The University of Texas at Austin
Electrical and Computer Engineering

Carnegie Mellon University
Electrical & Computer Engineering

Putting the “Machine” Back in Machine Learning: The Case for Hardware-ML Model Co-design

Diana Marculescu
The University of Texas at Austin and Carnegie Mellon University
dianam@{utexas.edu, cmu.edu}
enyac.org

Hey Siri...

What's 100 divided by 2?

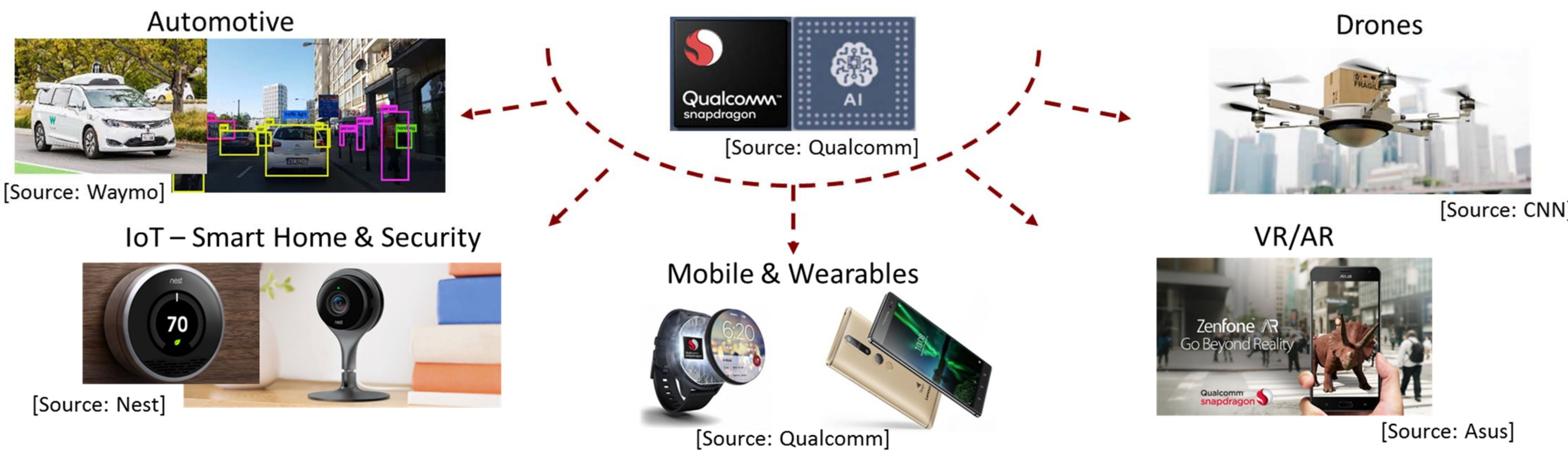
What's my name?

What is Apple?

Off-network

Machine Learning Applications Push Hardware to its Limits

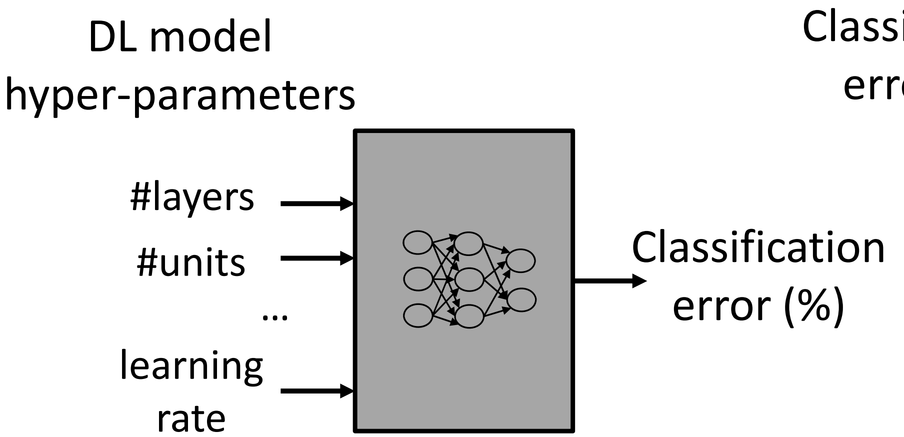
- **Deep Learning (DL) models are now used in every modern computing system**

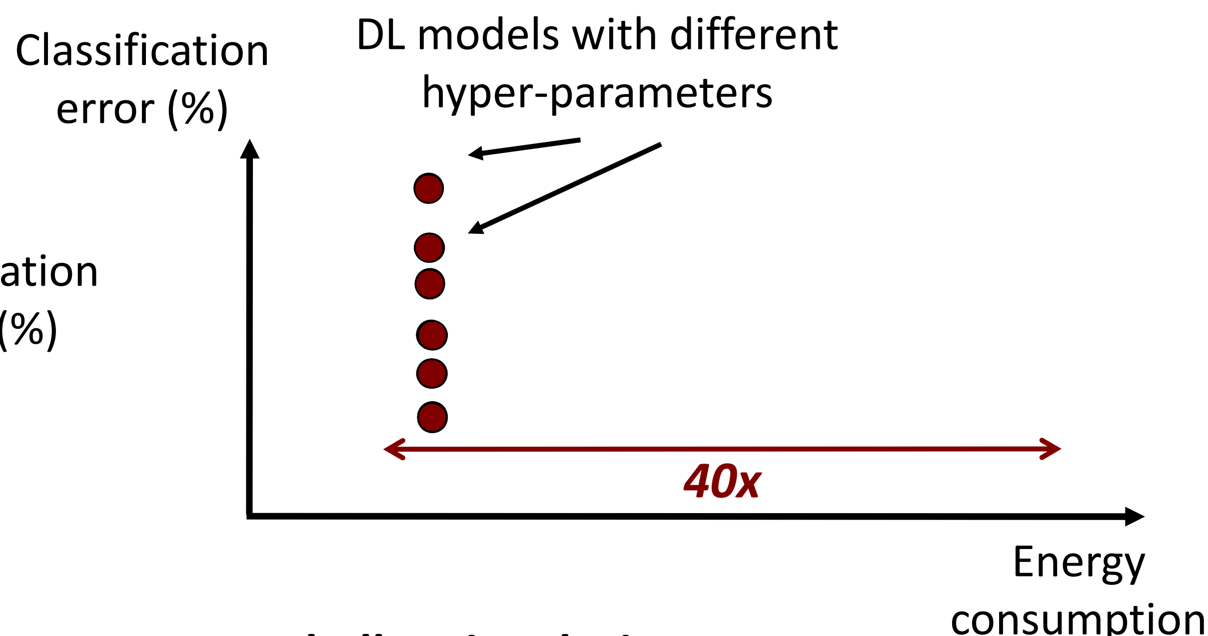


- **Hardware constraints are a key limiting factor for DL on mobile platforms**
  - **Energy** constraints: object detection drains smartphone battery in 1 hour! [Yang *et al., CVPR'*17]
  - Edge-cloud **communication** constraints
  - On-device **inference** (**response**) time constraints

Challenge: Designing DL Models under Hardware Constraints is Hard

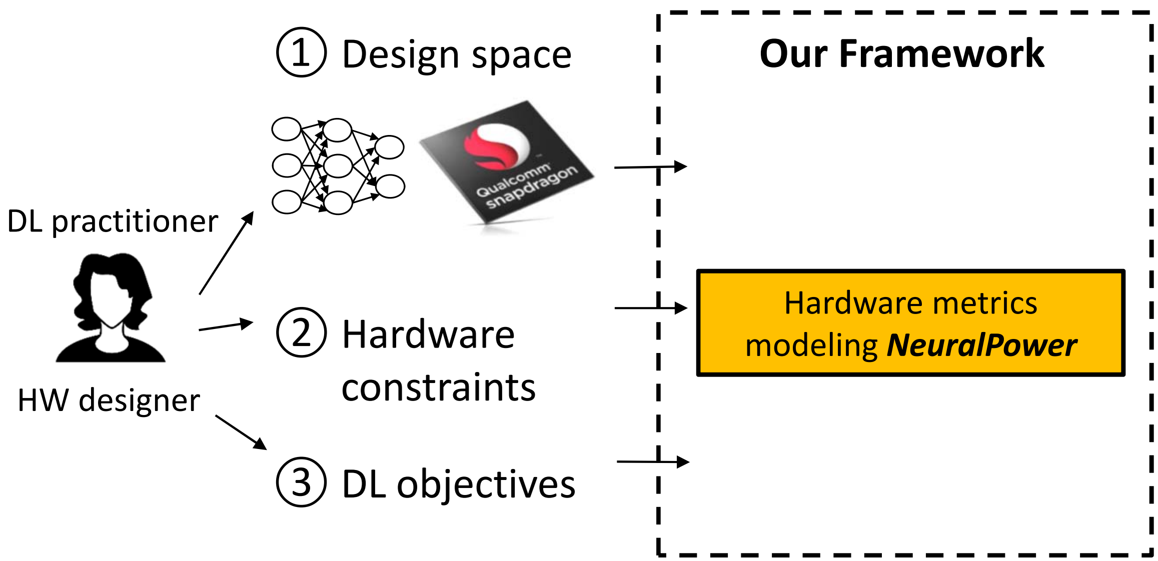
- **Hyper-parameter optimization: Find DL model with optimal learning performance**

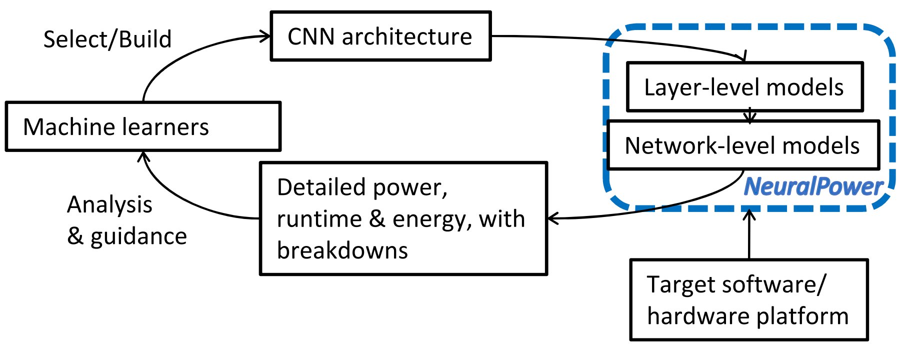




- **Hardware constraints lead to an ever more challenging design space**
  - 12k models, 800 GPUs, 28 days ≈ 62 GPU-years! [Zoph *et al.*, *arXiv:1707.07012*, 2017]

