# RoofTune: Accelerating the Tuning Process Based on Roofline Model

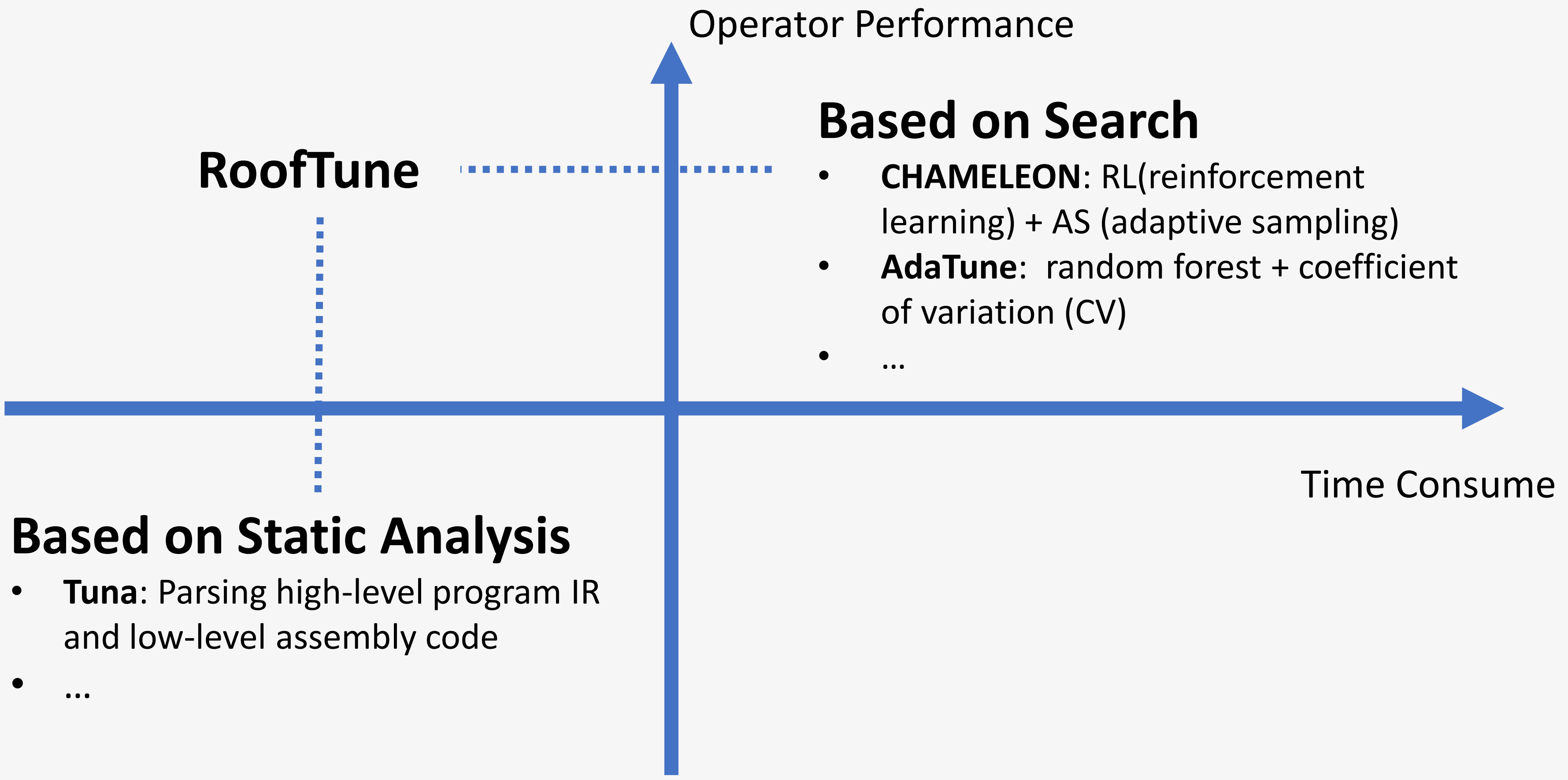
Hui Zhong, HaiWen Fu, XiaoHua Shi
钟辉，付海文，史晓华

