nn-Meter: Towards Accurate Latency Prediction of DNN inference on Diverse Edge Devices

Li Lyna Zhang[1], Shihao Han[1,2], Jianyu Wei[1,3], Ningxin Zheng[1],
Ting Cao[1], Yuqing Yang[1], Yunxin Liu[4]

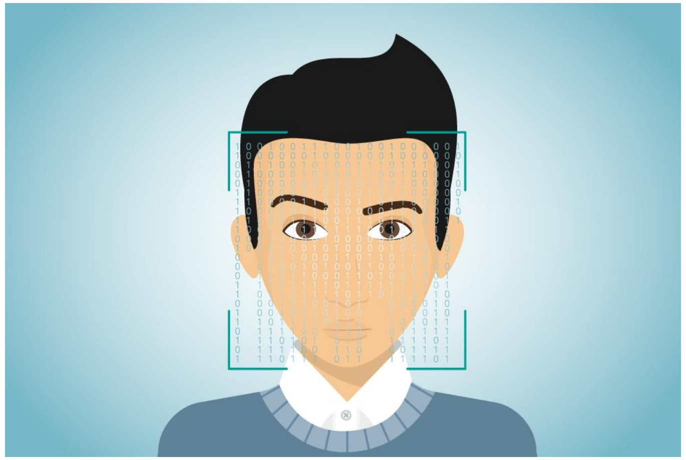
[1]Microsoft Research, [2]Rose-Hulman Institute of Technology,

[3]University of Science and Technology of China,

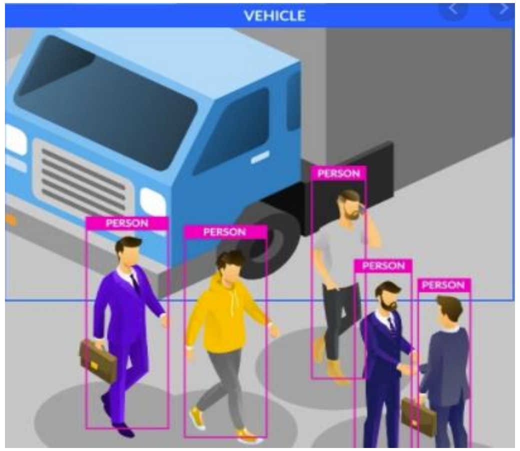
[4]Institute for AI Industry Research (AIR)Tsinghua University

https://github.com/microsoft/nn-Meter

Large demand of DNN deployment on edge devices



Face Recognition



On-device video analytics

AR/VR

Mobile apps require low-latency inference

No one-size-fits-all model on Diverse Edge Devices

It’s challenging to design one-for-all DNN to meet latency requirements on diverse edge devices:

- Various NN optimizations in inference frameworks
- Different hardware chips exhibit various computation/memory capability

MACE
Mobile AI Compute Engine
ONNX RUNTIME
OpenVINO

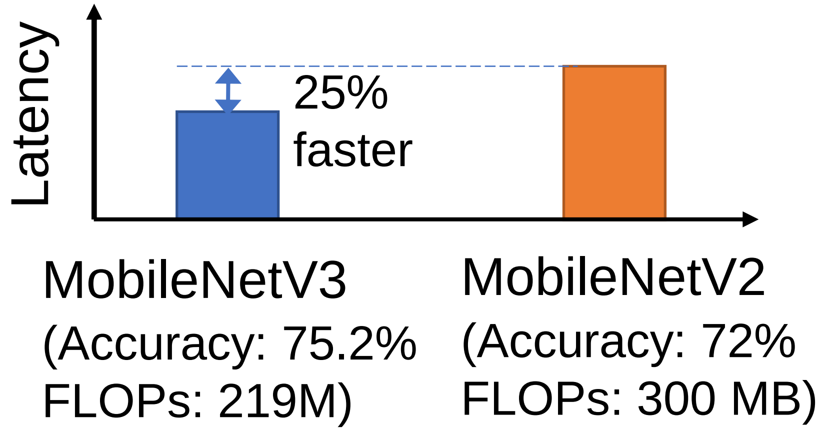
Mobile CPU | Mobile GPU | DSP | Myriad VPU | Edge TPU