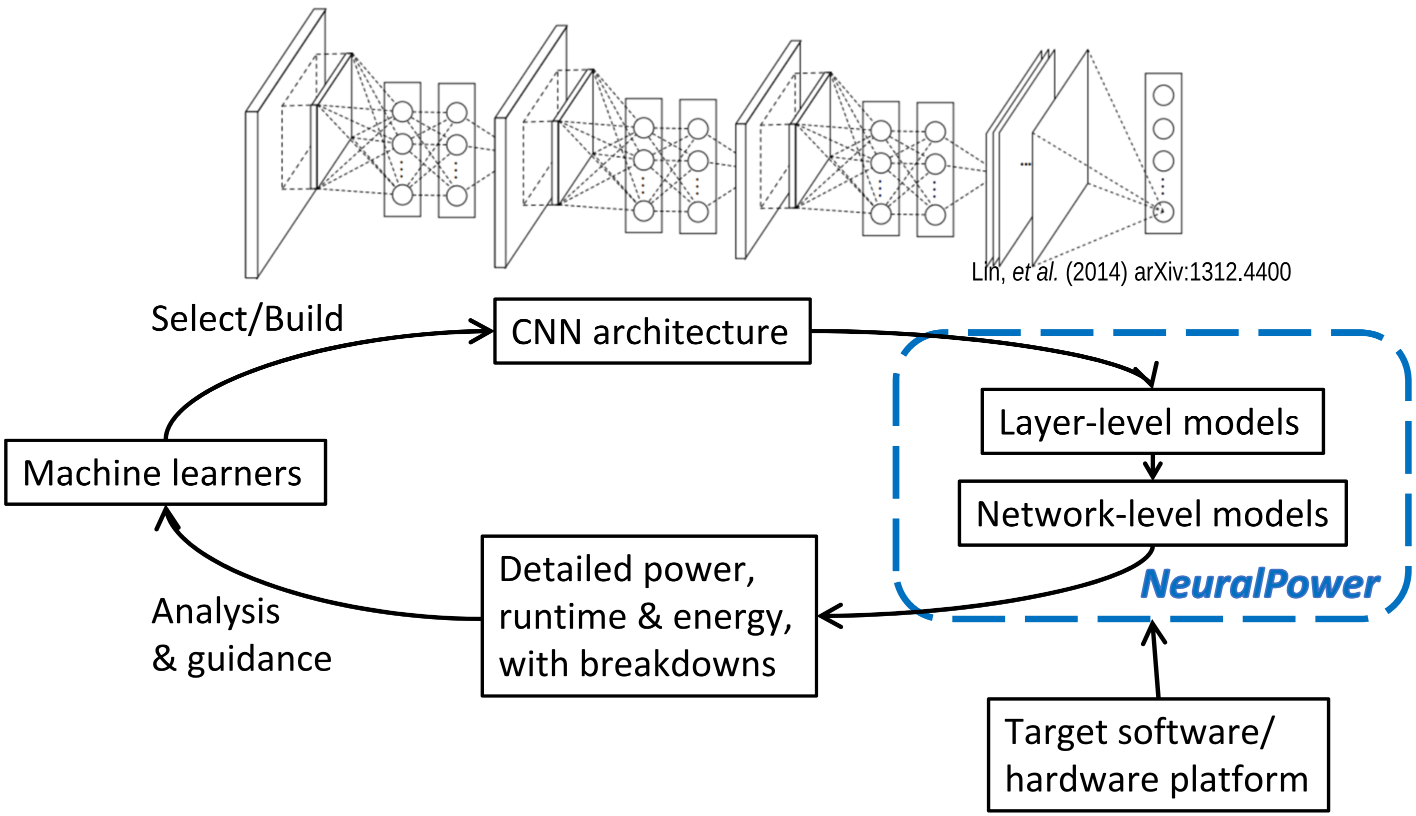
We Can’t Optimize What We Can’t Measure: DL-HW Models





- **90% accurate** models for **power, energy, and latency** for DL running on HW platforms; can be used as an objective or constraint

NeuralPower: A Layer-wise Predictive Framework

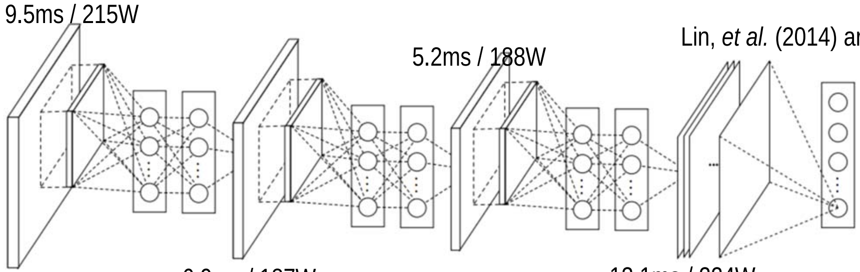


[E. Cai, D. Stamoulis, D.-C. Juan, D. Marculescu, *ACML*'17]

NeuralPower: Network-Level Models

- **Energy:**

$$\hat{E}_{total} = \hat{T}_{total} \cdot \hat{P}_{avg} = \sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n$$



- **Runtime:**

$$\hat{T}_{total} = \sum_{n=1}^{N} \hat{T}_n$$

- **Power:**

$$\hat{P}_{avg} = \frac{\sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n}{\sum_{n=1}^{N} \hat{T}_n}$$

NeuralPower: Layer-Level Models

- **Runtime model:**

Degree K_T polynomial terms

$$\hat{T}(\mathbf{x}_T) = \sum_j c_j \boxed{\prod_{i=1}^{D_T} x_i^{q_{ij}}} + \sum_s c'_s \boxed{\mathcal{F}_s(\mathbf{x}_T)}$$

Additional terms

Feature space

$$\text{where } \mathbf{x}_T \in \mathbb{R}^{D_T};\ q_{ij} \in \mathbb{N};\ \forall j,\ \sum_{i=1}^{D_T} q_{ij} \leq K_T$$

e.g., Feature space for Conv. = {kernel size, stride size, padding size, #filters, …}

- **Power model:**

Degree K_P polynomial terms

$$\hat{P}(\mathbf{x}_P) = \sum_j z_j \boxed{\prod_{i=1}^{D_P} x_i^{m_{ij}}} + \sum_k z'_k \boxed{\mathcal{F}_k(\mathbf{x}_P)}$$

Additional terms

Feature space

$$\text{where } \mathbf{x}_P \in \mathbb{R}^{D_P};\ m_{ij} \in \mathbb{N};\ \forall j,\ \sum_{i=1}^{D_P} m_{ij} \leq K_P$$

e.g., Feature space for Conv. = {kernel size, log(kernel size), stride size, log(stride size), …}

Layer-level Results

- **Runtime:**
  - Baseline: Paleo [Qi *et al.,* ICLR'17]: uses analytical methods to calculate the response time for CNNs

Layer type	*NeuralPower*			Paleo Qi et al. (2016)	
	Model size	RMSPE	RMSE (ms)	RMSPE	RMSE (ms)
Convolutional	60	39.97%	1.019	58.29%	4.304
Fully-connected	17	41.92%	0.7474	73.76%	0.8265
Pooling	31	11.41%	0.0686	79.91%	1.763

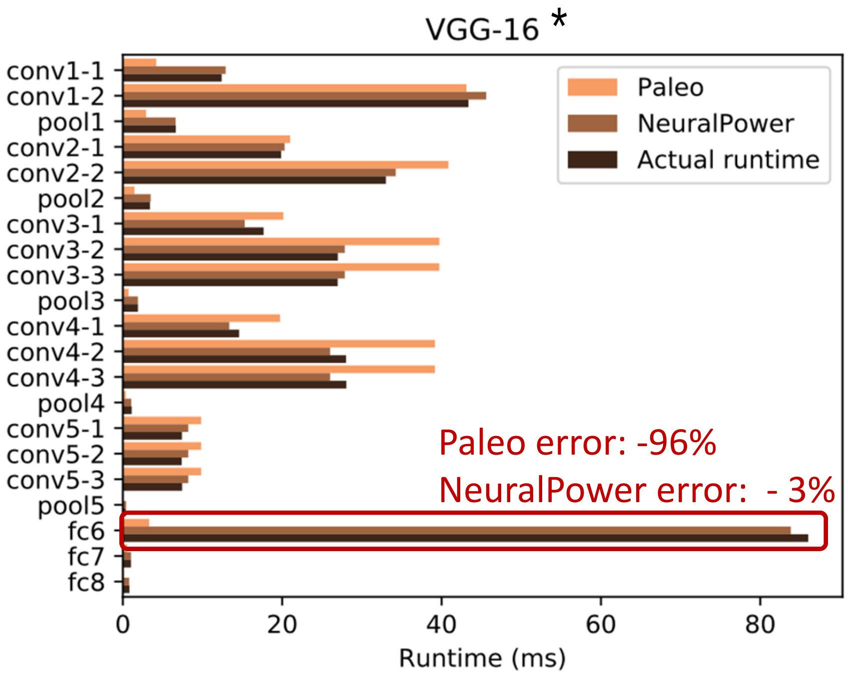
- **Power:**
  - No prior work with respect to power prediction

Layer type	*NeuralPower*		
	Model size	RMSPE	RMSE (W)
Convolutional	75	7.35%	10.9172
Fully-connected	15	9.00%	10.5868
Pooling	30	6.16%	6.8618

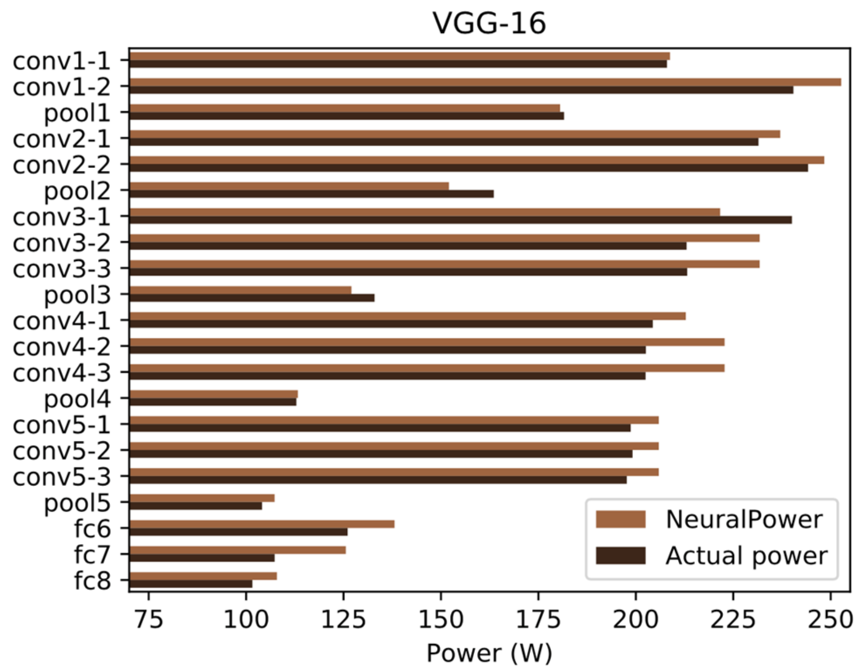
[E. Cai, D. Stamoulis, D.-C. Juan, D. Marculescu, *ACML*'17]