Beihang University

Apache TVM and Open Source ML Acceleration Conference
December 15-17 2021

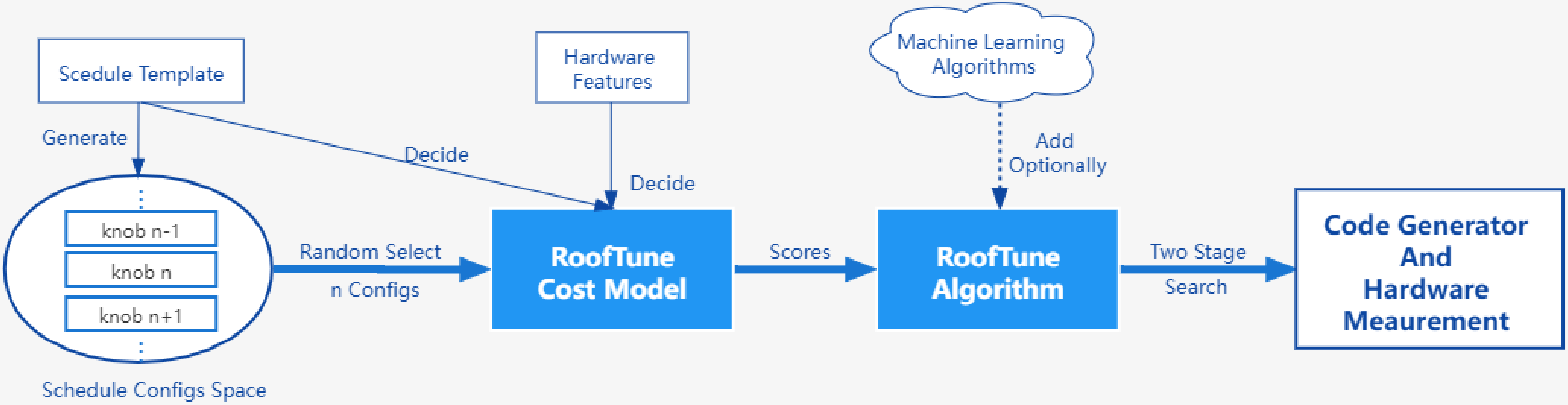
# Motivation & Related Work

- **The current tuning process is extremely time-consuming…**
- Current work can be summarized in two aspects:



# Overall Design of RoofTune

- **Cost Model:** Manual designed, based on Roofline Model, easily to be deployed
- **Search Algorithm:** Combined with the cost model, two stage search



# Roofline Model

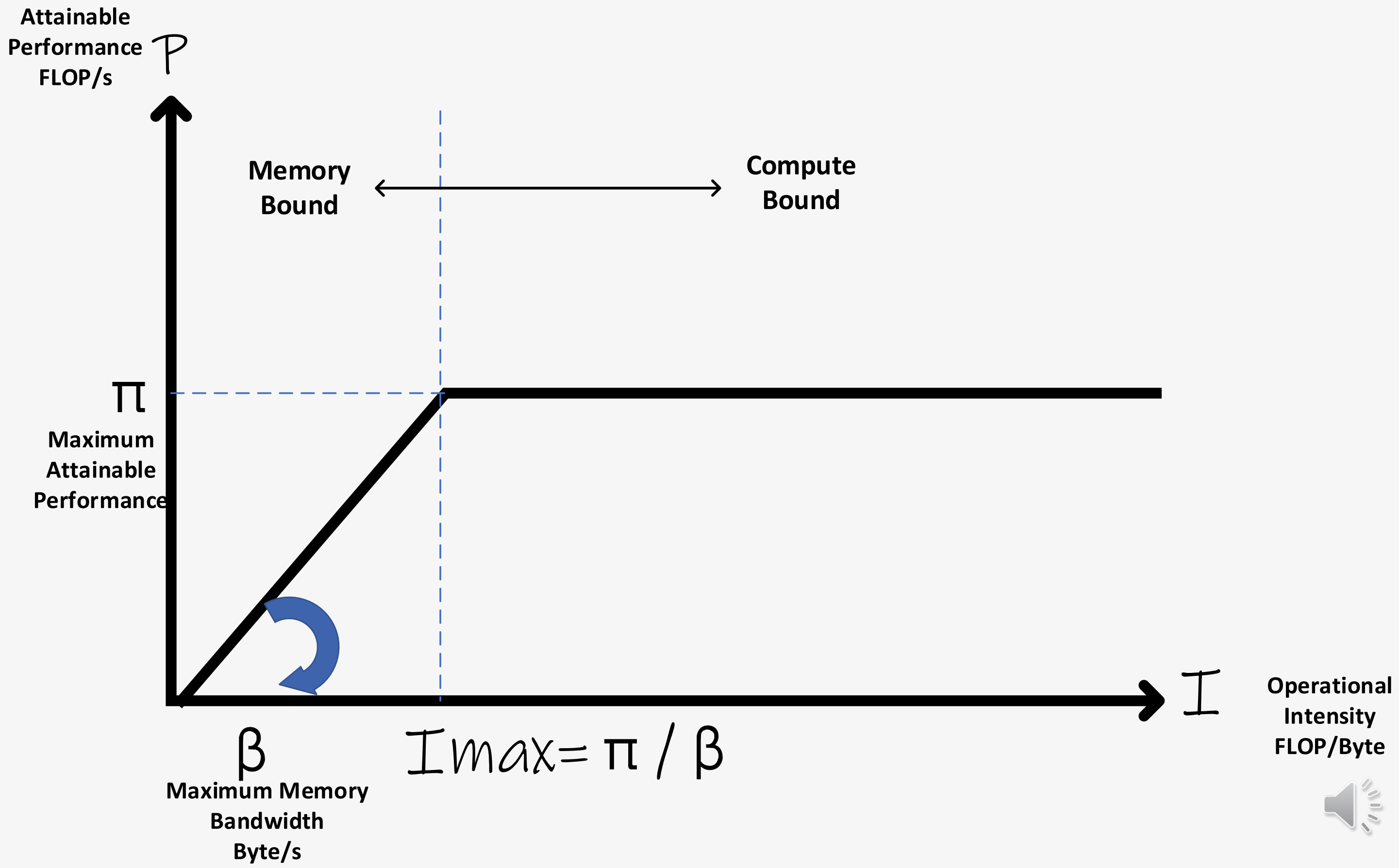
**Roofline Model tells us :**

***Operational intensity*** can be used to predict the operator's performance.

**Ideas:**

We introduce Roofline Model to kernel's level (fine grained):

- ***Operational intensity*** can be expressed by some specific schedule parameters.