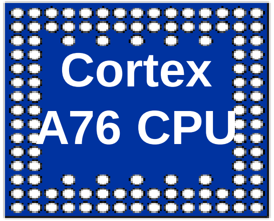
ARM | QUALCOMM | QUALCOMM | intel | Google

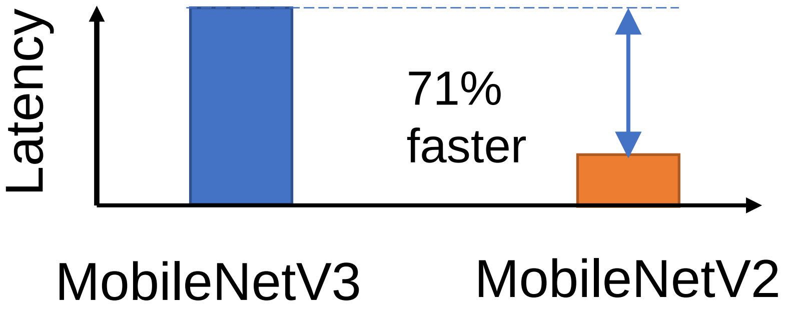
No one-size-fits-all DNN models

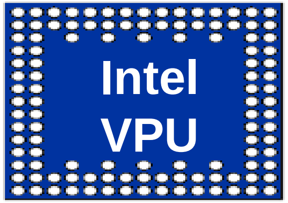
An Example

Consider inference latency in the NN design process





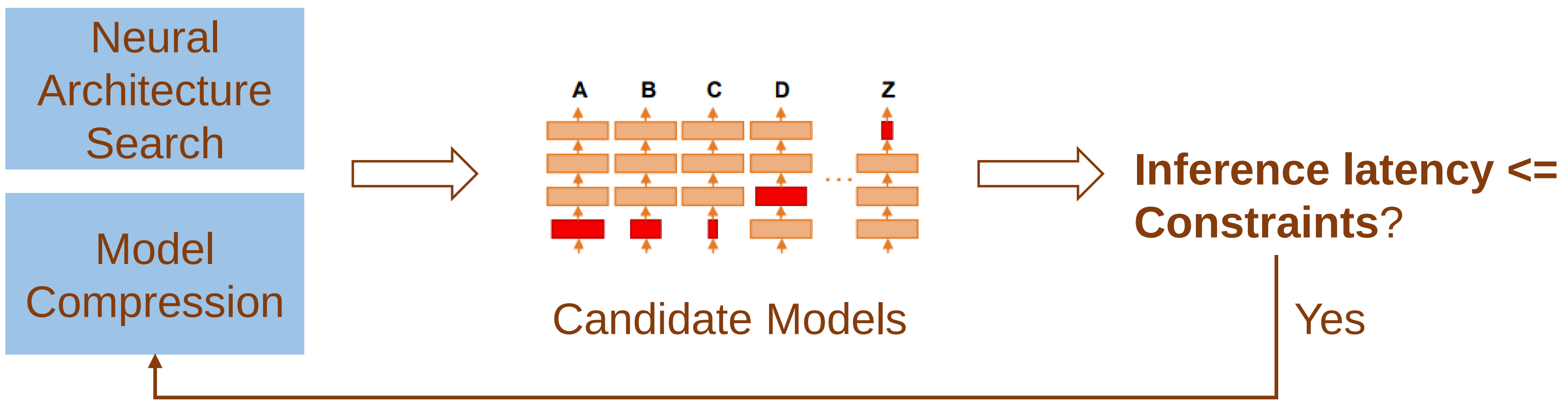




Latency: the important model design metric

To design a model that meets device latency requirements :

- Model design algorithms consider the inference latency in the design process



How to get the inference latency of DNNs on various edge devices?

