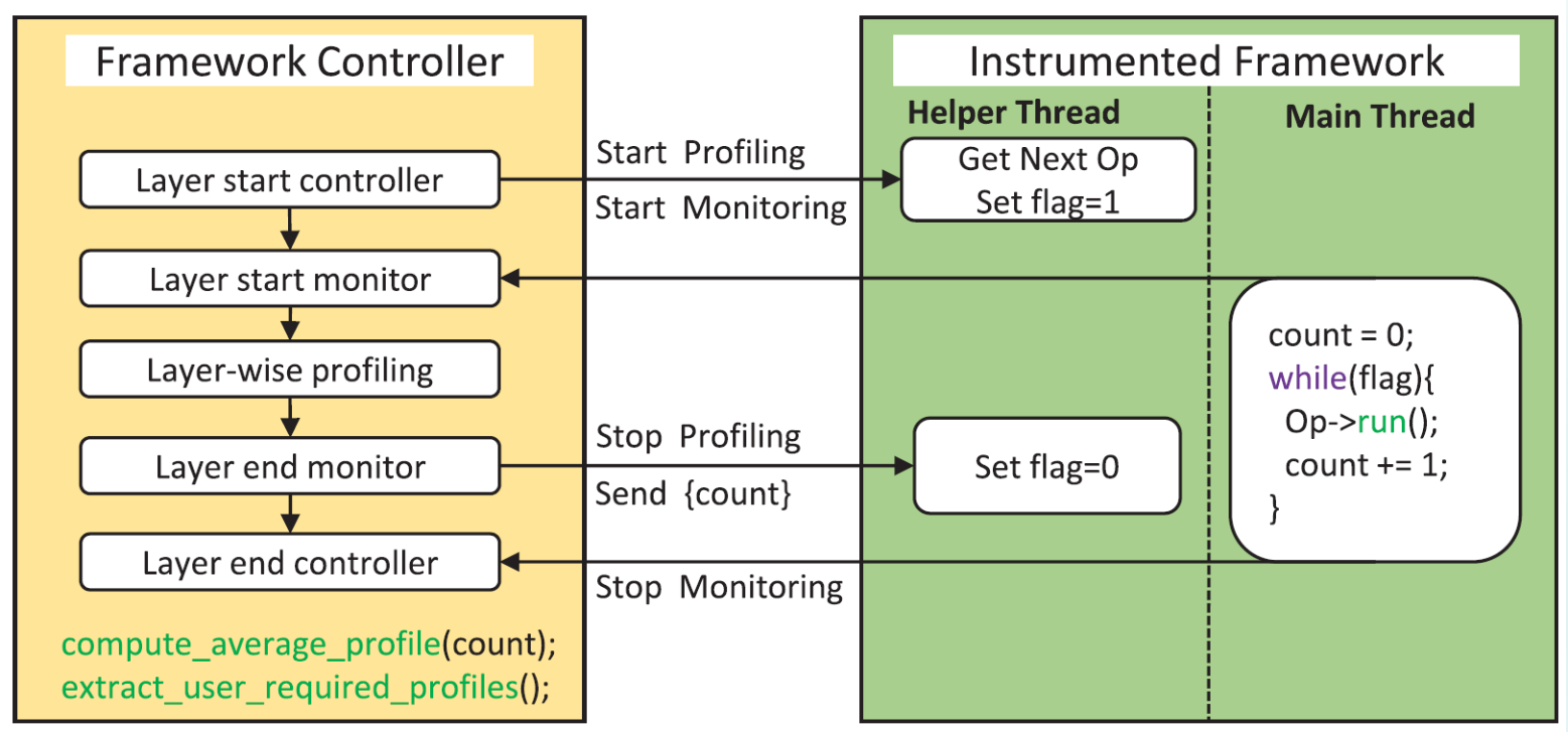
Objective
objective: energy-first

Processing
Processing_unit: mobile_CPU, cloud_CPU
CPU_threads: 4
CPU_affinity: big+little
framework: tensorflow, MACE
DNN_library: eigen, NNPACK
profiling_tools: Snapdragon_profiler, Simpleperf
energy_profiling: true
profiling_events: CPUUtilization, Mem_footprint, LLCmiss

An example for the configuration interface.