# Cost Model Design Principle

$$Score = I_{kernel} \times Concurrency$$

kernel's arithmetic intensity:

$I_{kernel} = W_{kernel} \div T_{kernel}$

kernel's execution concurrency:

$Concurrency = f_C(knob0, knob1, knob2 \ldots)$

kernel's arithmetic amount:

$W_{kernel} = f_w(knob0, knob1, knob2 \ldots)$

kernel's memory traffic:

$T_{kernel} = f_T(knob0, knob1, knob2 \ldots)$

# Conv2d Cost Model Design on GPU

KNOBS IN THE SCHEDULE CONFIG SPACE TO OPTIMIZE CONVOLUTION

| KNOBS | DEFINITION |
|---|---|
| tile_f,tile_y,tile_x | Conv's data and weight factors for tiling and binding |
| tile_rc,tile_ry,tile_rx | Channels,height and width of filters for tiling reduction axis |
| auto_unroll_max_step | Threshold of number of steps in the loop to be automatically unrolled |
| unroll_explicit | Explicitly unroll loop |

$$Score = I_{kernel} \times Concurrency$$

kernel's arithmetic intensity:

$I_{kernel} = W_{kernel} \div T_{kernel}$

kernel's execution concurrency(consider blocks in GPU):

$$Concurrency = \begin{cases} 1 & , if\ blocks > SPs \times 3 \\ blocks \div (SPs \times 3) & , if\ blocks \leq SPs \times 3 \end{cases}$$

kernel's arithmetic amount:

$W_{kernel} = \prod(f[3], y[3], x[3], f[1], y[0], x[1], rf[0], f[1], y[0], ry[1], rx[0], rx[1])$

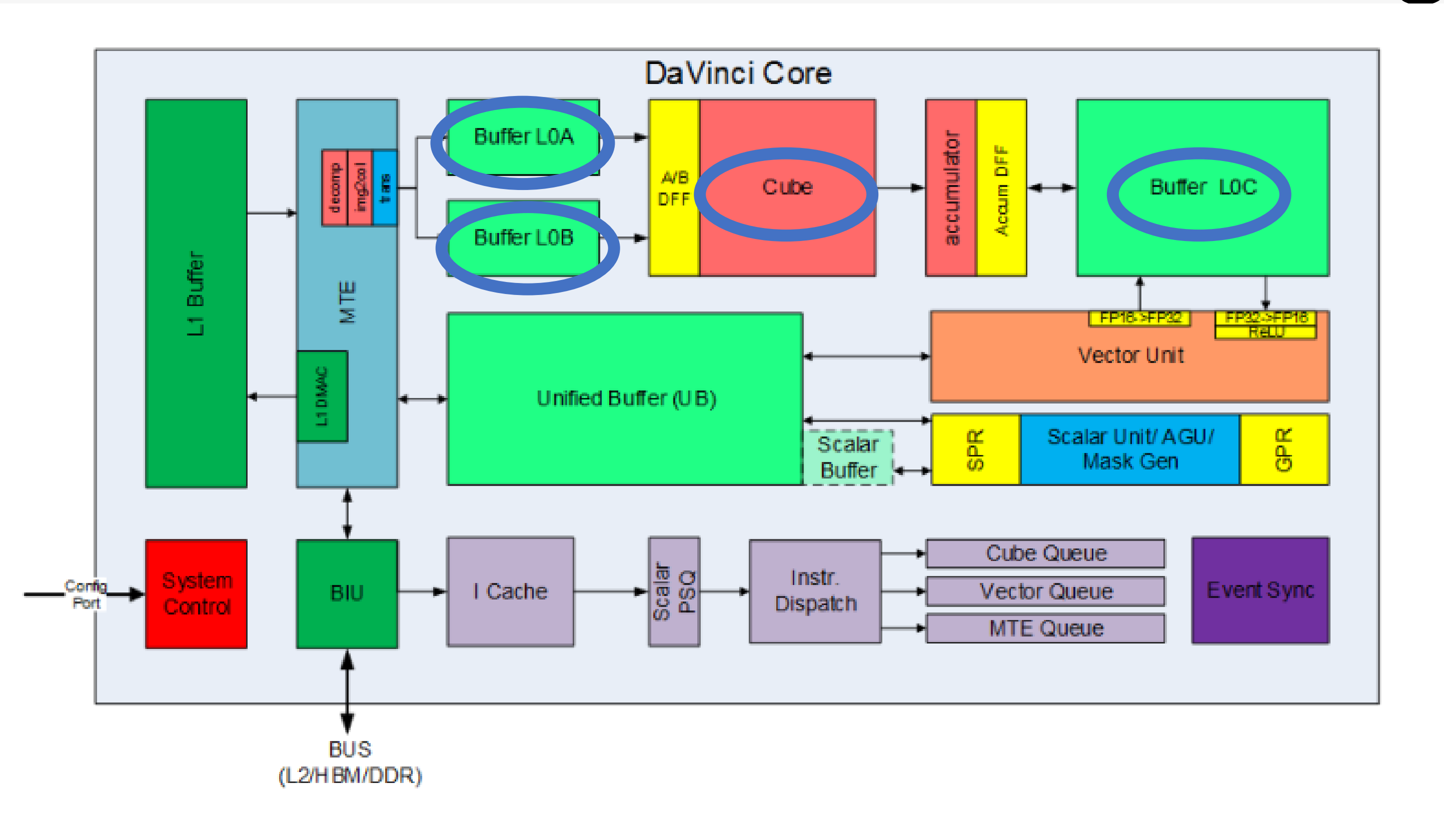
kernel's memory traffic:

$T_{kernel} = data\text{-}io + weight\text{-}io + out\text{-}io$

kernel's data memory traffic: … …

$data\text{-}io = data\text{-}shape[2] \times weight\text{-}shape[2] \div \prod(f[2], y[2], x[2], y[0], x[0])$