Network-level Results: Breakdown

Runtime



Power



* Comparison against prior art: "[*H.Qi, E.R. Sparks, and A. Talwalkar.*, ICLR'17]

[E. Cai, D. Stamoulis, D.-C. Juan, D. Marculescu, *ACML'*17]

Network-level Results: Runtime & Power

- **Runtime**

CNN name	Qi et al. (2016) Paleo (ms)	***NeuralPower*** $\hat{T}_{total}$ (ms)	Actual runtime T_{total} (ms)
VGG-16	345.83	373.82	368.42
AlexNet	33.16	43.41	39.02
NIN	45.68	62.62	50.66
Overfeat	114.71	195.21	197.99
CIFAR10-6conv	28.75	51.13	50.09

- **Power**

$$\hat{P}_{avg} = \frac{\sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n}{\sum_{n=1}^{N} \hat{T}_n}$$

CNN name	***NeuralPower*** $\hat{P}_{total}$ (W)	Actual power P_{avg} (W)
VGG-16	206.88	204.80
AlexNet	174.25	194.62
NIN	179.98	226.34
Overfeat	172.20	172.30
CIFAR10-6conv	165.33	188.34

[E. Cai, D. Stamoulis, D.-C. Juan, D. Marculescu, *ACML*'17]

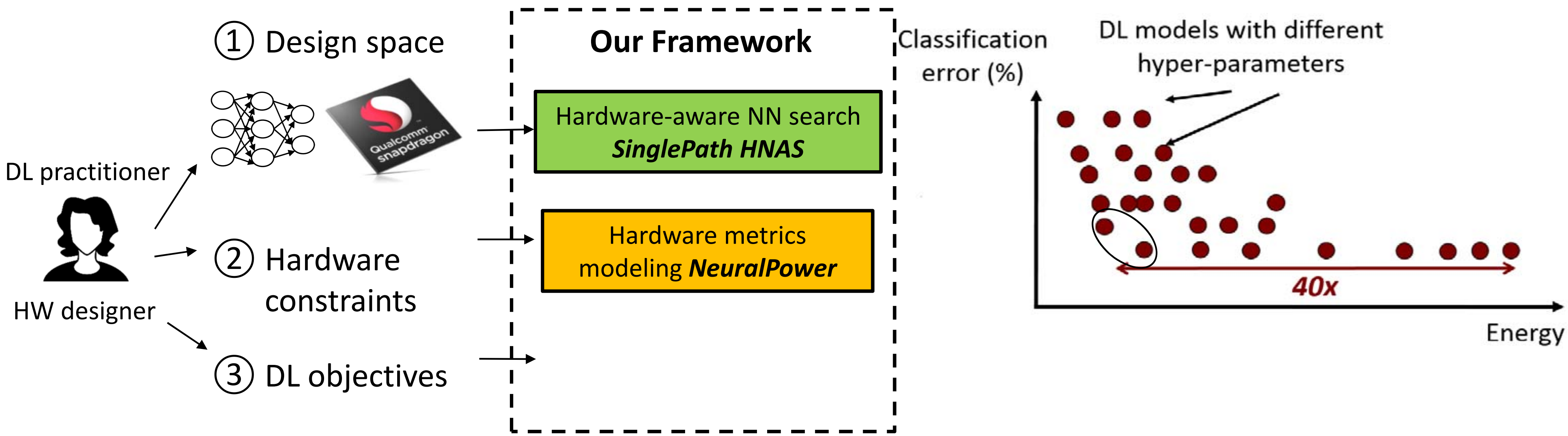
Network-level Results: Energy

- **Energy** $$\hat{E}_{total} = \hat{T}_{total} \cdot \hat{P}_{avg} = \sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n$$

CNN name	***NeuralPower*** $\hat{E}_{total}$ (J)	Actual energy E_{total} (J)
VGG-16	77.312	75.452
AlexNet	7.565	7.594
NIN	11.269	11.465
Overfeat	33.616	34.113
CIFAR10-6conv	8.938	9.433

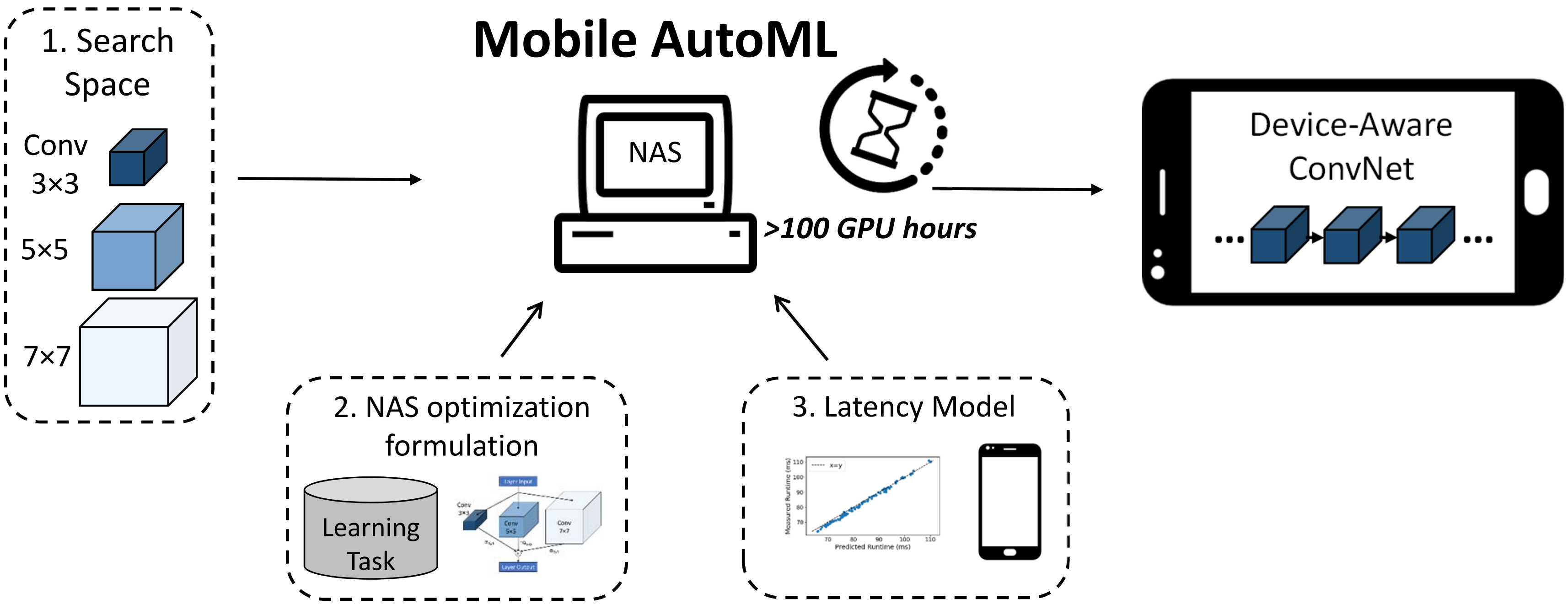
[E. Cai, D. Stamoulis, D.-C. Juan, D. Marculescu, *ACML*'17]

If We Can Measure, Can We Optimize It Efficiently?



- **Neural architecture search** can bring **5-10x** improvement in energy or latency with minimal loss in accuracy; or can satisfy **real-time constraints** for inference

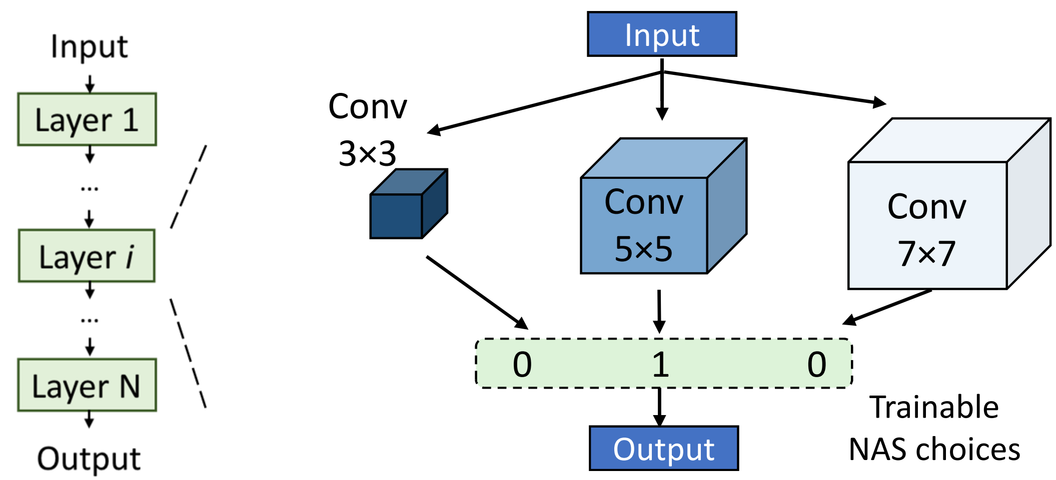
Device-aware ConvNet design: Key questions for practitioners



- *Can we **automatically** design ConvNets with **highest** image classification accuracy under smartphone **latency constraints**?*
- *Can we reduce the search cost of Neural Architecture Search (NAS) **from days down to a few hours**?*

Background: Multi-Path Differentiable NAS

Existing Multi-Path Differentiable NAS approaches [1,2,3]



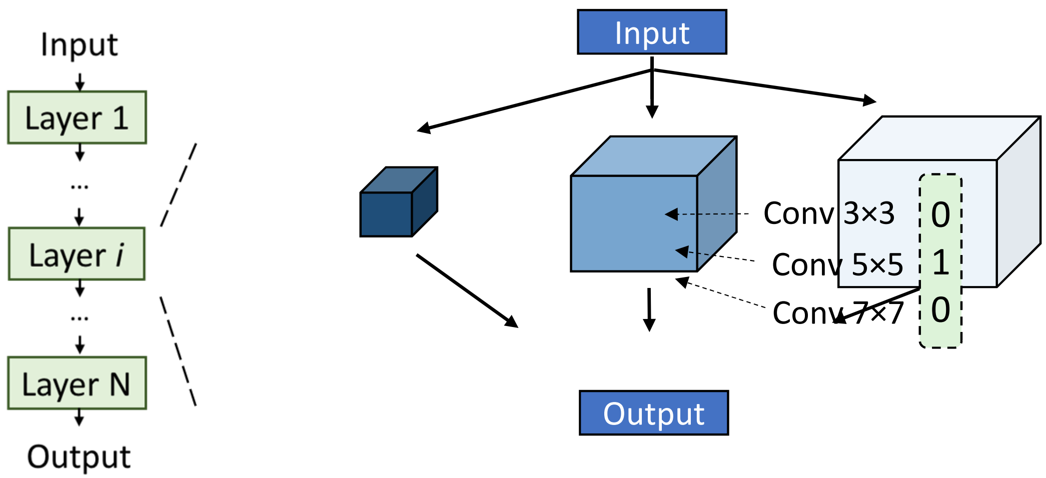
- **Supernet**: each candidate operation as a separate path per layer
- **NAS problem** viewed as an expensive path-level selection
- **Number of parameters** per layer: all weights across all paths