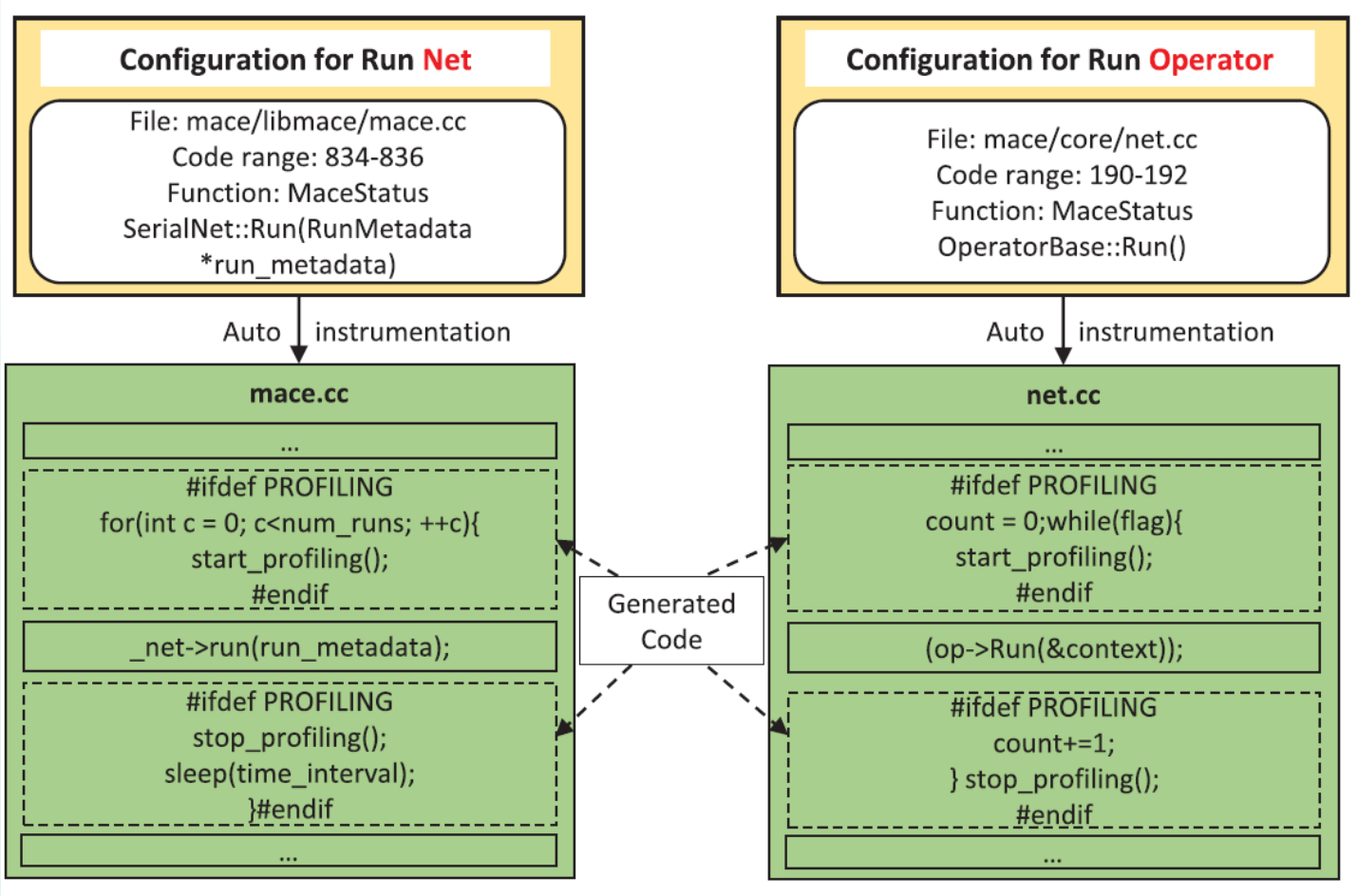
DNNTune Framework

- How to collect layer-wise execution info
  - Snapdragon Profiler samples the system power every 50 ms,
  - “bottleneck_3_2” in MobileNet-V2 is 5 ms



The instrumentation for layer-wise profiling in DNNTune.

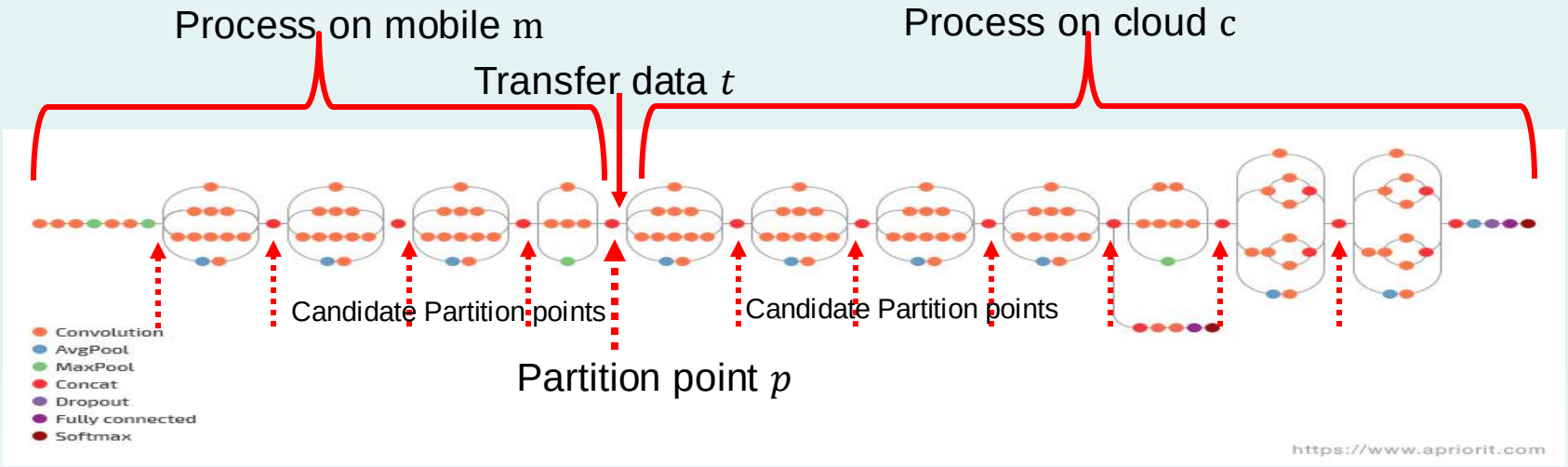
DNNTune Framework



Mechanism for auto instrumentation.

DNNTune Framework

- Deployment Analyzer
  1. Running and profiling.
  2. Computing for each partitioning point.
     1. $T^p = T_{\mathrm{m}} + T_{\mathrm{t}} + T_c$
     2. $E^p = E_{\mathrm{m}} + E_{\mathrm{t}}$
     3. T: latency, E:energy p: partition point, m:mobile platform processing, c: cloud platform processing, t:transfer data
  3. Traversing.



https://www.apriorit.com/images/articles/applying_inception_v3/figure-1.jpg

Outline

- Introduction and Backgrounds
- DNNTune Framework
- **Experimental Setup**
- Mobile-only Optimal Deployment
- Analyzing Model Partitioning
- Analyzing Influence Factors
- Conclusion

Experimental Setup

Small RNN
Big RNN
Small MLP
Big MLP
Big CNN
Small CNN

Table 1. DNN Specifications

Name	Input	Output	Layers	# Params	FLOPs
rnn_ptb_small	[20, 200]	[20, 10K]	2	2.65M	14.8M
DeepSpeech	[16, 494]	[16, 29]	6	29M	464M
mnist_mlp	[784]	[10]	5	13.9M	13.9M
Transformer	[1,7]	[1, 14]	12	49M	-
Inception-V1	[224, 224, 3]	[1K]	22	6.79M	3.19 G
Inception-V3	[299, 299, 3]	[1K]	48	22.75M	6 G
ResNet-50	[224, 224, 3]	[1K]	50	25.6M	3.8 G
Vgg16	[224, 224, 3]	[1K]	16	138M	16 G
MobileNet-V1	[224, 224, 3]	[1K]	15	4.2M	576M
MobileNet V2	[224, 224, 3]	[1K]	20	3.4M	300M
SqueezeNet-V11	[227, 227, 3]	[1K]	15	1.2M	360M
ShuffleNet-V2 0.5×	[224, 224, 3]	[1K]	15	1.4M	41M
DeepLab-V3	[513, 513, 3]	[65, 65, 21]	20	2.1M	17.8 G

“Small” for optimized for mobile devices
“Big” for models that aim to reach the highest accuracy

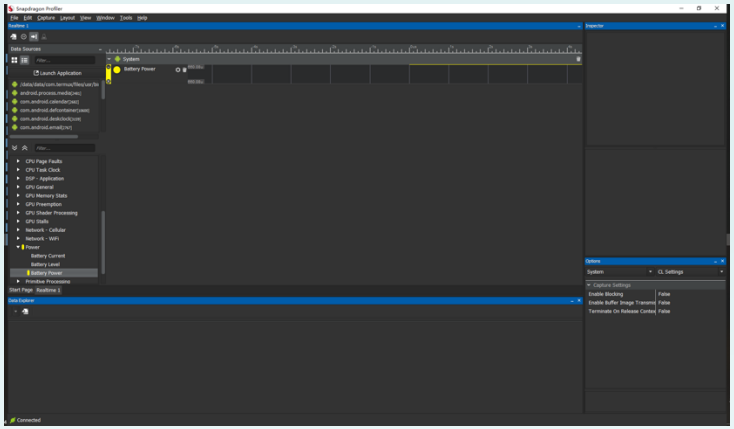
Experimental Setup

- HW
  - Cloud platform:
    - Intel Xeon E5-2620 v4
  - Mobile device with AI accelerator
    - NVIDIA Jetson TX2 with a 256-core Pascal GPU
    - Honor 10 with a Neural Processing Unit (NPU) in Kirin 970
  - Three Mobile devices from high-end to low-end