# Conv2d Cost Model Design on Huawei NPU




KNOBS IN THE SCHEDULE CONFIG SPACE TO OPTIMIZE CONVOLUTION

| KNOBS | DEFINITION |
|---|---|
| AL1_shape,BL1_shape | Allocate Buffer L1 to data, weights |
| AL0_matrix,BL0_matrix, CL0_matrix | Allocate Buffer L0 to data, weights, outputs |
| AUB_shape,BUB_shape, CUB_shape | Allocate Unified Buffer to data, weights, outputs |
| (AL1,BL1,AL0,BL0,CL0, AUB,BUB,CUB)_pbuffer | Decide whether to set double buffer at given every on-chip buffers |
| block_dim | Conv's data factors for tiling and binding on each AICore |
| (A,B)_overhead_opt_flag, n_bef_group_flag, n_bef_batch_flag | Some schedule adjustments that are difficult to estimate |

$$Score = I_{kernel} \times Concurrency$$

kernel's arithmetic intensity:

$$I_{kernel} = W_{kernel} \div T_{kernel}$$

kernel's arithmetic amount:

$$W_{kernel} = out\text{-}shape \times weight\text{-}shape$$

kernel's memory traffic:

$$T_{kernel} = data\text{-}io + weight\text{-}io + out\text{-}io$$

kernel's data memory traffic:

$$data\text{-}io = data\text{-}shape \div AL0\text{-}matrix$$

...

...

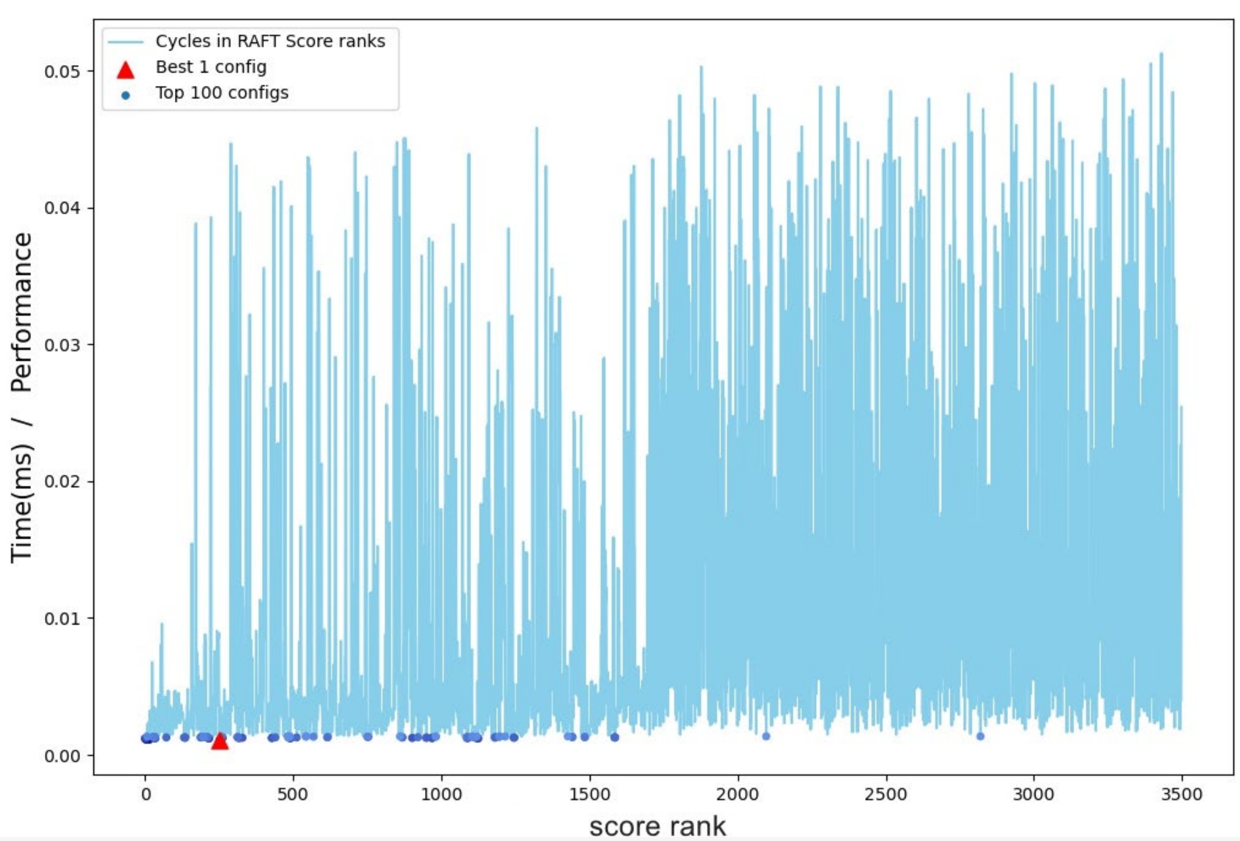
kernel's execution concurrency (consider AICores number in NPU):