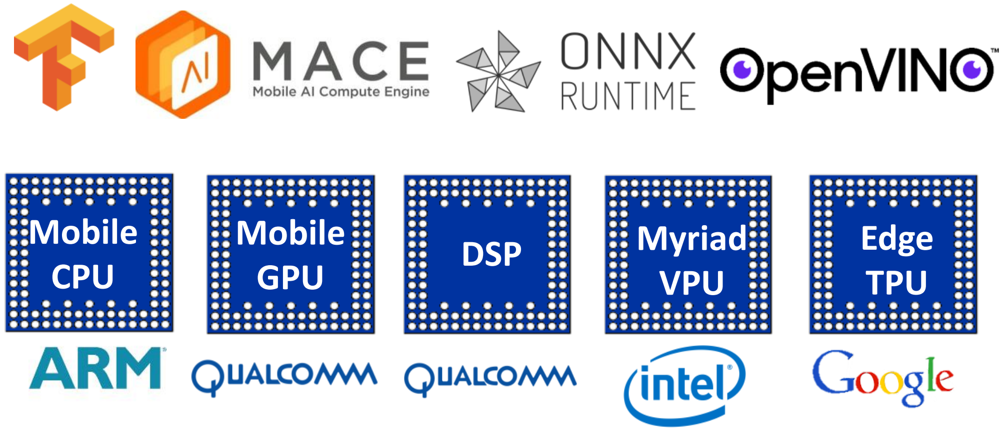
Measuring latency is expensive

Tremendous engineering efforts for model deployment

- Diverse inference frameworks
- Many chips

Time-consuming to measure a large number of models in NAS tasks

- ProxylessNAS explores ~0.3million models in one search



Diverse inference frameworks and chips

Related works: Predicting the latency

FLOPs-based method

- ***Disadvantage***: FLOPs is not a direct metric of inference latency

Operator-level method

- Sum all the operators' latencies
- ***Disadvantage***: unaware of graph optimization

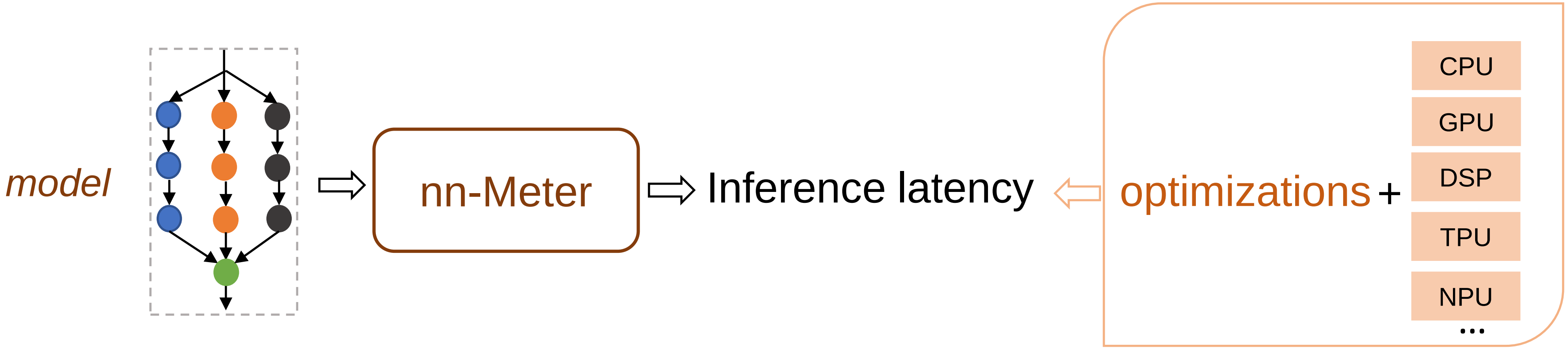
Model graph-level method

- GCN learns the graph optimization
- ***Disadvantage***: depends on the quality of training data (NN graphs), it's hard to generalize on unseen graphs

nn-Meter: capture the hardware optimizations

Goal: accurately predict the latency of arbitrary DNN models on diverse edge devices

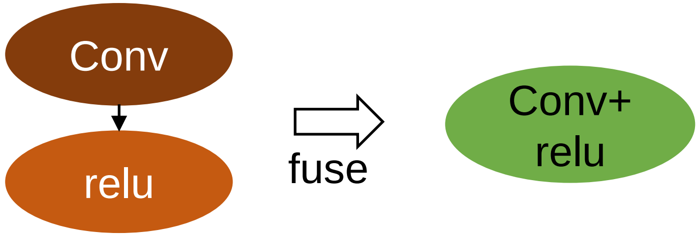
- Capture the various hardware optimizations that reduce model latency
- Be able to generalize on unseen model graphs