Device	OnePlus 5T		OnePlus 3		Redmi Note 4X	
SoC	Qualcomm Snapdragon 835		Qualcomm Snapdragon 820		Qualcomm Snapdragon 625	
Process	10 nm		14 nm		14 nm	
Notation	Mobile A-CPU	Mobile A-GPU	Mobile B-CPU	Mobile B-GPU	Mobile C-CPU	Mobile C-GPU
Platform	Kryo 280	Adreno 540	Kryo [2]	Adreno 530	Cortex A53	Adreno 506
Cores/Compute Units	4+4	256	2+2	256	8	96
L1 Cache	64 I+32 I KB | 64 D +32 D KB	-	32 I + 32 D KB	-	32 KB	-
L2 Chche/On Chip Mem	2+1 MB	1,024 KB	1 MB+512 KB	1,024 KB	1 MB	128 + 8 KB
Freq	4×2.45+4×1.85 GHz	710 MHz	2×2.15+2×1.59 GHz	624 MHz	2.0 GHz	650 MHz
RAM	8 GB LPDDR4X		6 GB LPDDR4		3 GB LPDDR3	
OS	Android 8.1		Android 8.1		Android 7.0	

Experimental Setup

- Framework
  - Tensorflow r1.7
  - TFLite
  - MACE
- Energy consumption
  - snapdragon_profiler developed by Qualcomm
  - $E = T_{\mathrm{m}}(e) * p_m(e)$
  - $p_m(e) = \mathrm{p}_m(e_{active}) - \mathrm{p}_m(e_{idle})$

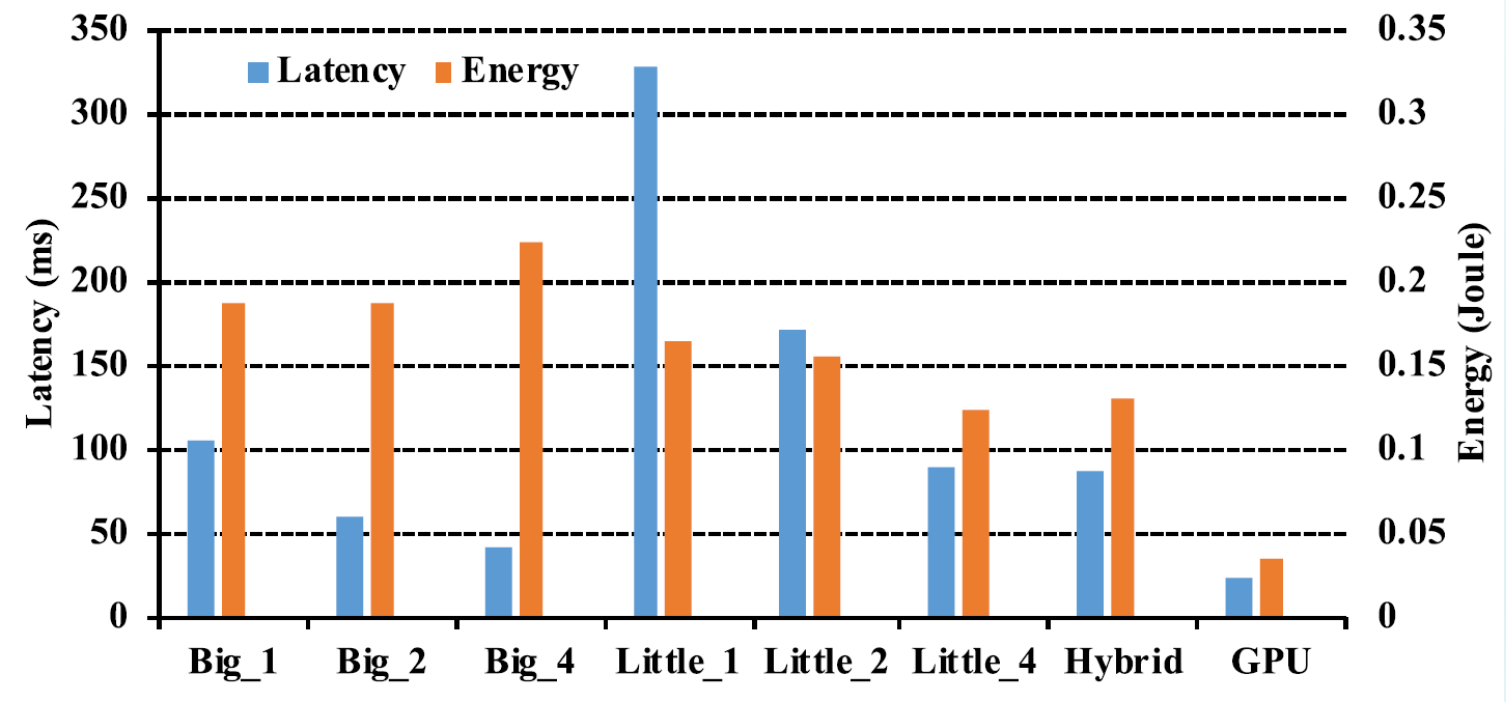


Outline

- Introduction and Backgrounds
- DNNTune Framework
- Experimental Setup
- **Mobile-only Optimal Deployment**
- Analyzing Model Partitioning
- Analyzing Influence Factors
- Conclusion