- Multi-path Differentiable NAS interchangeably updates NAS choices and model weights
- The combinatorially large design space leads to high search cost time (>100 GPU-hours)

[1] Cai *et al.* ProxylessNAS, ICLR'19, [2] Wu *et al.* FBNet, CVPR'19, [3] Liu *et al.* DARTS, ICLR'18

Proposed *Single-Path NAS*: Key contributions

Proposed methodology: incorporate all candidate ops over one single-path

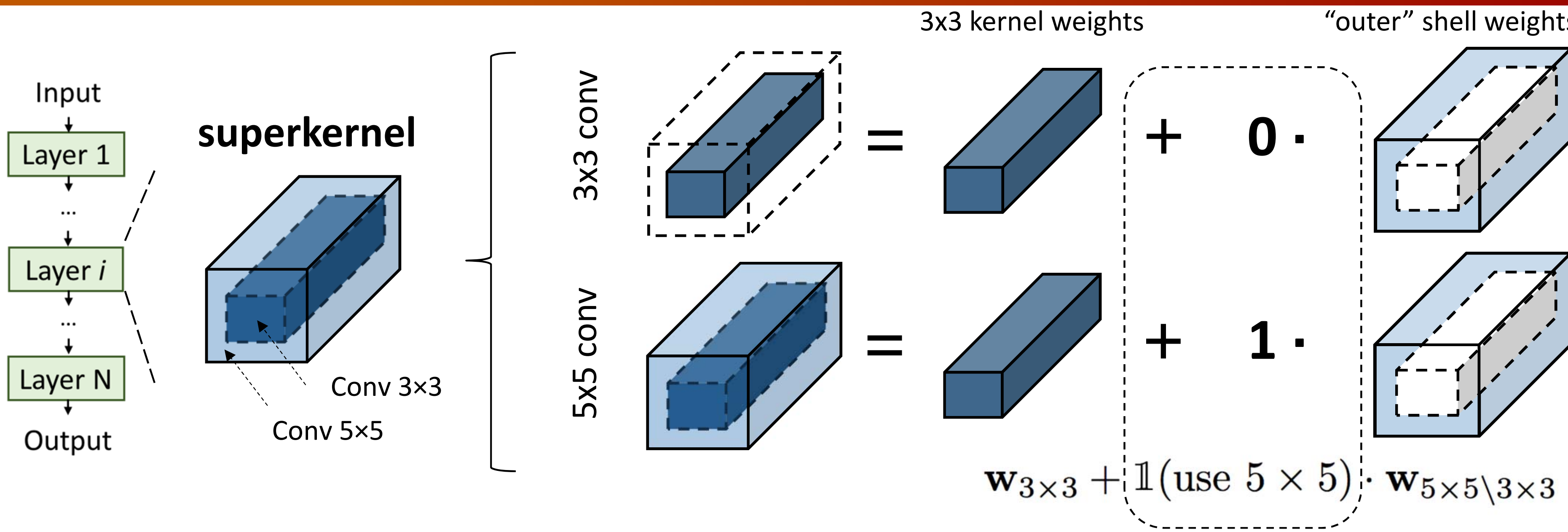


- **Supernet**: all candidate operations in a *single* ***superkernel*** per layer
- **NAS problem** viewed as an *efficient* kernel-level selection
- **Number of parameters** per layer: weights of largest candidate op *only*

- **Novel differentiable "encoding"** of NAS design choices over ***single-path*** **design space**
- **State-of-the-art AutoML:** up to **5,000× reduced** search cost, ImageNet top1 75.62%

[D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, D. Marculescu, *ECML-PKDD*'19]

Making kernel architectural decisions differentiable



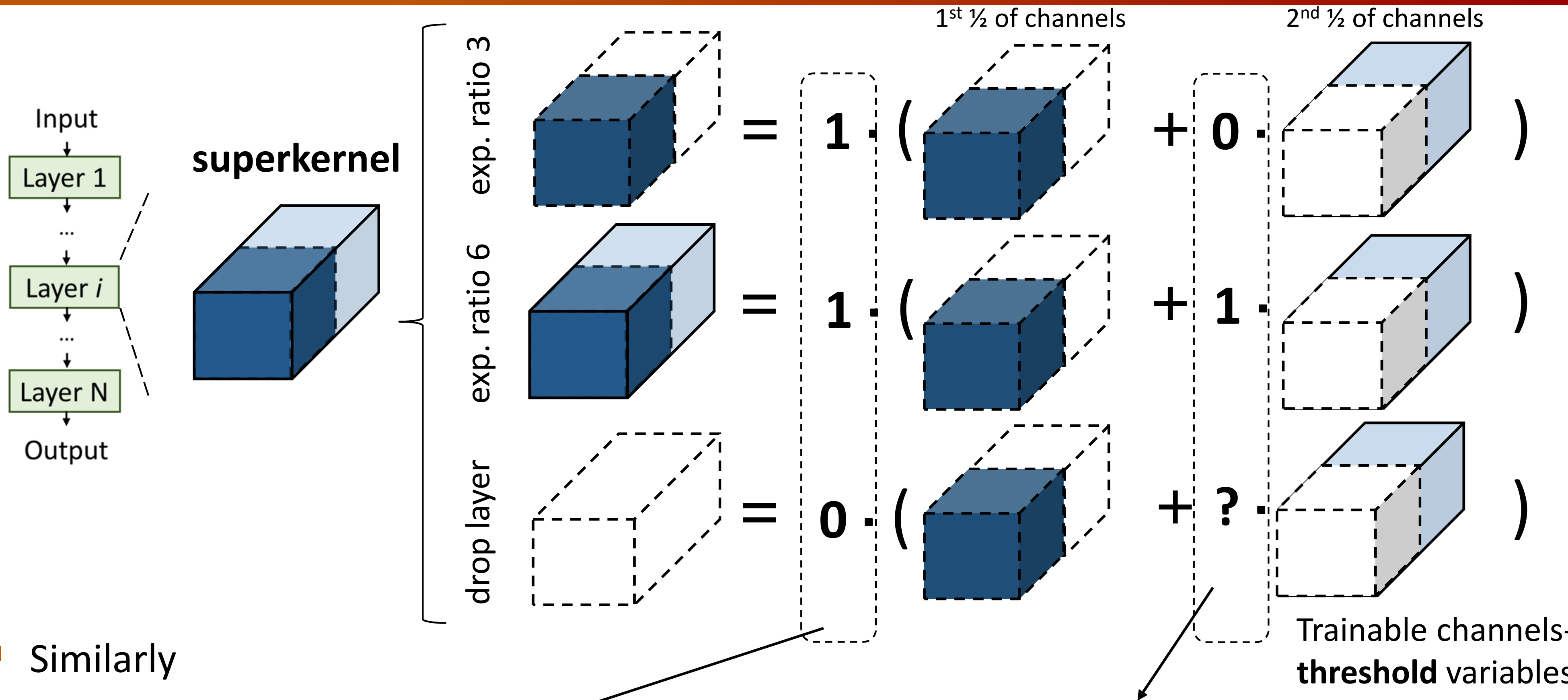
- NAS kernel choice is formulated via a differentiable decision function [1,2]

$$\mathbf{w}_k = \mathbf{w}_{3\times3} + \sigma(\underbrace{\left\|\mathbf{w}_{5\times5\backslash3\times3}\right\|^2}_{\text{Group lasso}} > t_k) \cdot \mathbf{w}_{5\times5\backslash3\times3}$$

Trainable kernel-**threshold** variable

[1] Ding *et al.* FlightNNs, DAC'19
[2] Choi *et al.*, PACT, 2018

Making channel architectural decisions differentiable



- Similarly

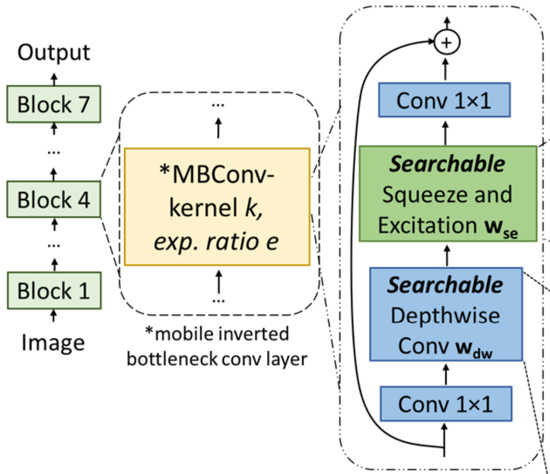
$$\mathbf{w} = \sigma\left(\left\|\mathbf{w}_{k,3}\right\|^2 > t_{e=3}\right) \cdot \left(\mathbf{w}_{k,3} + \sigma\left(\left\|\mathbf{w}_{k,6\backslash 3}\right\|^2 > t_{e=6}\right) \cdot \mathbf{w}_{k,6\backslash 3}\right)$$