Challenge#1

- Too many device optimizations impact the inference latency
  - Different optimizations are included in diverse inference frameworks and hardware chips
  - Many of them are black-box
  - Model latency < sum (all the operators' latencies)
  - It's hard to accurately predict latency by a cost model

- Our key insight: we identify the most important graph optimization technique, the **operator fusion**

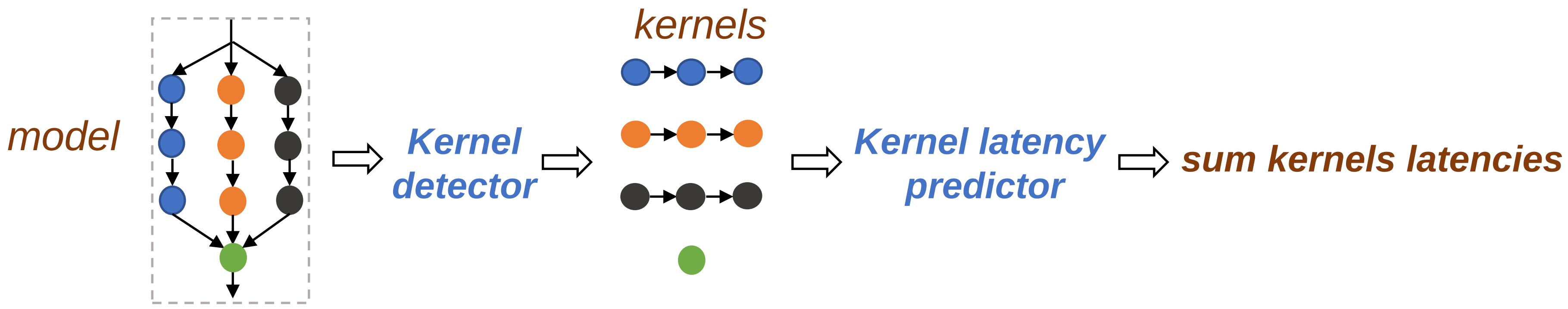


Intel VPU	Conv	relu	Fused Conv+relu
latency (ms)	**0.073**	0.029	**0.074**

An operator fusion example: **27.5% time saved**

Key idea of nn-Meter

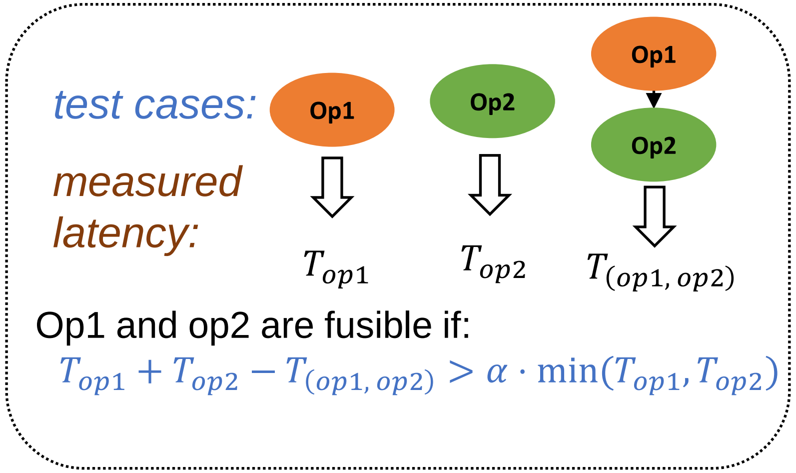
- *Definition: a kernel is the basic scheduling unit, can be a single operator or a fusion of multiple operators*
- Divide a whole model into ***kernels***, *conduct* ***kernel-level prediction***
  - Model latency is the sum of all kernels



nn-Meter tech#1: Automatic kernel detector

Fusion rule detection for black-box devices

- A set of test cases
- For every two operators, we generate 3 graphs
- Compare the latency difference



Backend	T_{pool} (μs)	T_{relu} (μs)	$T_{(pool,relu)}$ ($T_{pool} + T_{relu}$)	Rule
VPU	13	26	16 (39)	"pool_relu":True
GPU	5.08	3.50	6.00 (8.60)	"pool_relu":True
CPU	23.60	0.81	24.48 (24.42)	"pool_relu":False

A fusion detection example (pool, relu).