$$Concurrency = block\_dim$$

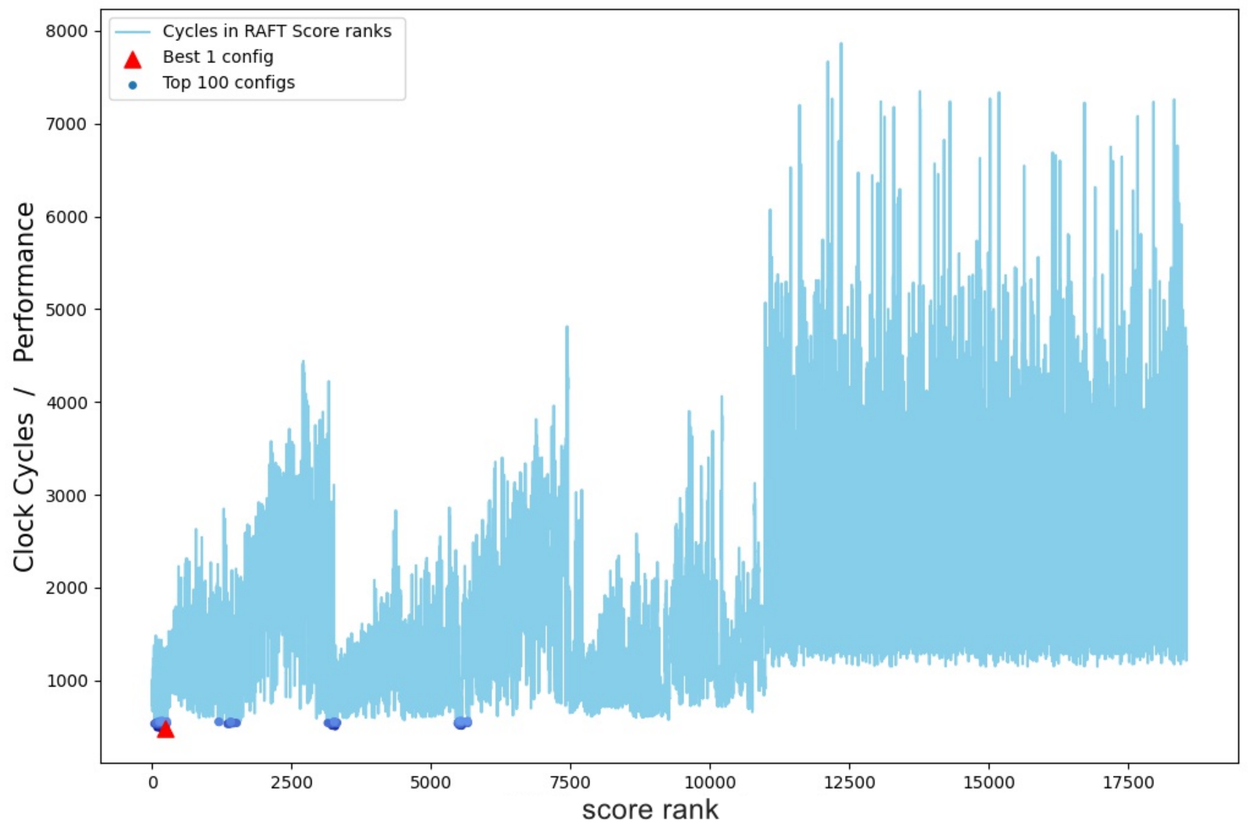
# Cost Model Offline Validation

Randomly select a large number of configs from the all schedule space

- Y-axis: operator execution time
- X-axis: scores rank from high to low




GPU

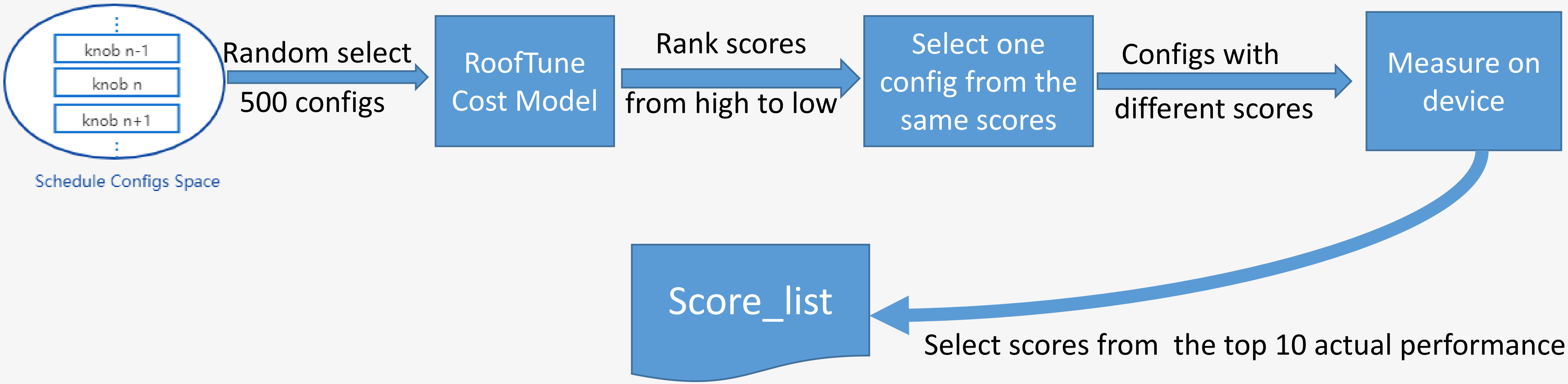