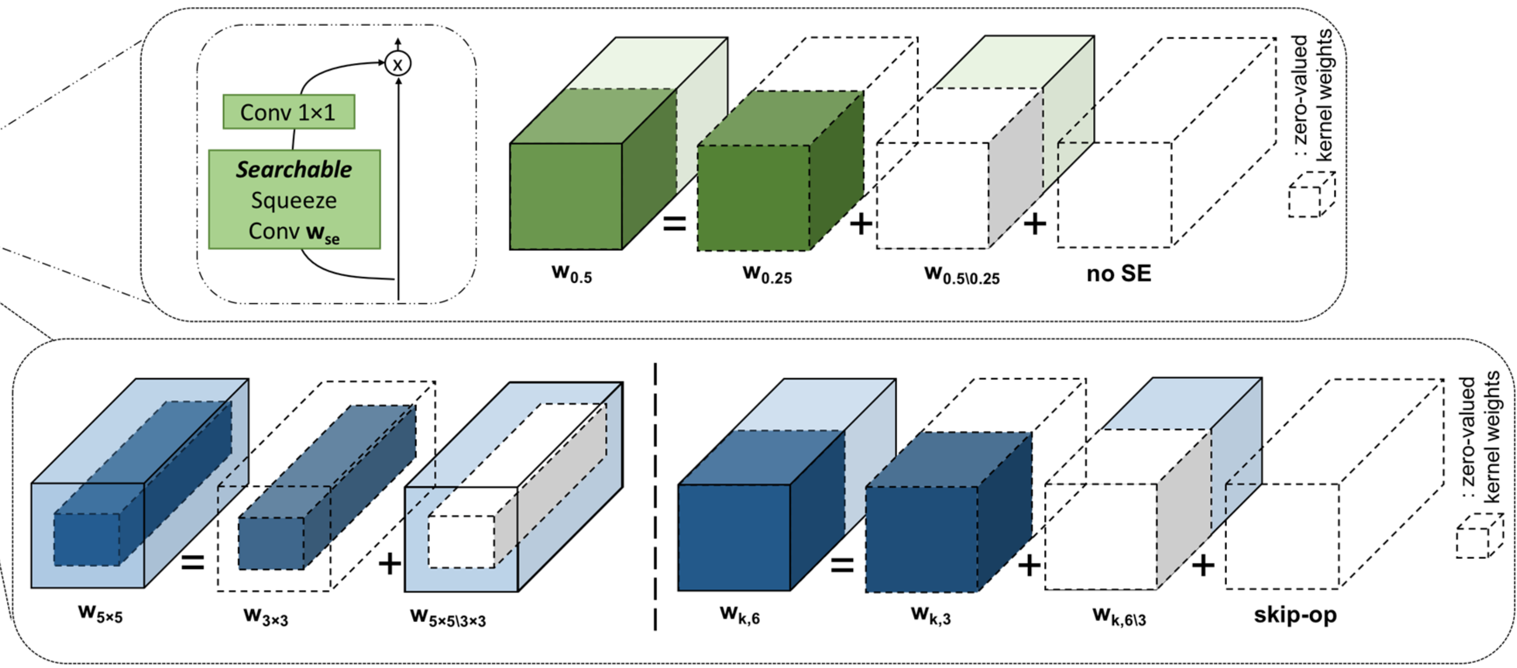
Single-Path NAS: as costly as training a compact model

- Flexibly extendable to various NAS choices
- MobileNet space: [Tan *et al.*,'19] model as large as largest candidate op

NAS Objective:

$$\min_{\mathbf{w}\, \mathbf{t}_k, \mathbf{t}_e} \mathcal{L}(\mathbf{w}|\mathbf{t}_k, \mathbf{t}_e) = \underbrace{CE}_{\text{Cross-entropy loss}} + \underbrace{\lambda}_{\text{Trade-off parameter}} \cdot \underbrace{R}_{\text{Mobile runtime loss term}}$$





Search space per MBConv:

- Kernel size ***k***: 3×3, 5×5
- Expansion ratio ***e***: 3,6
- SE ratio ***se***: 0, 0.25, 0.5
- Skip ops: skip layer or not

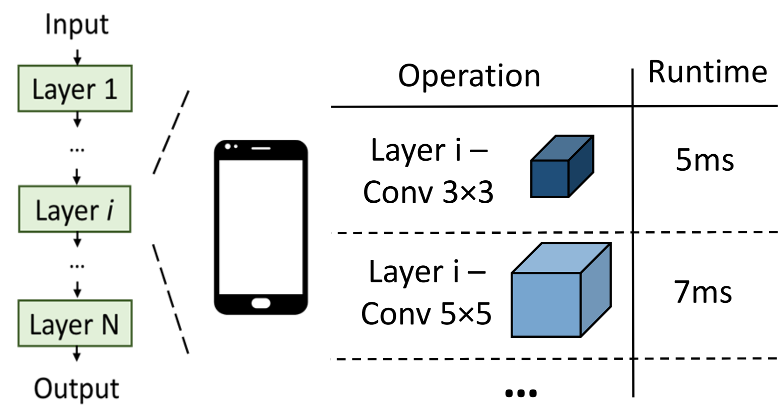
Hardware-Aware NAS: Making Runtime Term Differentiable

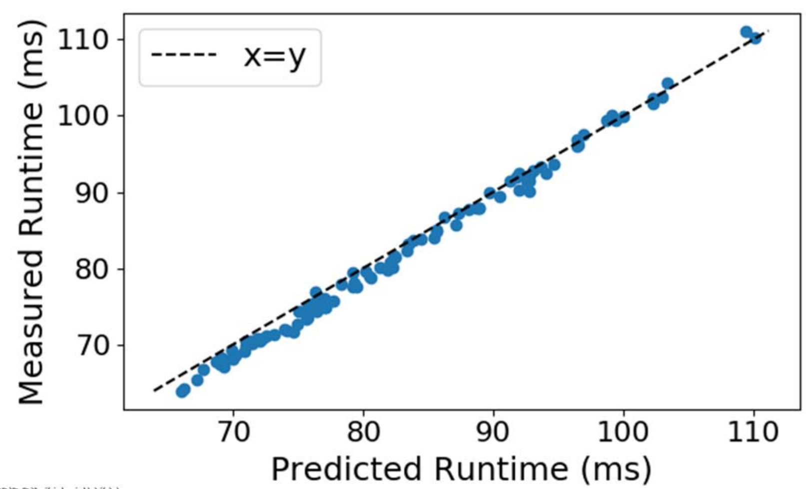
- Total ConvNet runtime is the sum of per-layer runtimes [1,2]

- We profile on ***Pixel 1 phone***
- Populate Look-up-Table model per layer *i*

- Express per-layer runtime as a function of the *Single-Path NAS* architectural choices

$$R_e^i = R^i_{3\times 3} + \sigma(\text{use } 5\times 5)\cdot(R^i_{5\times 5} - R^i_{3\times 3})$$





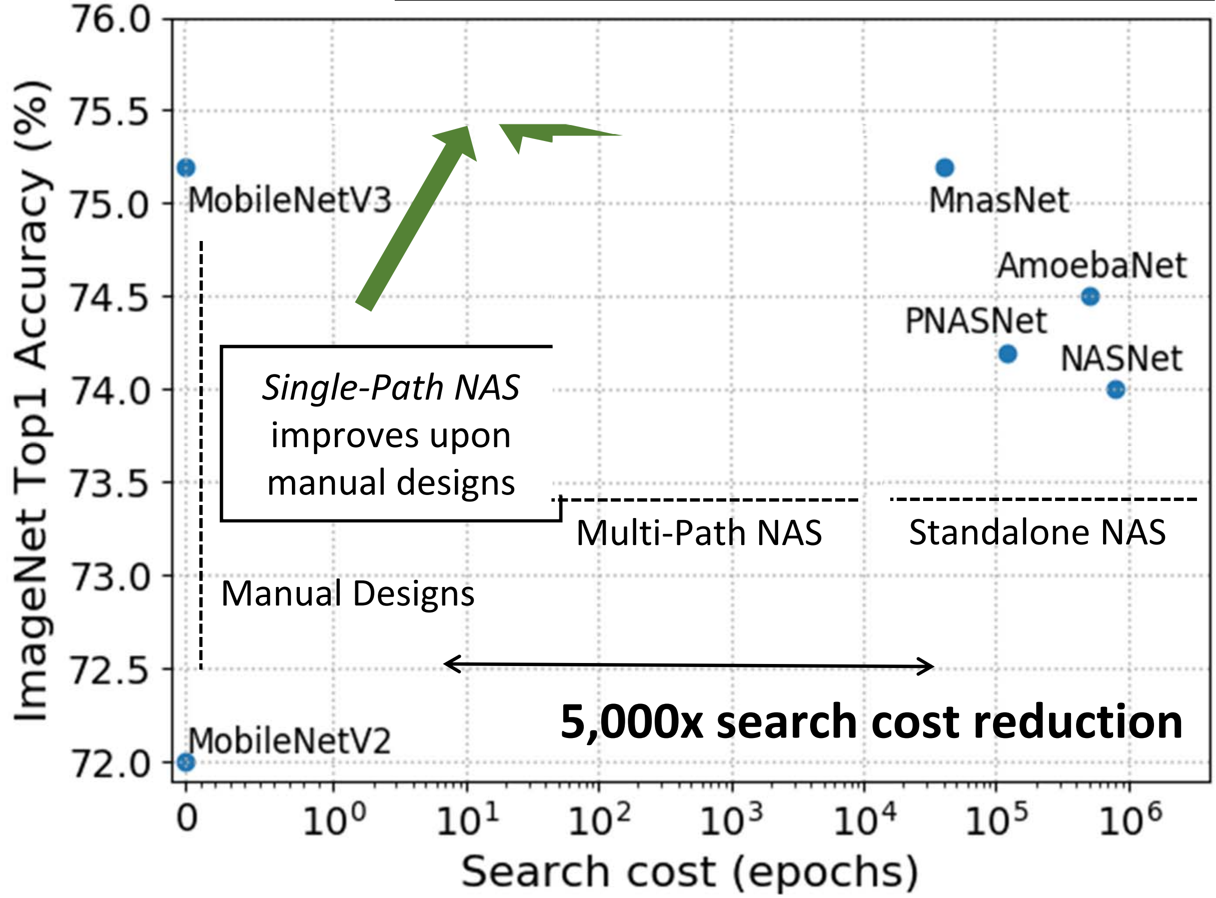
[1] Cai *et al.* ProxylessNAS, ICLR'19, [2] Wu *et al.* FBNet, CVPR'19

Single-Path NAS achieves state-of-the-art AutoML results

- *Single-Path* ConvNet: ***75.62% top-1 ImageNet*** accuracy (~80ms runtime)

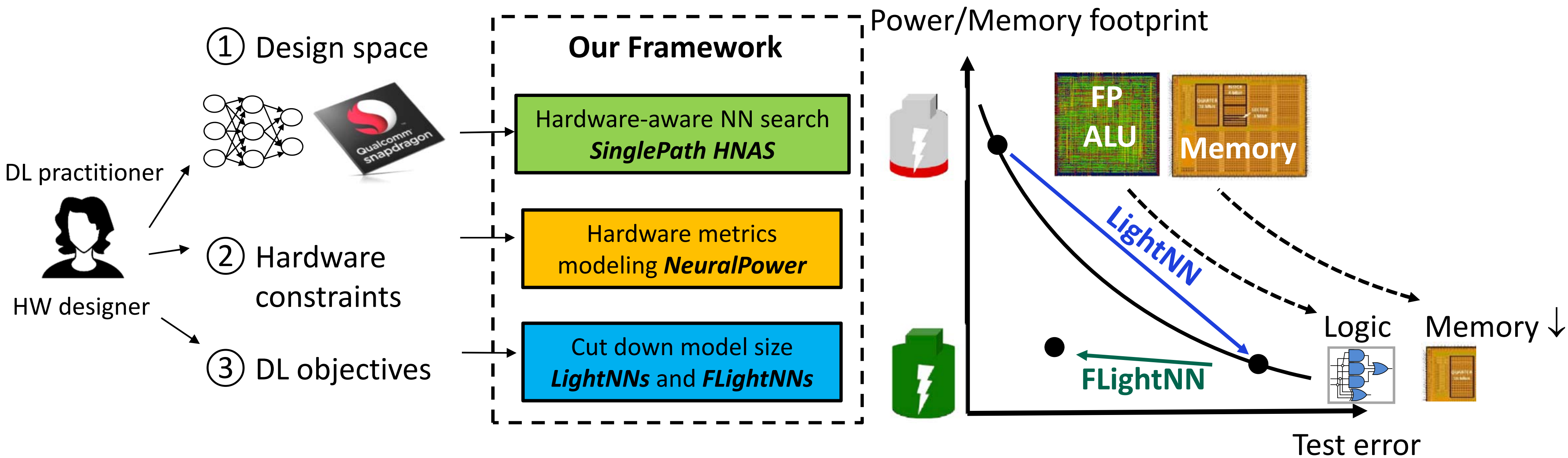
- *Single-Path NAS*: the reduced NAS search cost ***by up to 5,000 ×***