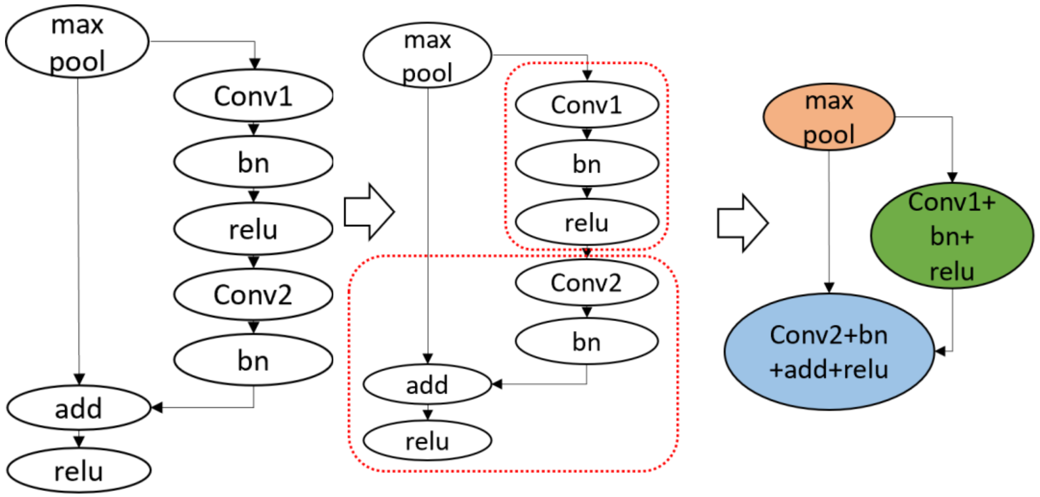
nn-Meter tech#1: Automatic kernel detector

Fusion rule detection for black-box devices

- A set of test cases:
- For every two operators, we generate 3 graphs
- Compare the latency difference

Kernel search by the fusion rules

- Apply the fusion rules to search maximum fused operators in target model



A resnet18 block example

Challenge#2

Large sample space for Conv-bn-relu

- Regarding latency, Conv-bn-relu is the most important kernel
- full size: 0.7 billion configurations
    - (total size: HW x K x S x Cin x Cout)

dimension	Configuration space
Input HW	224,112,56,32,28,27,14,13,8,7,1
Kernel size K	1,3,5,7,9
Stride S	1,2,4
Channel in Cin	Range(3,2160)
Channel out Cout	Range(16,2048)

0.7 billion configurations of Conv-bn-relu

Challenge#2

Large sample space for Conv-bn-relu

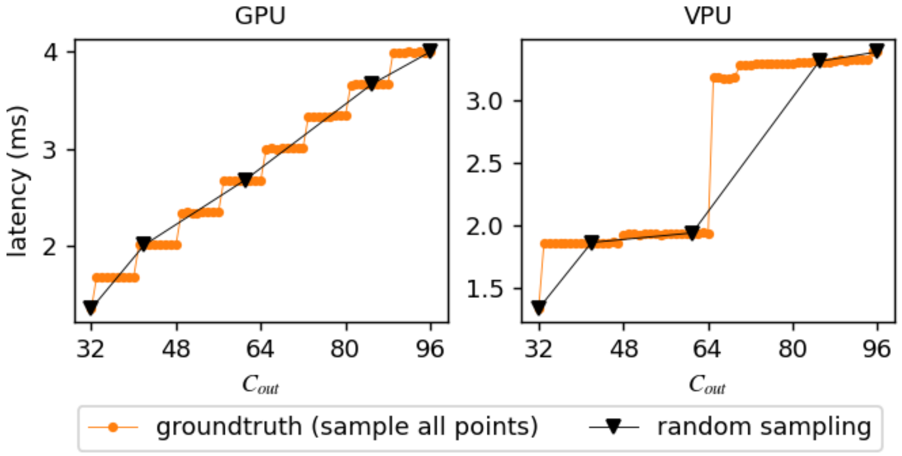
- Regarding latency, Conv-bn-relu is the most important kernel
- full size: 0.7 billion configurations
  - (total size: HW x K x S x Cin x Cout)

dimension	Configuration space
Input HW	224,112,56,32,28,27,14,13,8,7,1
Kernel size K	1,3,5,7,9
Stride S	1,2,4
Channel in Cin	Range(3,2160)
Channel out Cout	Range(16,2048)