Mobile-Only Optimal Deployment

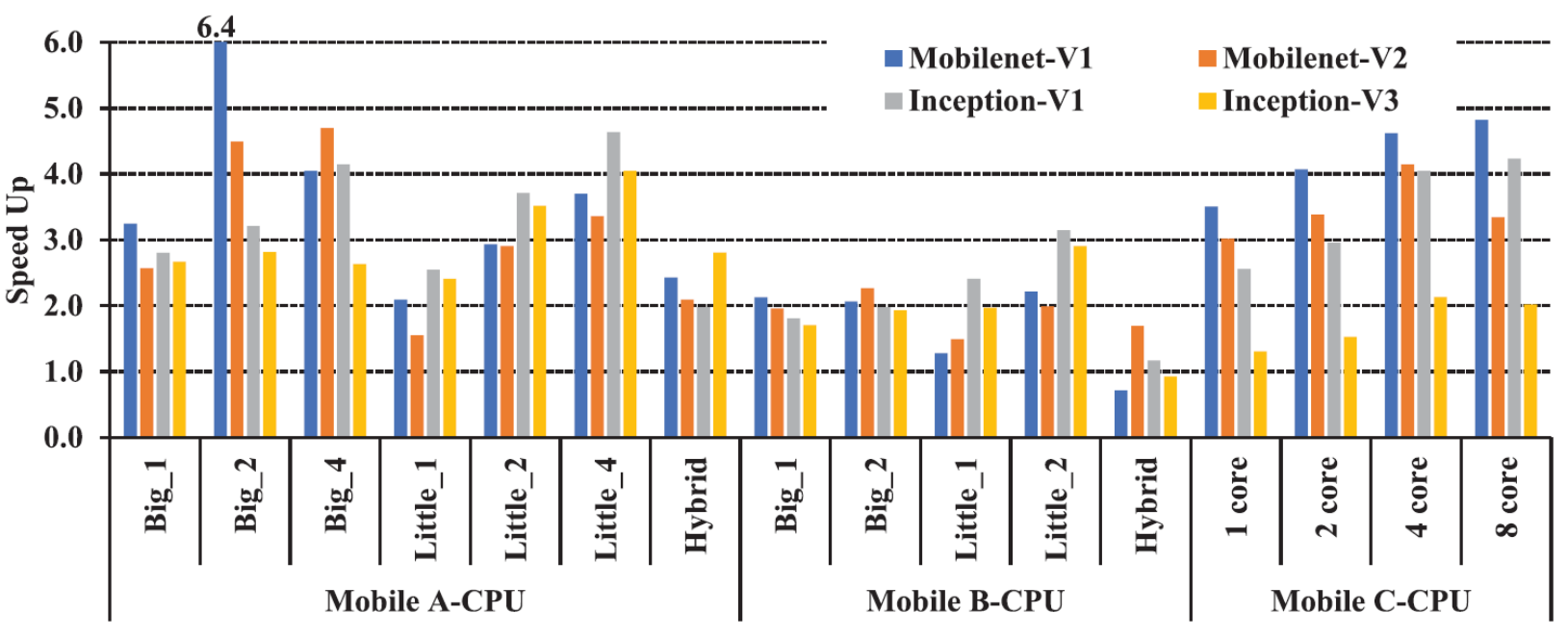
- Only CPU available
  - 4 big cores→best performance (42 ms)
  - 4 little cores →least energy
- GPU provides the best performance and energy efficiency



Latency and energy consumption of MobileNet-V2 on Mobile A when using different resources.

Discussing Model Quantization

- Significant speedup of quantization over non-quantization
- MobileNet-V2 gains 4.7×, 2.0×, and 4.0× on the three mobile CPUs



Outline

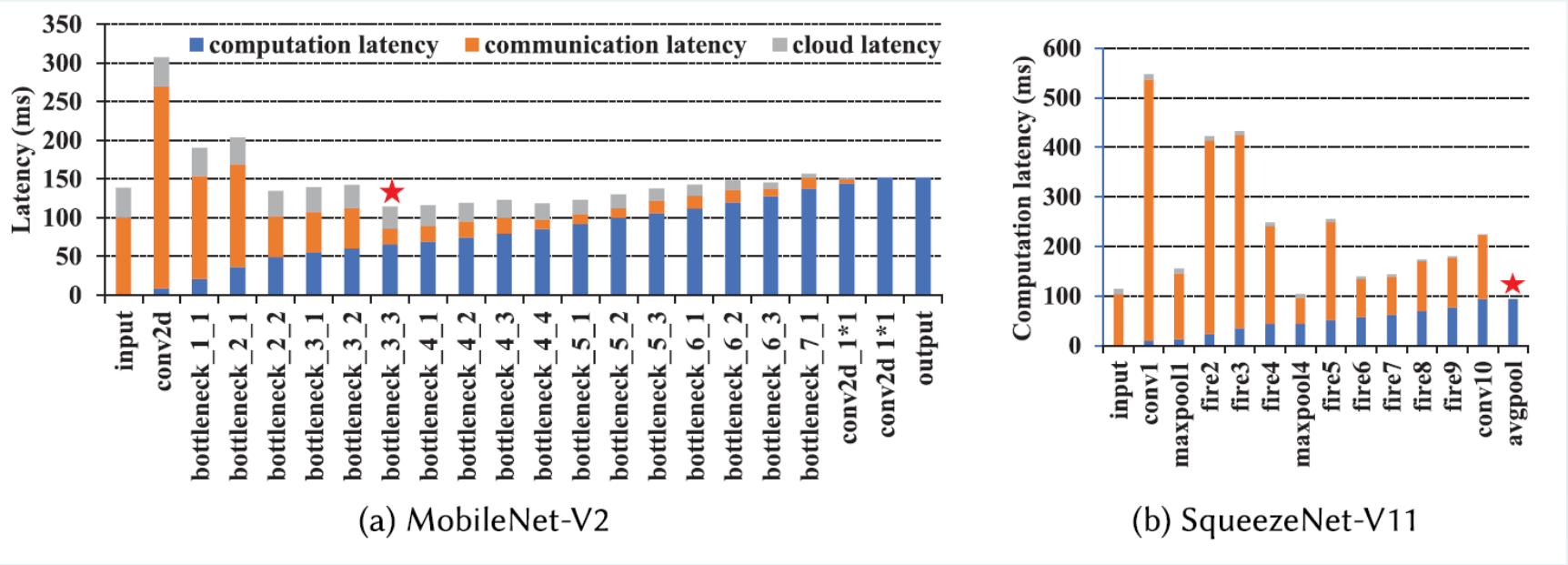
- Introduction and Backgrounds
- DNNTune Framework
- Experimental Setup
- Mobile-only Optimal Deployment
- **Analyzing Model Partitioning**
- Analyzing Influence Factors
- Conclusion

Analyzing Model Partitioning

- Evaluation Methodology
  - CPUs and GPUs of Mobile A, Mobile B, and Mobile C
  - TensorFlow for the evaluation of CPUs and MACE for mobile GPUs.
  - 3G, 4G, and Wi-Fi