Single-Path NAS pushes state-of-the-art w.r.t *both* accuracy & NAS search cost



[1] Tan *et al.* MnasNet, CVPR'19
[2] Wu *et al.* FBNet, CVPR'19
[3] Cai *et al.* ProxylessNAS, ICLR'19

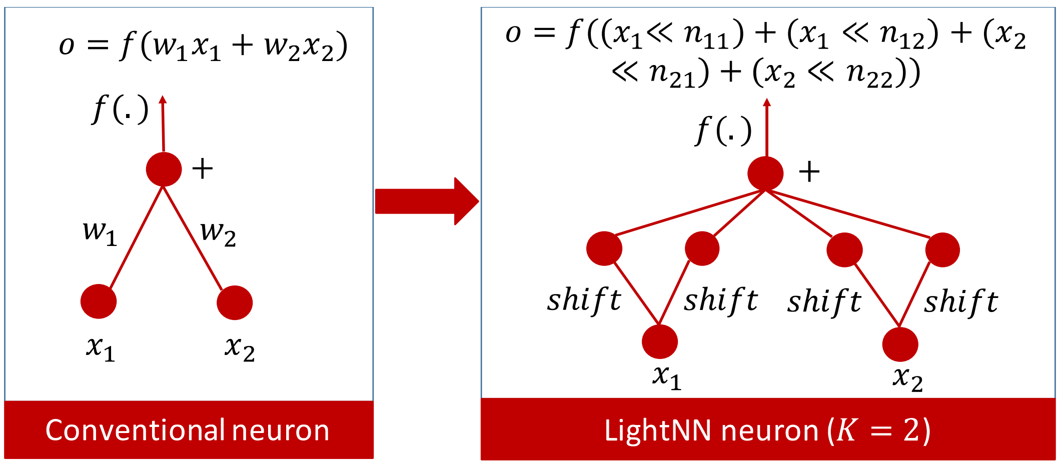
Can We Do Better?



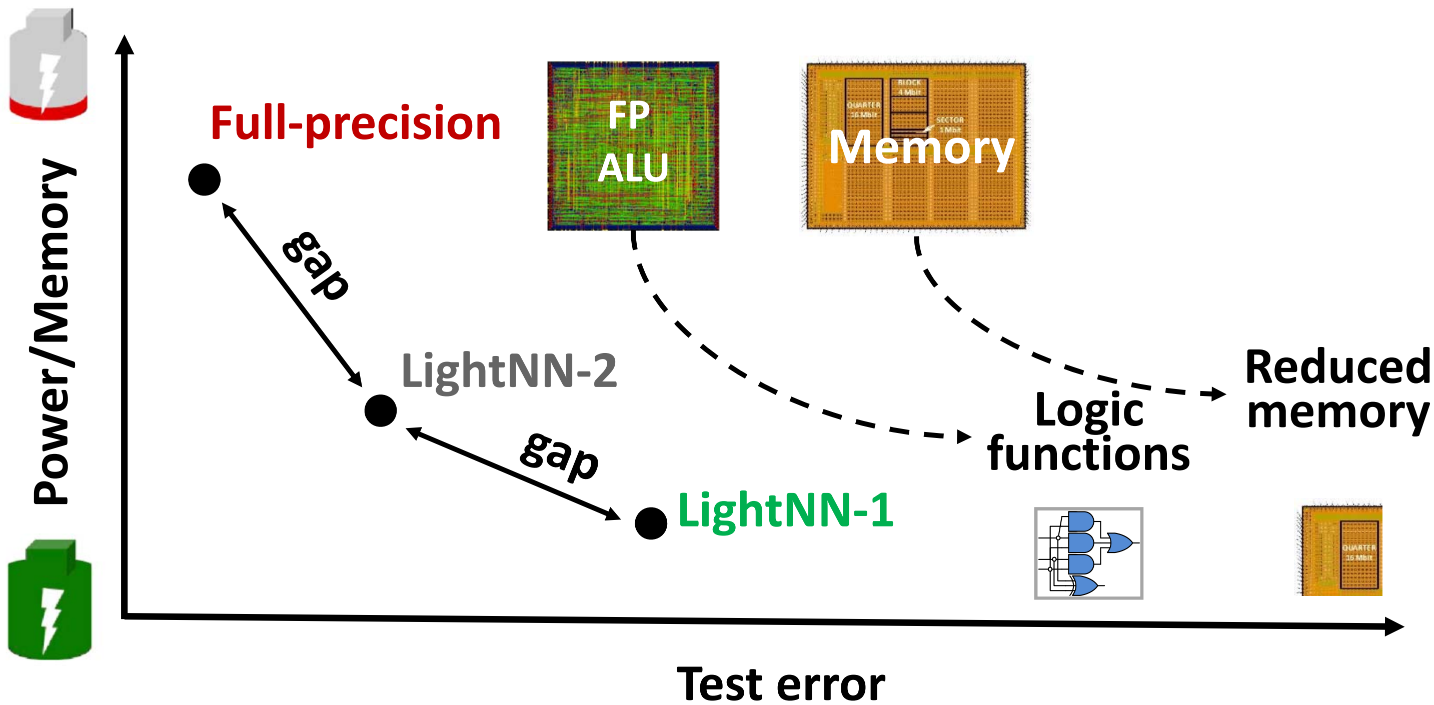
- Up to **100x lower energy**, **5x less area** with minimal loss in accuracy

LightNNs: Lightweight quantized DNN model

- **Replace multipliers with limited shift and add operators**
  - $w \cdot x = sign(w)(2^{n_1} + 2^{n_2} + \cdots + 2^{n_K}) \cdot x = sign(w)(x \ll n_1 + \cdots + x \ll n_K)$
  - We constrain K to be one or two
  - When $K = 1$, the equivalent multiplier is just a shift
  - When $K = 2$, the equivalent multiplier is two shifts and one add (shown below)

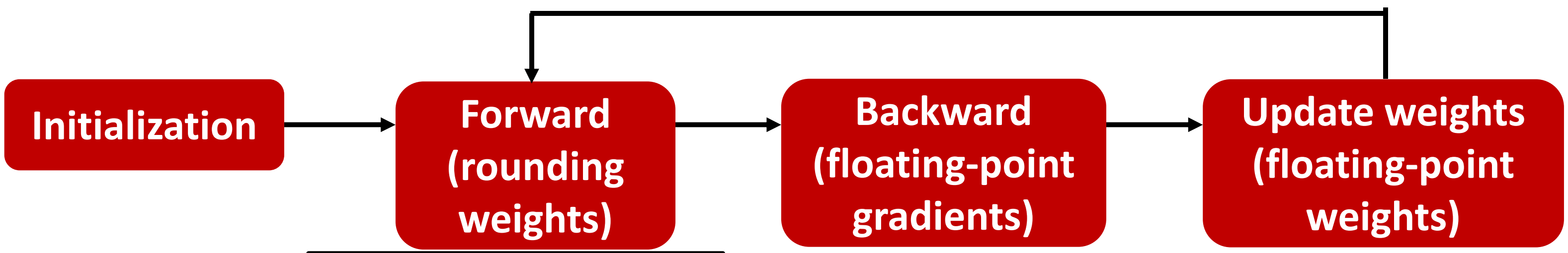


LightNNs: Lightweight quantized DNN model



Training LightNNs

- **Backpropagation algorithm is modified to improve the accuracy of trained LightNNs**



random $\boldsymbol{w_0}$

$\boxed{\boldsymbol{w_c} = round\ (\boldsymbol{w_{t-1}})}$

$\boldsymbol{w_{t-1}}$ is also saved for weights update

$$\frac{\partial l}{\partial \boldsymbol{w_c}}$$

l: loss function

$$\boldsymbol{w_t} = \boldsymbol{w_{t-1}} - \eta \frac{\partial l}{\partial \boldsymbol{w_c}}$$

η: learning rate

[R. Ding, D. Liu, S. Blanton, D. Marculescu, *GLSVLSI*'17, *ACM TRETS'19*]

Test error results

- **In most cases, from good to bad: Conventional > LightNNs > BNNs**

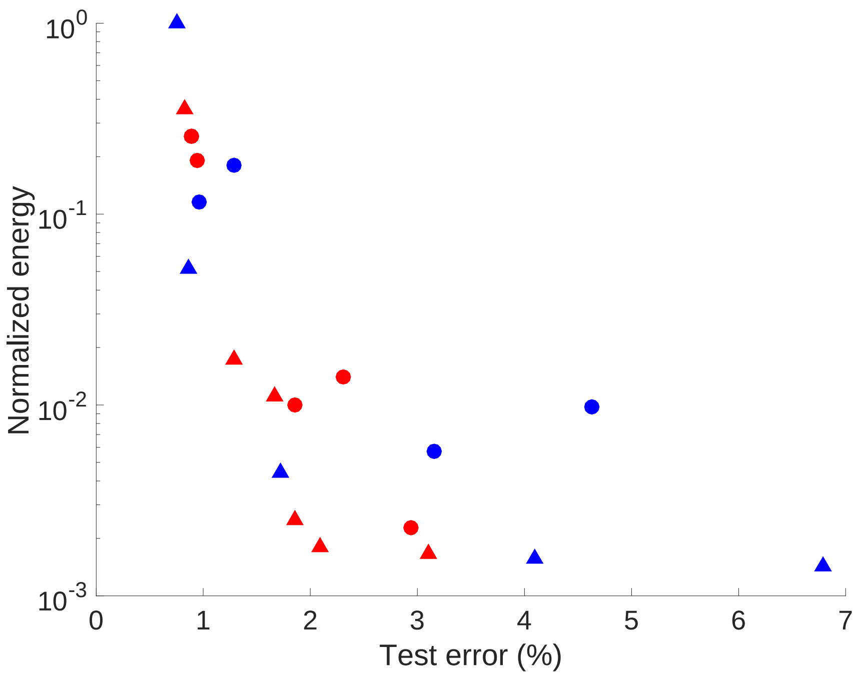
		MNIST			CIFAR-10	
		1-hidden	2-conv	3-hidden	3-conv	6-conv
Number of parameters		79,510	431,080	36,818,954	82,208	39,191,690
Test error	Conventional	1.72%	0.86%	0.75%	21.16%	10.94%
	LightNN-2	**1.86%**	**1.29%**	**0.83%**	**24.62%**	**8.84%**
	LightNN-1	**2.09%**	**2.31%**	**0.89%**	**26.11%**	**8.79%**
	BinaryConnect	4.10%	4.63%	1.29%	43.22%	9.90%
	LightNN-2-bin	**2.94%**	**1.67%**	**0.89%**	**32.58%**	**10.12%**
	LightNN-1-bin	**3.10%**	**1.86%**	**0.94%**	**36.56%**	**9.05%**
	BinaryNet	6.79%	3.16%	0.96%	73.82%	11.40%