0.7 billion configurations of Conv-bn-relu

Kernels show the non-linearity step latency pattern

- Random sample can miss hardware-crucial data

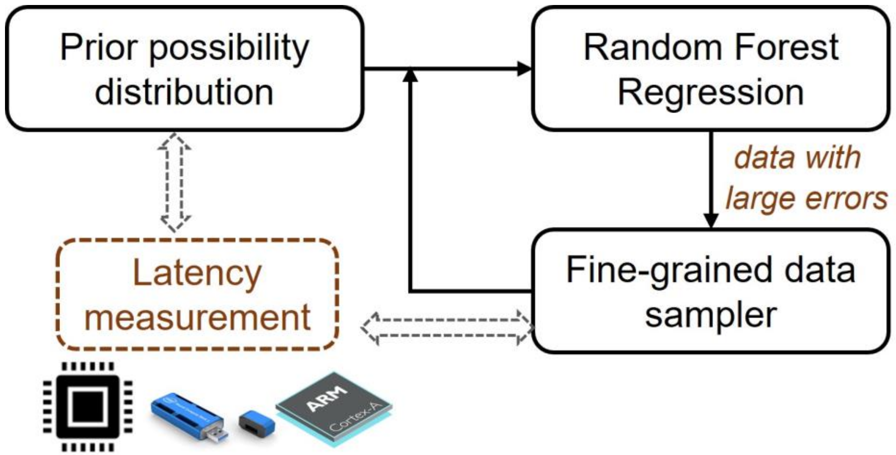


Cout and latency show a step pattern

nn-Meter tech#2: Adaptive data sampler

Sample the most beneficial data (kernel configuration) instead of random sampling

- ❑Sample configurations that are likely to be considered in model design
  - Prior possibility distribution: learned from model zoo
- ❑Fine-grained sampling around inaccurate prediction data



nn-Meter Implementation

- 4 types of popular edge platforms
- Detected kernels: 22 (CPU), 26 (GPUs), 22 (VPU)
- Kernel predictors: RandomForest models

	Device	Processor	Framework
CPU	Pixel4	CortexA76 CPU	TFLite v2.1
GPU	Xiaomi Mi9	Adreno 640 GPU	TFLite v2.1
GPU1	Pixel3XL	Adreno 630 GPU	TFLite v2.1
VPU	Intel NCS2	MyriadX VPU	OpenVINO2019R2[16]

Kernel	CPU		GPU		VPU	
	RMSE (ms)	±10% Acc.	RMSE (ms)	±10% Acc.	RMSE (ms)	±10% Acc.
Conv+bn+relu	6.24	89.1%	6.77	82.0%	18.74	67.9%
DWConv+bn+relu	0.21	97.4%	0.10	98.7%	0.28	89.4%
FC	0.64	94.3%	0.07	96.2%	0.12	93.9%
maxpool	0.12	89.6%	0.06	97.1%	0.21	89.7%
avgpool	1.94	99.0%	0.01	99.7%	0.26	95.4%
SE	0.45	87.1%	0.39	99.8%	0.44	99.0%
hswish	0.16	98.1%	0.01	100%	0.02	100%
channelshuffle	0.14	99.5%	-	-	0.35	100%
bn+relu	0.85	80.7%	0.01	100%	-	-
add+relu	0.10	93.7%	0.003	98.3%	0.02	98.9%
concat	0.09	89.3%	0.42	77.1%	-	-