NPU

# RoofTune Two-stage Search Algorithm

**First Stage:**

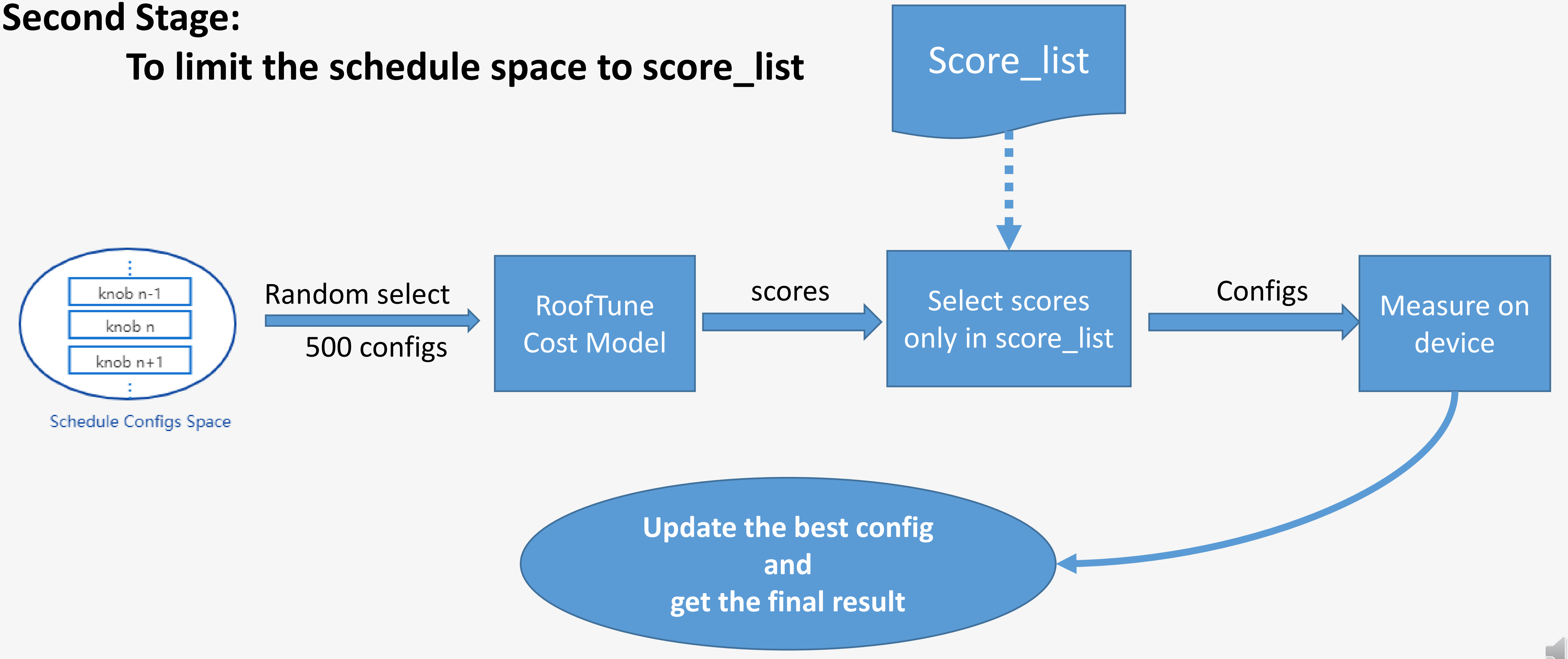
**To find out the promising scores**



# RAFT based Two-stage searching algorithm

**Second Stage:**

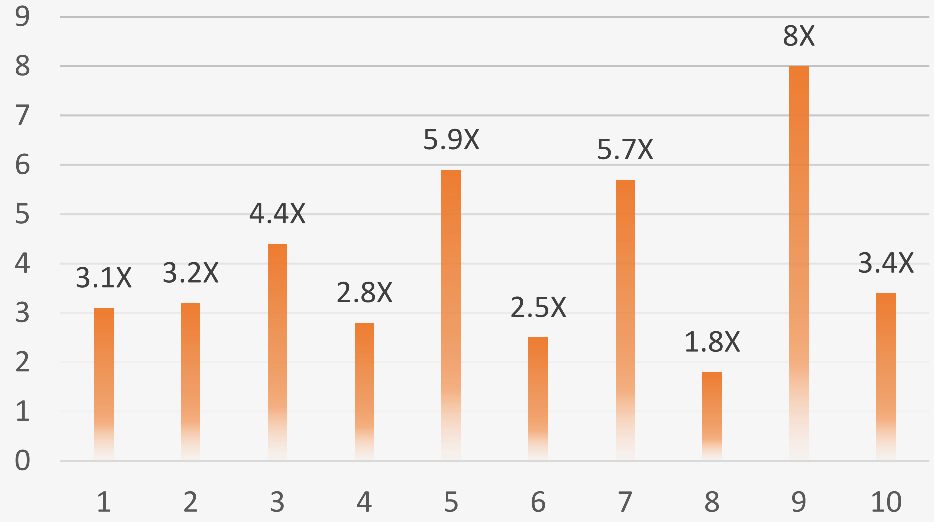
**To limit the schedule space to score_list**