Energy-Accuracy results

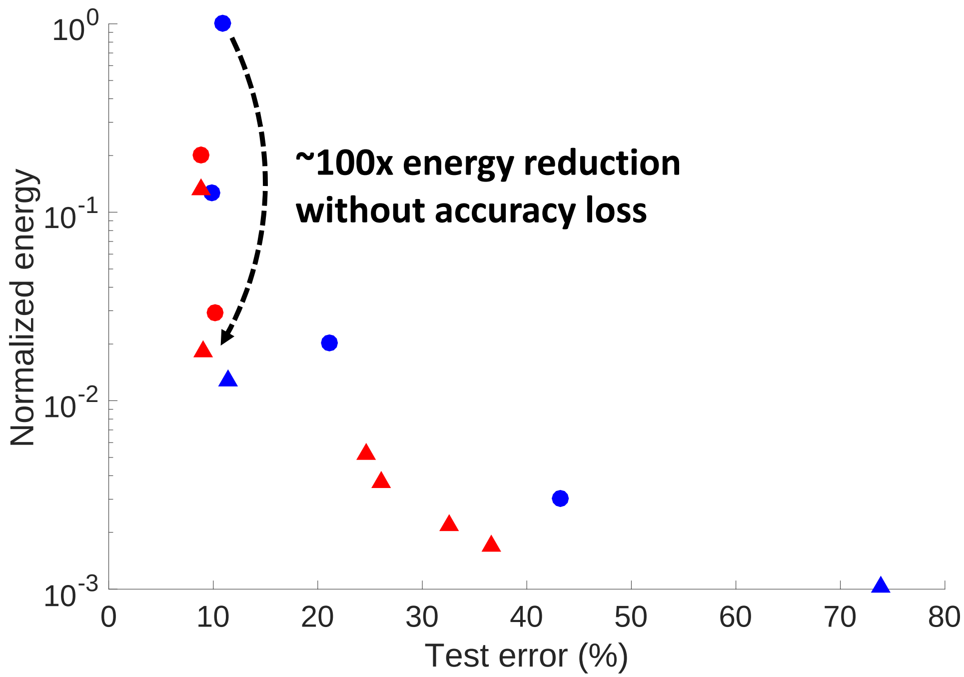
LightNNs achieve more continuous Pareto front compared to conventional DNN models

● ▲ LightNNs
● ▲ Conventional models
▲ indicates Pareto-optimal configs

MNIST

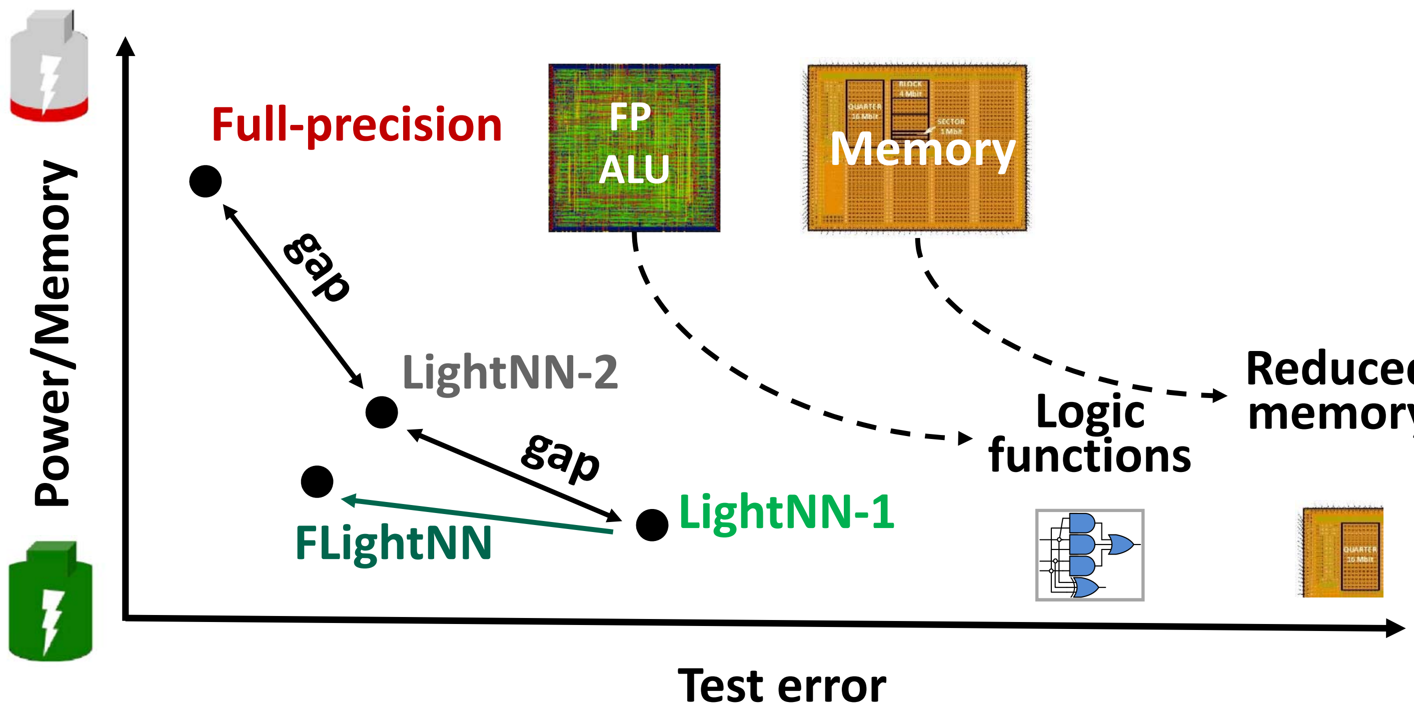


CIFAR-10



FLightNNs = Flexible LightNNs

- **With higher flexibility and improved training algorithm, FLightNNs create a better Pareto front**



Flexible-k LightNNs (FLightNNs)

- FLightNNs use customized k for each filter

LightNN-1 filters

0.5	0.25	0.25	-1
-0.5	-1	1	1
0.25	0.25	1	1
0.5	0.5	-0.5	0.25

FLightNN filters

0.5	0.25	0.25	-1
-0.5	-1	1	1
-0.25	1	0.375	1
0.375	0.375	0.625	-0.5

LightNN-2 filters

0.375	0.125	0.375	0.625
-0.5	0.625	0.125	-0.5
-0.25	1	0.375	1
0.375	0.375	0.625	-0.5

[R. Ding, D. Liu, T.-W. Chin, S. Blanton, D. Marculescu, *DAC'19*]

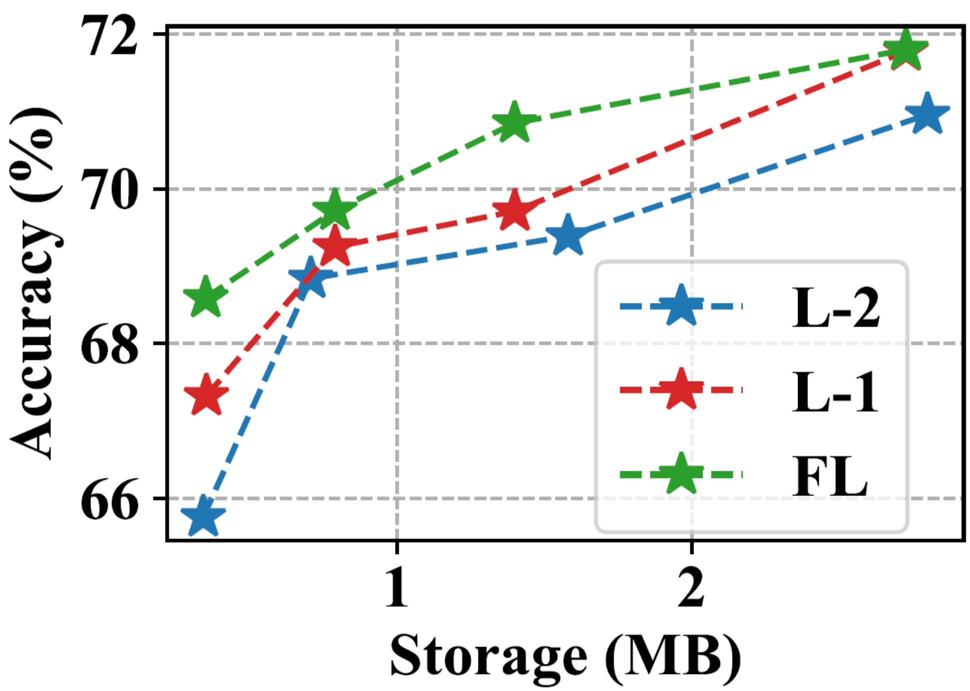
FPGA Simulation Results

- **FPGA simulation results show that FLightNNs can achieve 30x speedup compared to full-precision DNNs with negligible accuracy loss**