Main kernel predictors and the performance

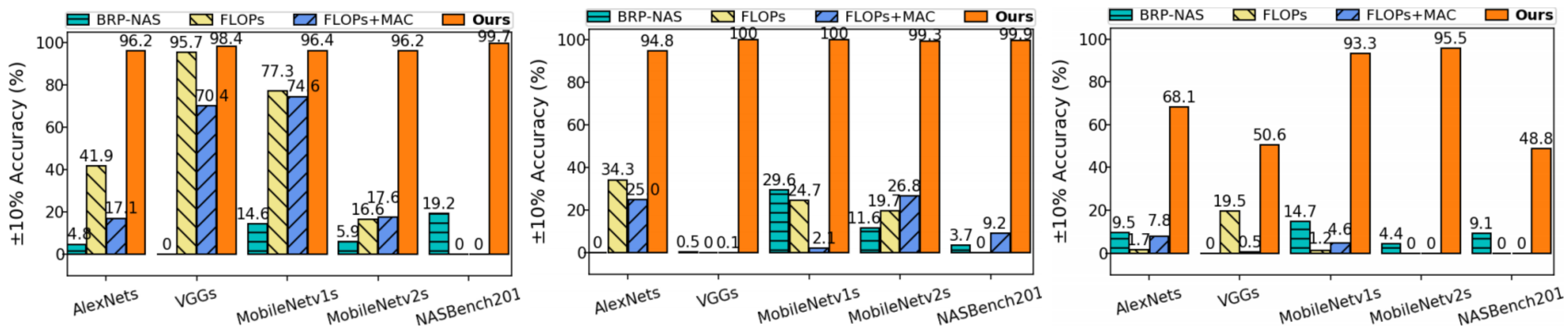
nn-Meter Evaluation

- Dataset: we generate 26k models and measure the latency on four devices
  - AlexNets: 2000 model variants of AlexNet (re-sample channel number, kernel size for each layer)
  - Large prediction scope

Model variants	avg FLOPs (M)	Latency(ms)		
		Mobile CPU min - max	Mobile GPU min - max	Intel VPU min - max
AlexNets	973	7.1 - 494.4	0.4 - 81.7	2.1 - 47.3
VGGs	28422	178.4 - 10289	20.1 - 1278	25.6 - 1467
DenseNets	1794	109.6 - 431.6	26.7 - 69.5	26.4 - 70.7
ResNets	4151	35.9 - 1921.7	7.3 - 329.5	10.7 - 145.5
SqueezNets	1597	42.7 - 524.9	7.5 - 72.2	6.9 - 57.3
GoogleNets	1475	115.5 - 274.6	23.0 - 49.0	12.2 - 24.4
MobileNetv1s	547	27.5 - 140.0	5.5 - 28.8	8.9 - 37.0
MobileNetv2s	392	15.6 - 211.0	3.5 - 37.0	11.3 - 86.1
MobileNetv3s	176	10.4 - 78.4	4.3 - 18.6	17.4 - 70.8
ShuffleNetv2s	307	22.2 - 84.3	-	20.9 - 44.2
MnasNets	327	25.6 - 99.3	5.8 - 24.1	19.8 - 60.9
ProxylessNass	532	34.5 - 195.9	7.9 - 72.2	18.0 - 77.8
NASBench201	97.5	5.6 - 27.9	1.8 - 8.3	2.3 - 6.4

nn-Meter Evaluation

- **Prediction accuracy**: 99.0% (CPU), 99.1% (Adreno640 GPU), 99.0% (Adreno630 GPU) and 83.4% (Intel VPU) on our benchmark dataset
- **Generalization performance on unseen model graphs**
  - Comparison baselines: FLOPS, FLOPS+MAC, BRP-NAS (GCN),
  - On average: nn-Meter achieves 89.2%, significantly better than FLOPs (22.1%), FLOPs+MAC (17.1%), and BRP-NAS (8.5%)



(a) CPU (b) GPU (c) VPU

nn-Meter Evaluation

- **Comparing with operator-level prediction**
  - nn-Meter achieves +8%(CPU), +45.5%(GPU) and +75.1%(VPU) higher prediction accuracy
- **Adaptive data sampling vs. random data sampling**
- **Low measurement cost for building predictors for new device**

Device	Random Sampling		Adaptive Sampling	
	RMSE	±10% Acc.	RMSE	±10% Acc.
CPU	25.47 ms	21.92%	10.13 ms	71.78%
GPU	1.67 ms	48.70%	1.19 ms	75.34%
VPU	7.87 ms	23.98%	7.58 ms	54.33%