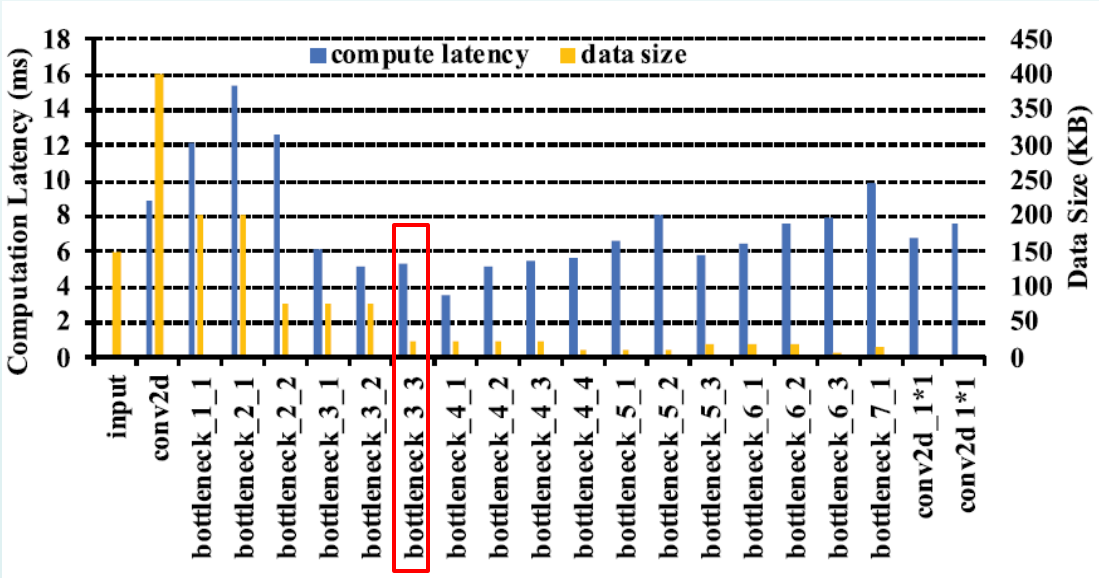
Latency-first Model Partitioning

- MobileNet-V2
  - Bottleneck_3_3 is optimal (115 ms vs 138 ms)
- SqueezeNet-V11
  - Mobile-only is optimal (99 ms vs 111 ms)

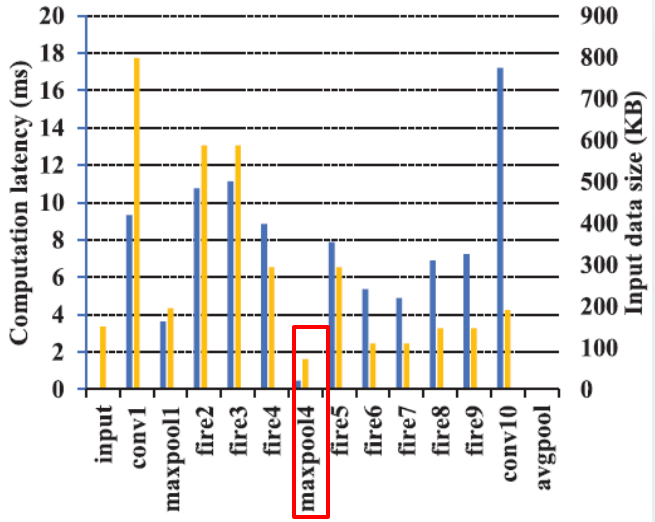


Latency-first Model Partitioning

- Analyzing CNN Behaviors for Model Partitioning
  - communication overhead at the partitioning point to transfer data from the mobile to cloud
  - benefit of processing the latter layers in cloud



(a) MobileNet-V2



(b) SqueezeNet-V11

Latency and energy consumption of MobileNet-V2 on Mobile A when using different resources.

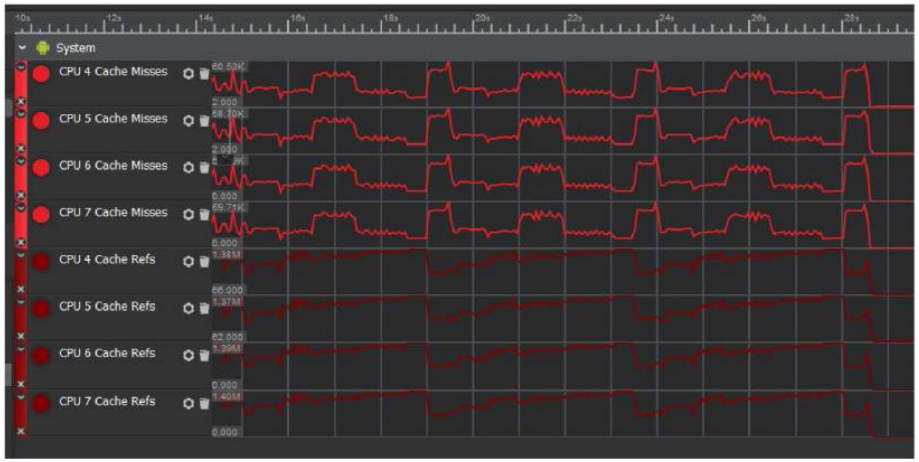
Energy-first Model Partitioning

- *3G.*
  - For MobileNet-V2 and ShuffleNet-V2 0.5x, the mobile-only is the optimal
  - Communication energy overhead is higher than the computation energy
- Wi-Fi
  - Saving 15% energy as most for ShuffleNet-V2 0.5x

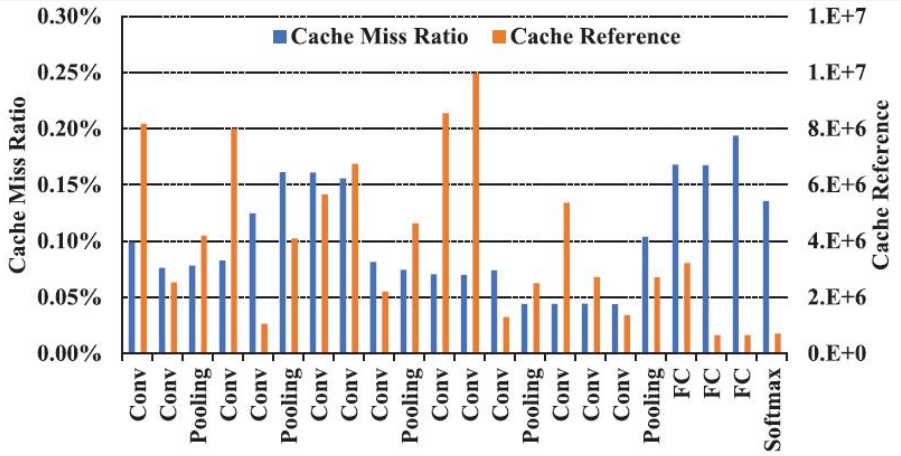
Table 4. Energy Consumption on Three Mobile CPUs Using Wi-Fi for ShuffleNet-V2 0.5×

	mobile-only	cloud-only	mobile-cloud	energy saving	partition point
Mobile A-CPU	0.088	0.121	0.077	12.5%	Stage2
Mobile B-CPU	0.099	0.121	0.084	15.1%	Stage2
Mobile C-CPU	0.055	0.121	0.052	5.5%	Stage4