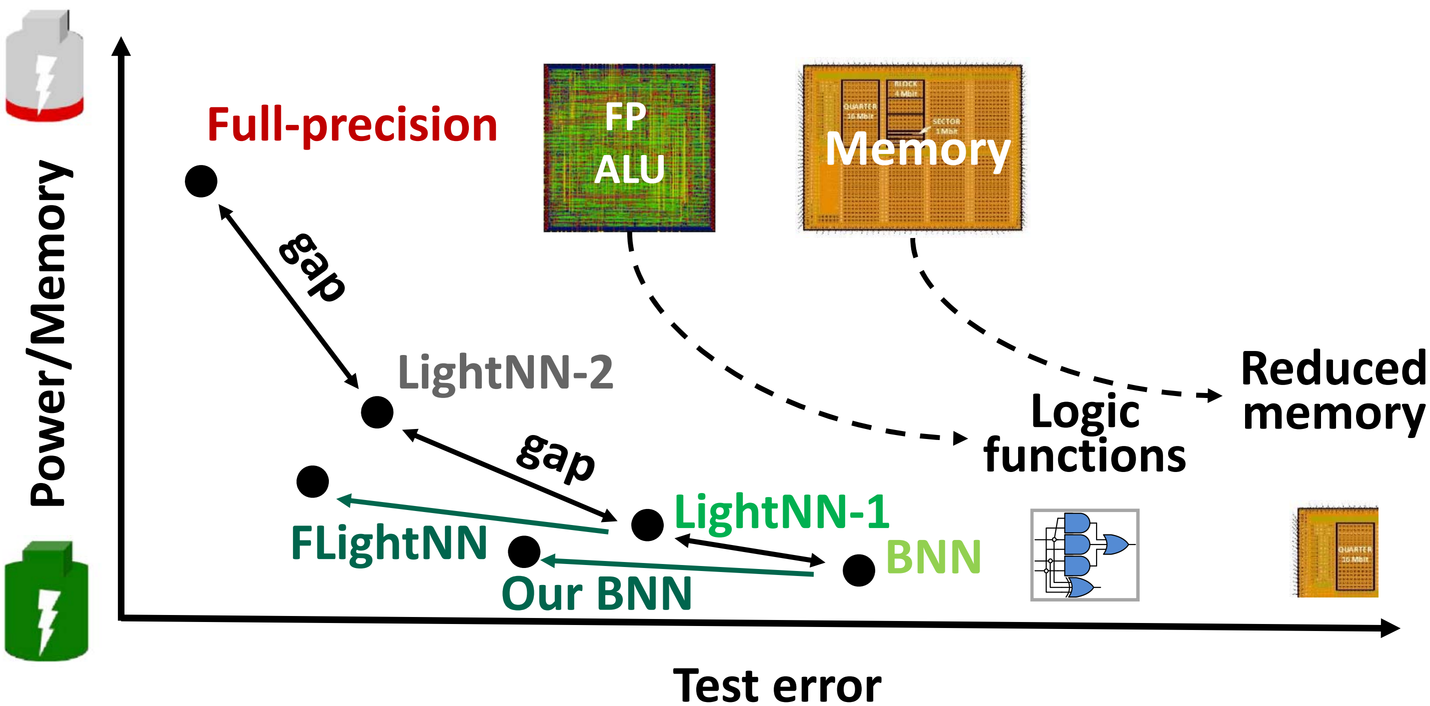
dataset	Model	Accuracy	Storage	Throughput	Speedup
CIFAR-10	Full	**92.85**	18.5	1.3	1×
	L-1	91.93	2.3	39.2	30.2×
	FP	92.23	2.3	19.8	15.2×
	FL_{3a}	**92.59**	**2.3**	**39.2**	**30.2×**
CIFAR-100	Full	**71.22**	11.2	7.4E+1	1×
	L-1	69.71	1.4	1.1E+3	15.2×
	FP	69.34	1.4	6.9E+2	9.3×
	FL_{7a}	**70.85**	**1.4**	**1.1E+3**	**30.2×**
ImageNet	L-2	**75.04**	1.8	2.7E+2	1×
	FL_{8a}	**74.80**	**1.5**	**3.1E+2**	**1.16×**

FLightNN vs. LightNNs

- **Experiment on CIFAR-100 shows that FLightNNs create a better Pareto front than LightNN-1 and LightNN-2**



Can we recover BNN accuracy loss?



Regularizing activation distribution for increased accuracy

- **Identify which of the issues is present**
  - Degeneration
  - Saturation
  - Gradient mismatch

- **Adjust regularization**
  - Shift distribution to 25-75 percentiles

- **Enable differentiability**

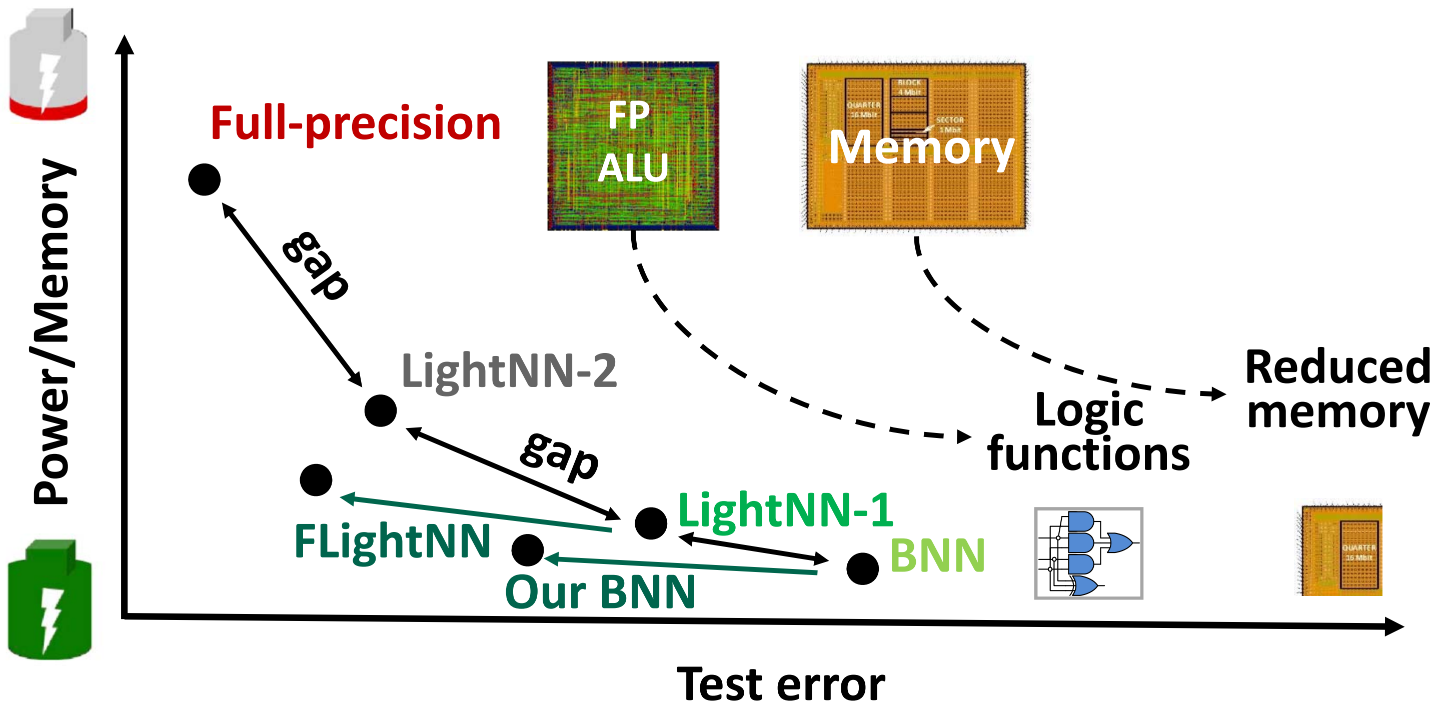
[R. Ding, T.-W. Chin, D. Liu, D. Marculescu*, CVPR'19*]

Accuracy improvement results

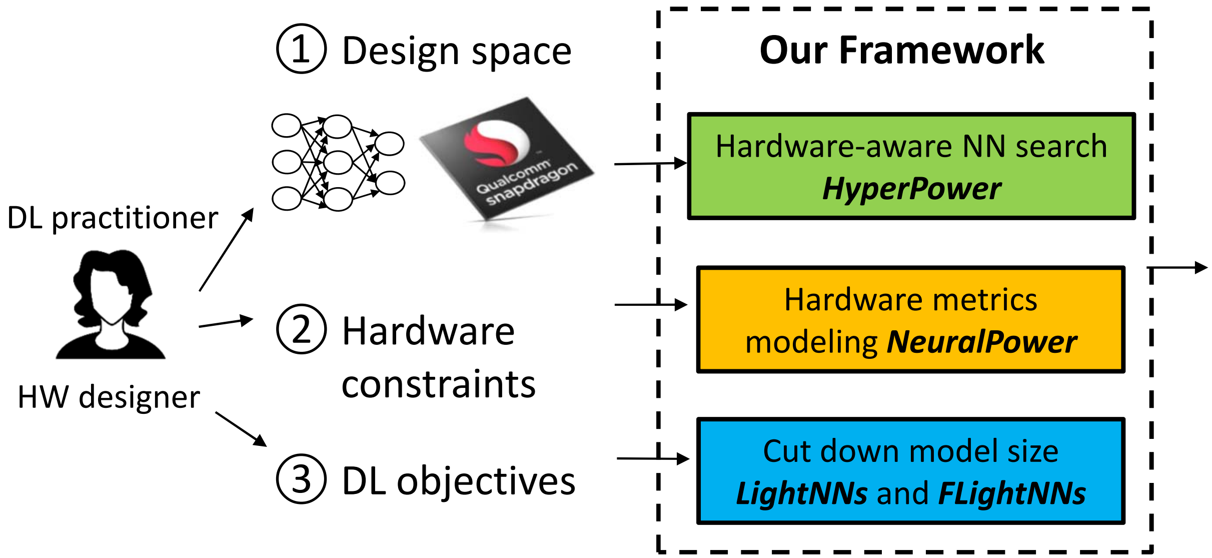
- **Our proposed regularization loss consistently improves accuracy of prior BNNs**

Model	Baseline		Ours	
	Top-1	Top-5	Top-1	Top-5
BNN [NIPS'16]	36.1%	60.1%	41.3%	65.8%
XNOR-Net [ECCV'16]	44.2%	69.2%	47.8%	71.5%
DoReFa-Net [Arxiv'16]	43.5%	-	47.8%	71.5%
Compact Net [AAAI'17]	46.6%	71.1%	47.6%	71.9%
WRPN [ICLR'18]	48.3%	-	53.8%	77.0%

FLightNNs and our improved BNNs create a better Pareto front



We Put the “Machine” Back in ML for True Co-Design



- ***Expected benefits for DL training***: Speedup hardware-constrained optimal DL model search **by 3 orders of magnitude**
- ***Expected benefits for DL deployment***: Improve DL model hardware efficiency **by 2 orders of magnitude**

Impact: This methodology can enable the optimal design of **hardware-constrained DL applications** running on **mobile/IoT platforms**

Hey Siri…

What's my name?

Off-network

The University of Texas at Austin
Electrical and Computer Engineering

Carnegie Mellon University
Electrical & Computer Engineering

Thank you!

Questions

Acknowledgements:

Collaborators: Shawn Blanton (CMU), Da-Cheng Juan (Google)
Students: Ermao Cai, Zhuo Chen, Ting-Wu (Rudy) Chin, Ruizhou Ding, Dexter Liu, Dimitrios Stamoulis

EnyAC group webpage: enyac.org

Code available: github.com/cmu-enyac and github.com/dstamoulis/single-path-nas

NSF

Qualcomm Innovation Fellowship

Google Cloud

TensorFlow RESEARCH CLOUD