Prediction performance for conv-bn-relu

	CPU	GPU	VPU
total measure time	2.5 *days*	1 *day*	4.4 *days*

Measurement cost

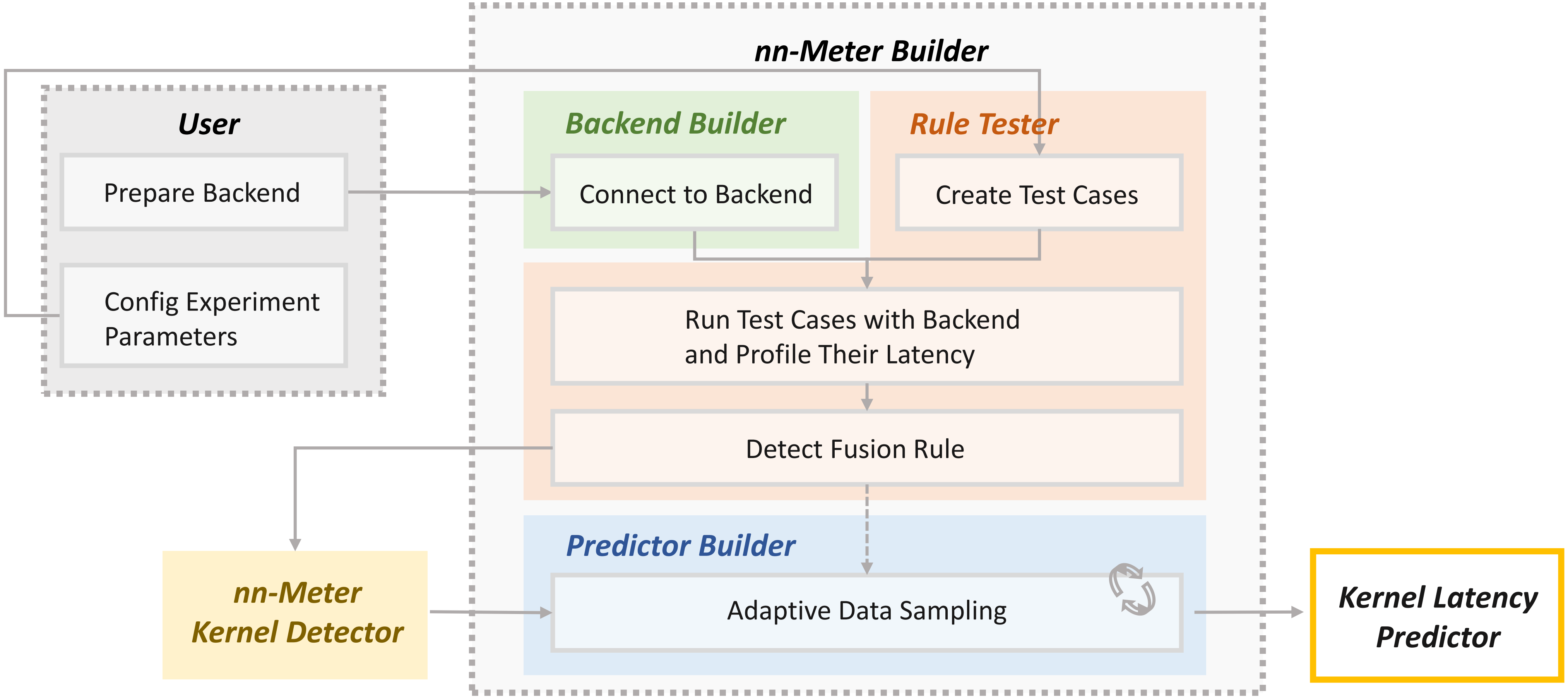
nn-Meter Opensource

https://github.com/microsoft/nn-Meter

- Prediction tools
  - latency prediction on 4 devices
  - Support tensorflow, onnx, pytorch, and NNI models
  - Input models: model file or pytorch NN module instance
- Benchmark dataset
  - 26k CNN model graphs and their latency
- Hardware-aware NAS algorithms in NNI
  - Random search
  - ProxylessNAS: gradient-based and RL
- Building tools
  - Build latency predictors for custom devices
  - (more types of inference frameworks and hardware chips)

nn-Meter Building Tools

Use nn-Meter to build latency predictor for your own device!



Summary

- nn-Meter: an efficient and novel system to predict DNN model inference latency on various edge devices
  - kernel-level prediction and adaptive data sampler
  - Key insight #1: kernel can capture the runtime optimization
  - Key insight #2: learn to sample the most important data
- Evaluated a large dataset on four edge platforms
- Impressive high prediction accuracy
  - 99.0% (CPU), 99.1% (Adreno640 GPU), 99.0% (Adreno630 GPU) and 83.4% (Intel VPU)