Layer-wise Profiling



(a) Raw profiling data



(b) cache references and miss ratio

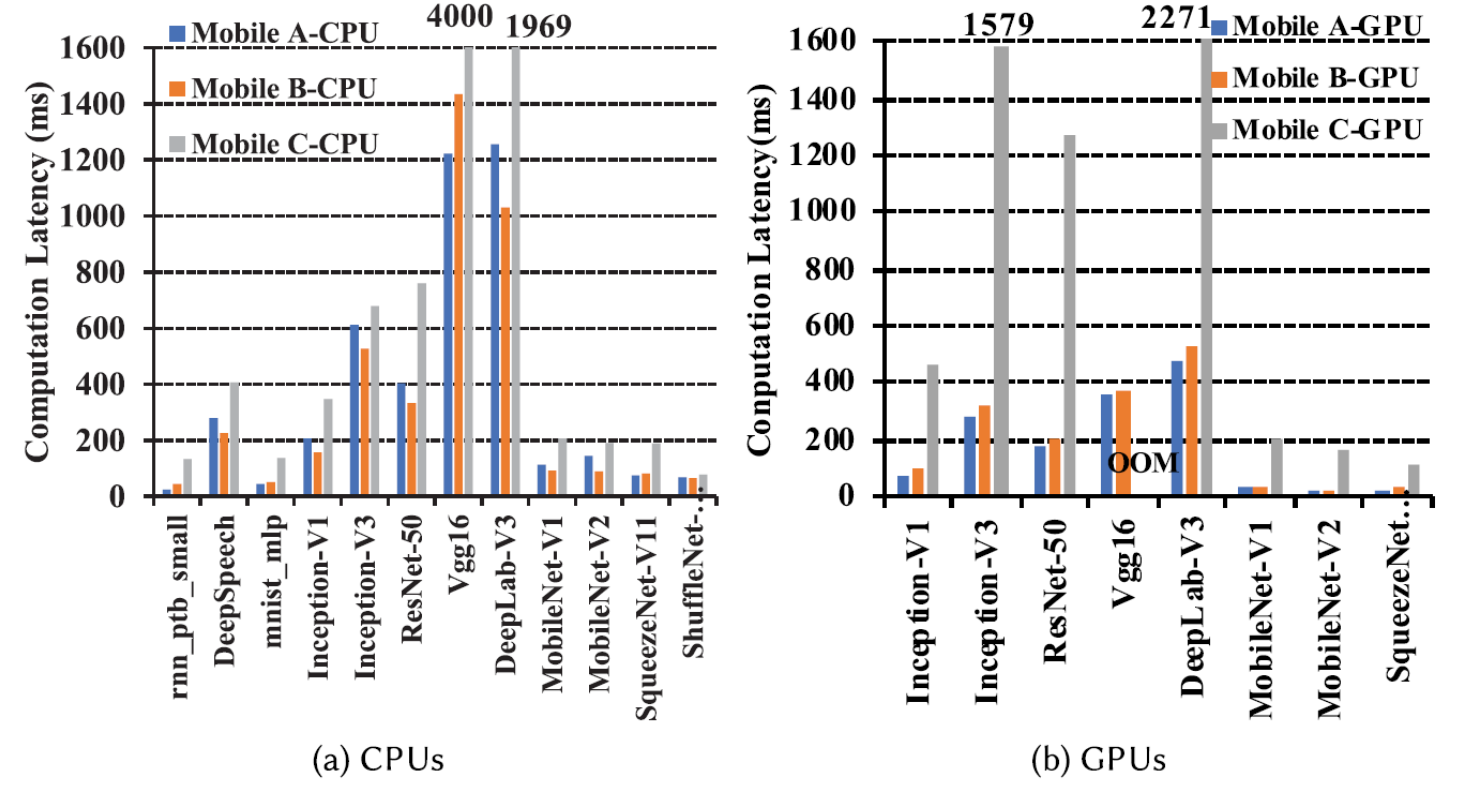
The layer-wise cache references and miss ratio for Vgg16 profiled by DNNTune.

Outline

- Introduction and Backgrounds
- DNNTune Framework
- Experimental Setup
- Mobile-only Optimal Deployment
- Analyzing Model Partitioning
- **Analyzing Influence Factors**
- Conclusion

Analyzing Influence Factors: Processing Unit

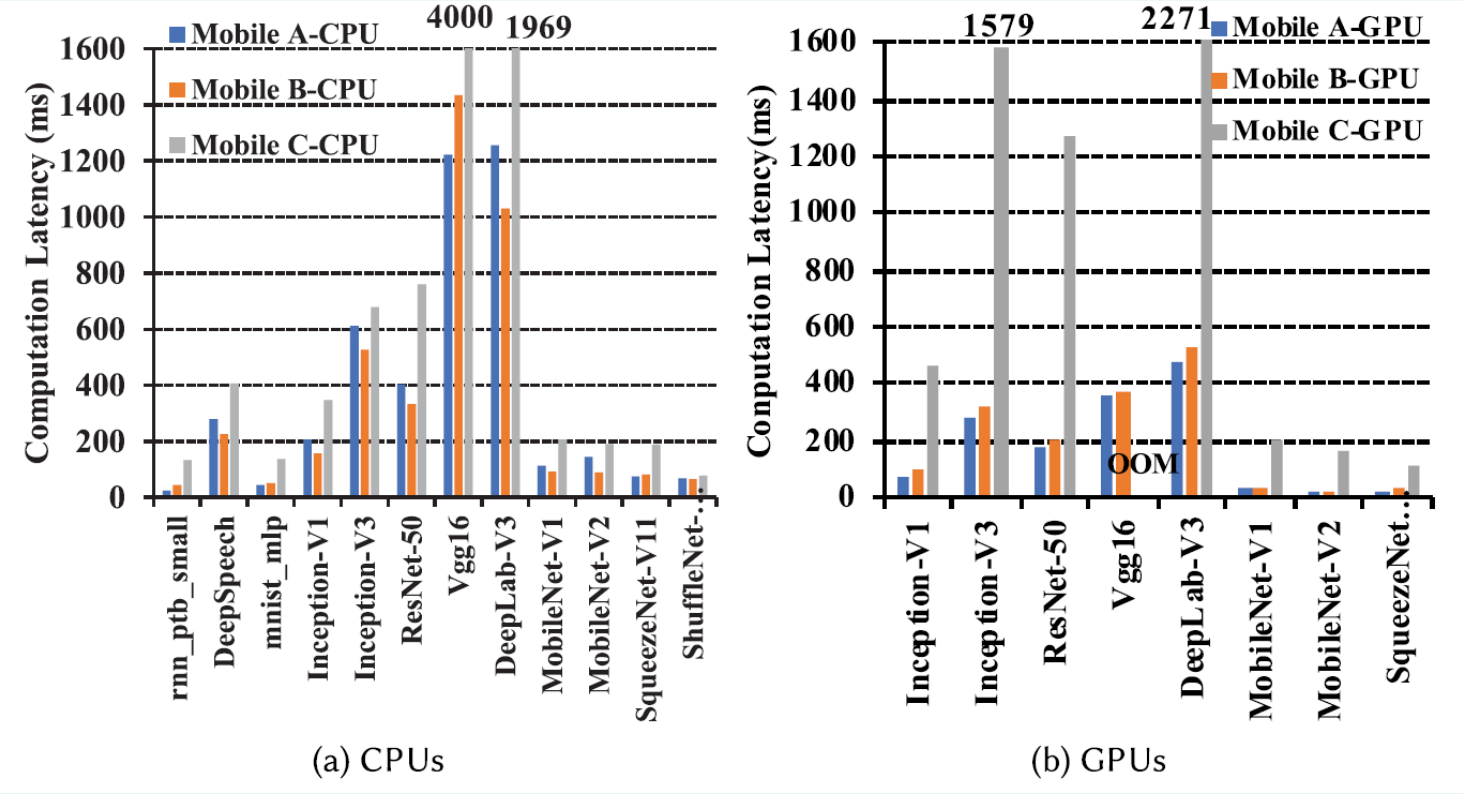
Observation 1: Modern mobile CPUs are powerful enough to process small LSTM/MLP models, but cannot process large LSTM/MLP models and most CNNs unless they are specifically optimized for mobiles.