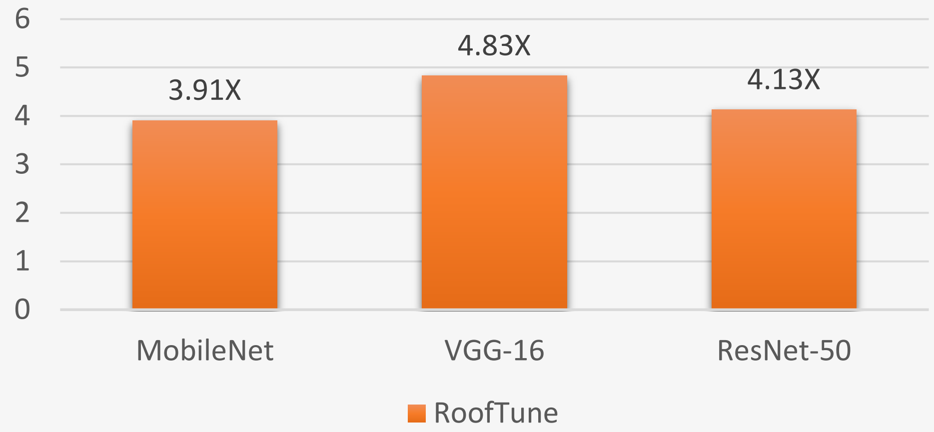
# Search time speedup ratio

GPU | NPU

## MOBILENET CONV2D LAYER (GPU)

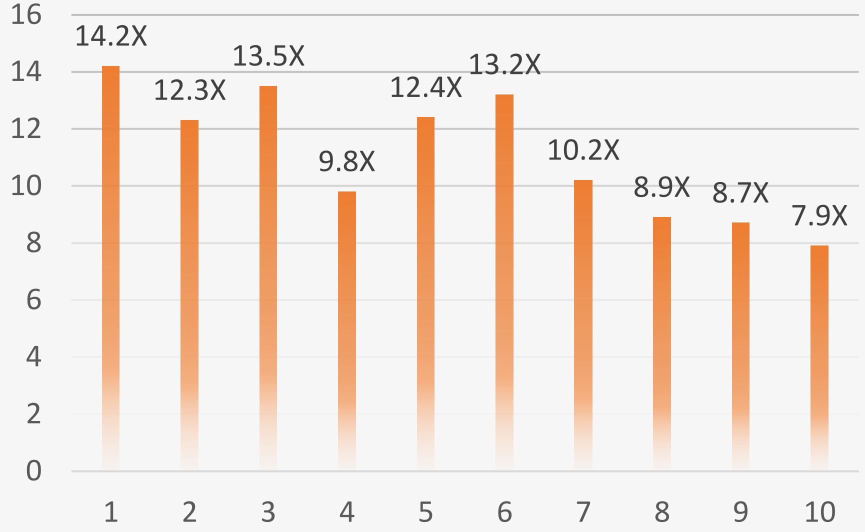



## End To End Optimization Time (GPU)

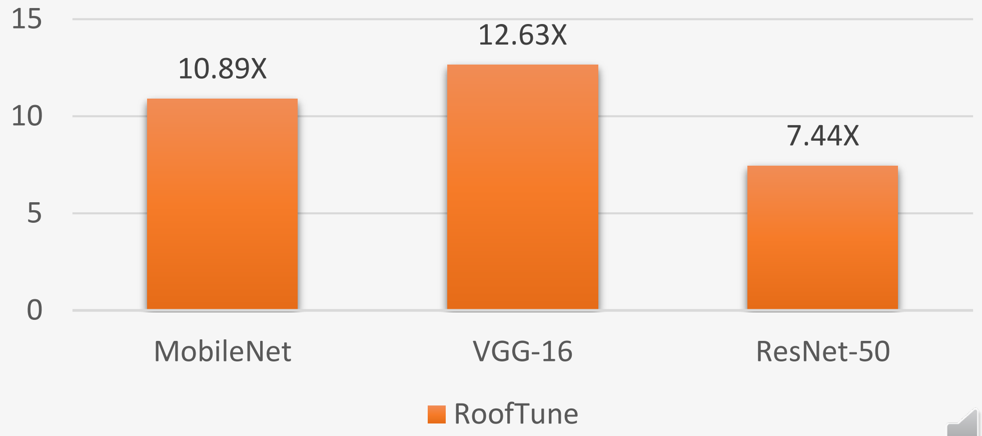



RoofTune vs XGB on AutoTVM(GPU)