Computation latency of DNN models on mobile CPUs and GPUs with optimal configuration.

Analyzing Influence Factors: Processing Unit

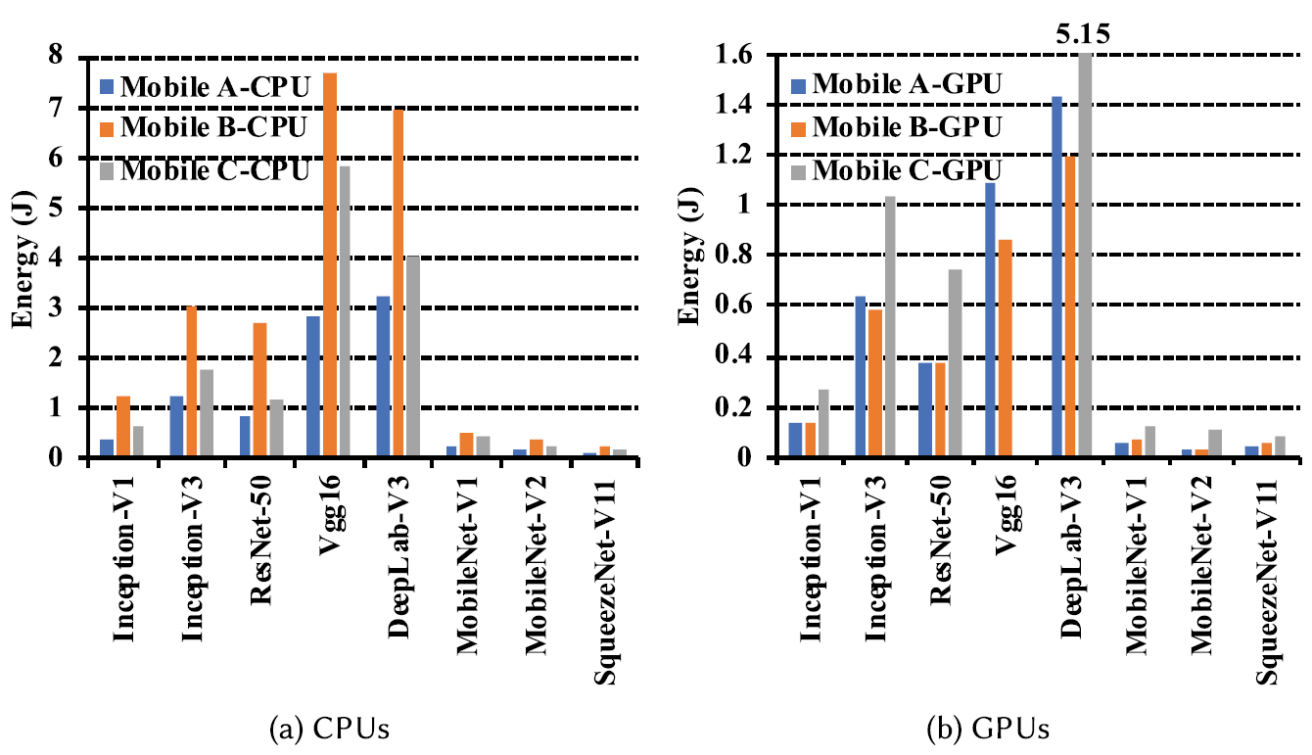
Observation 2: High-end and mid-end mobile GPUs can significantly reduce the computation latency than CPUs, while low-end cannot.



Computation latency of DNN models on mobile CPUs and GPUs with optimal configuration.

Analyzing Influence Factors: Processing Unit

Observation 3: Mobile GPUs are more energy-efficient than Mobile CPUs, especially for high-end and mid-end mobile GPUs.



Computation latency of DNN models on mobile CPUs and GPUs with optimal configuration.

Analyzing Influence Factors: CPU Affinity

Observation 4: Simultaneously using big and little cores does not always benefit.

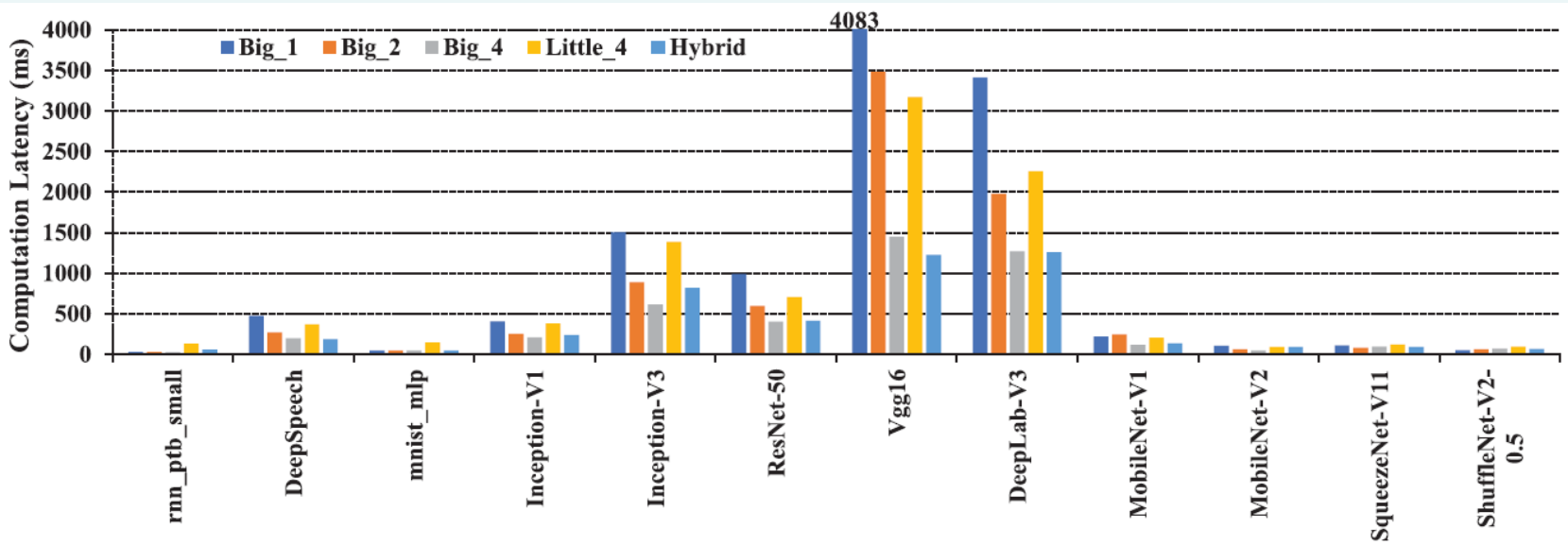


Fig. 14. Performance on Mobile A-CPU varying with CPU affinity and thread number.

Performance on Mobile A-CPU varying with CPU affinity and thread number.

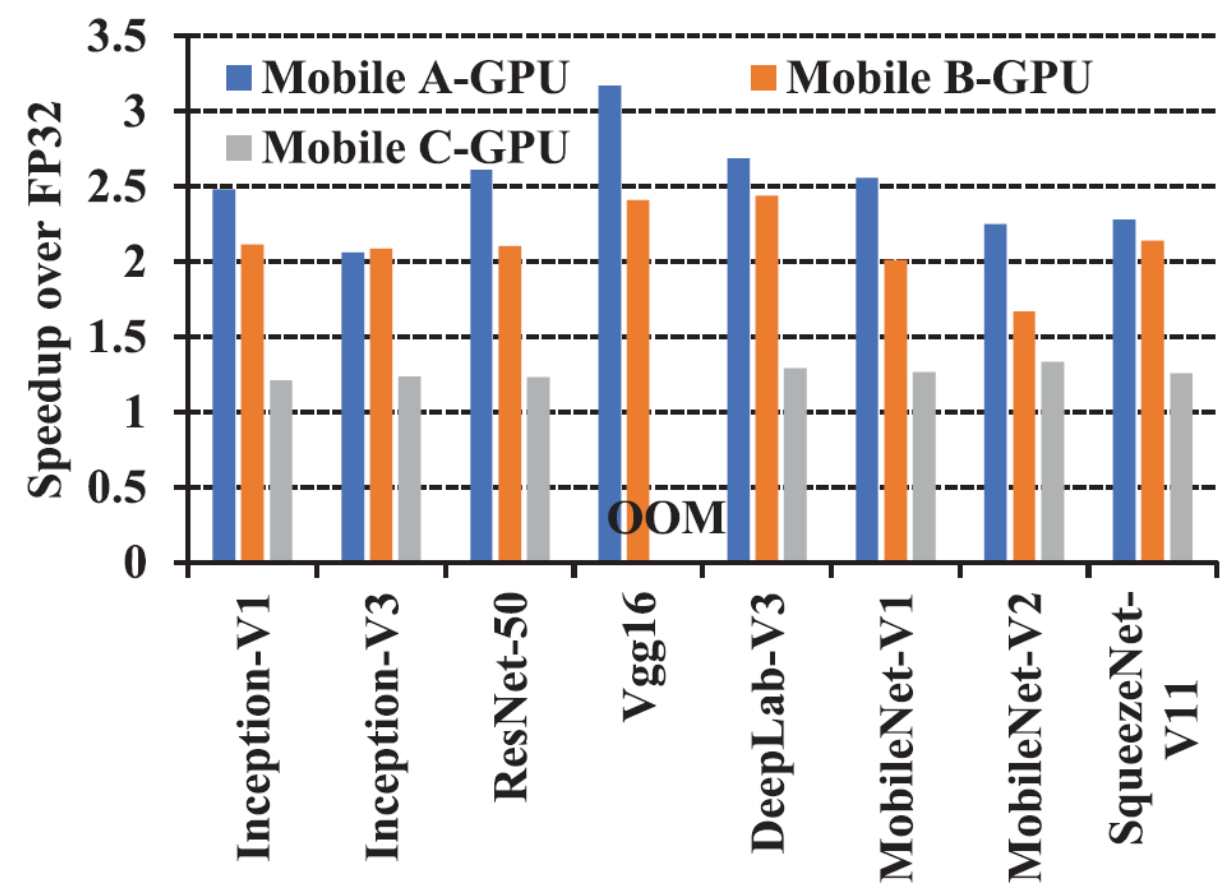
Analyzing Influence Factors: Processing Frameworks

- Latency
  - MACE is the optimal framework for latency.
  - TFLite quantized is the optimal
- Energy
  - There is no such framework that is optimal on energy consumption for all DNN models.
- Memory
  - TFLite and MACE are more memory-efficient than Tensorflow
  - Quantized models can save 74% memory

Table 5. Computation Latency, Energy Consumption, and Memory Usage of Seven CNN Models

	Inception-V1			Inception-V3			ResNet-50			Vgg16			DeepLab-V3			MobileNet-V1		
	Lat	En	M	Lat	En	M	Lat	En	M	Lat	En	M	Lat	En	M	Lat	En	M
TF	207	0.413	110	612	1.812	310	403	1.126	340	1,446	5.881	1,200	1,256	5.259	150	113	0.296	70
TFLite	216	0.663	50	865	2.876	120	479	1.805	120	1,959	8.365	630	1,547	5.047	80	124	0.204	30
MACE	87	0.459	90	574	2.958	100	196	1.023	150	432	2.169	400	843	4.276	100	45	0.237	20
Quant	50	0.194	10	216	0.963	30	122	0.570	30	446	2.202	160	720	3.146	20	24	0.102	10

Analyzing Influence Factors: GPU half-precision



Speedup of FP16 over FP32.

Analyzing Influence Factors: AI Accelerators

- AI accelerators can achieve significant performance improvement for “big” CNNs
  - NPU is 8.08× and 3.91× faster than the best CPU and GPU for Inception-V3
  - Only 0.79x to 1.81x faster for SqueezeNet-V11

Table 6. DNN Performance on Jetson TX2 and NPU

Platform	DNN Model	Inception-V1	Inception-V3	Resnet-50	MobileNet-V1	SqueezeNet-V11
NPU	Computation Latency (ms)	35	71	53	33	13
	Speedup to best mobile CPU	2.49×	8.08×	3.70×	1.76×	0.79×
	Speedup to best mobile GPU	2.27×	3.91×	3.25×	0.96×	1.81×
Jetson TX2	Computation Latency (ms)	17	91	59	15	12
	Speedup to best mobile CPU	5.12×	6.3×	3.70×	3.83×	0.86×
	Speedup to best mobile GPU	4.67×	3.05×	3.32×	2.11×	1.96×

Conclusion

- Present a DNN tuning framework, DNNTune
- Use it to find optimal partition point for mobile-cloud computing
- And analyze the behaviors of three typical kinds of DNNs on diverse mobile platforms with different influence factors

Thank you.