## MOBILENET CONV2D LAYER (NPU)



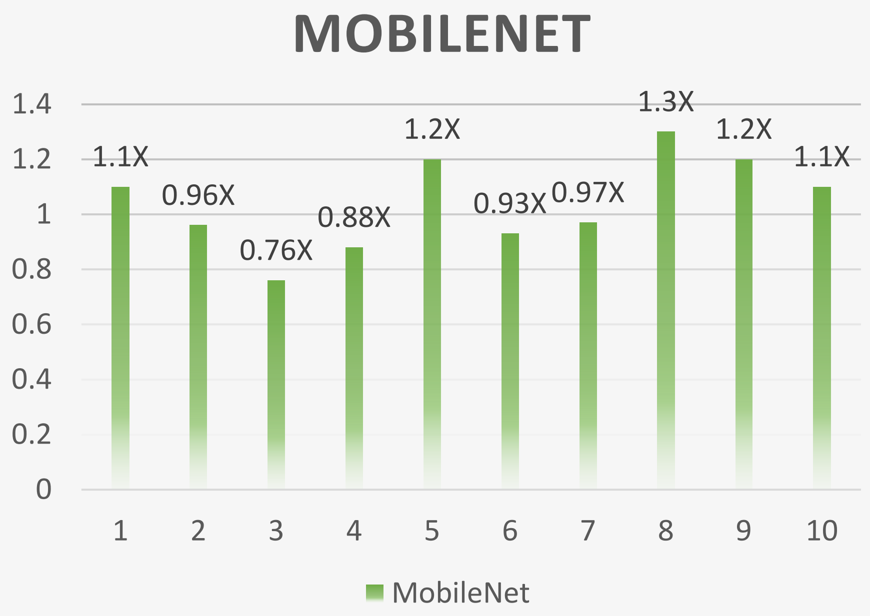

## End To End Optimization Time (NPU)

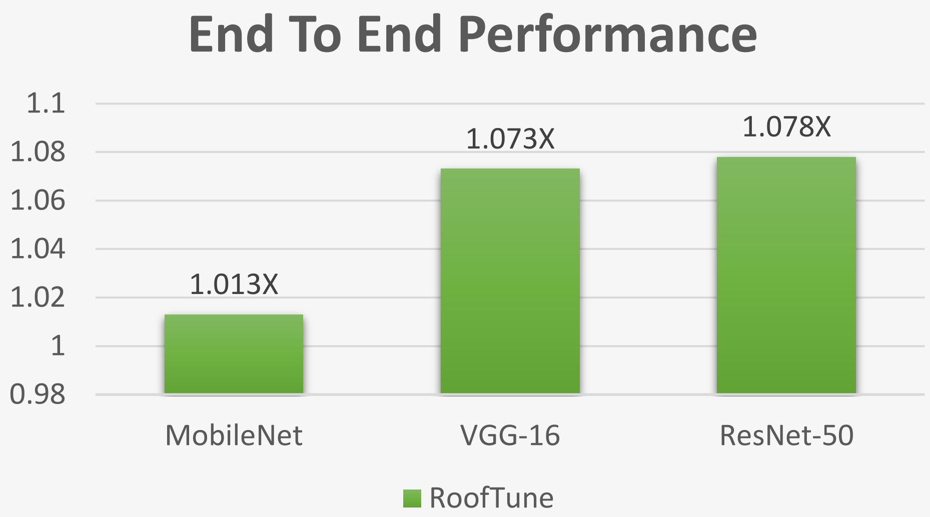



RoofTune vs GA on TBE’s AutoTune(NPU)

# Performance Evaluation

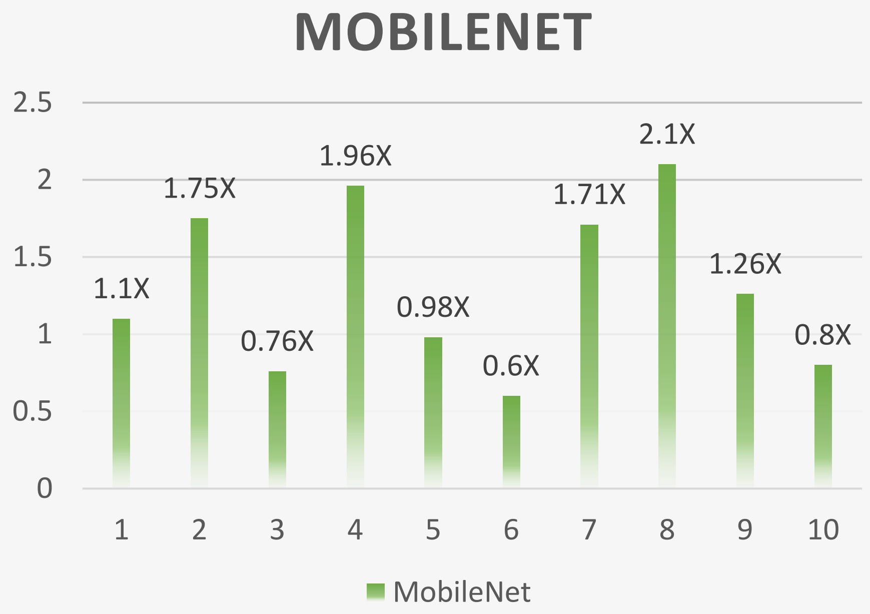
## GPU

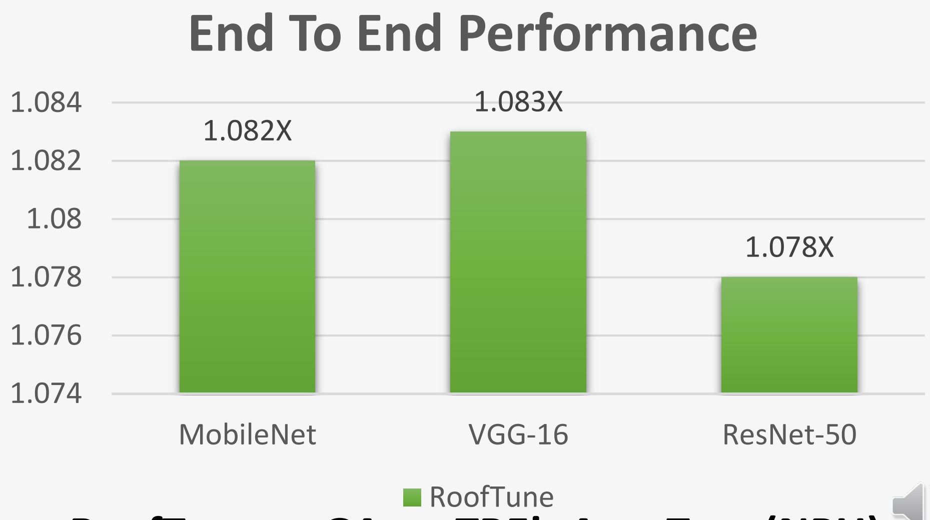



**RoofTune vs XGB on AutoTVM(GPU)**

## NPU




**RoofTune vs GA on TBE's AutoTune(NPU)**

# Acknowledgment


- This work is funded by Huawei Technology.


HUAWEI