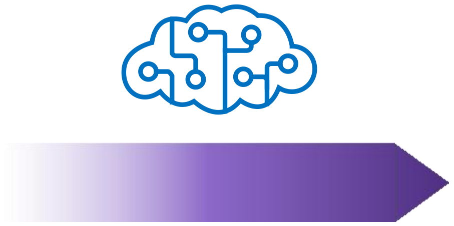
Microsoft

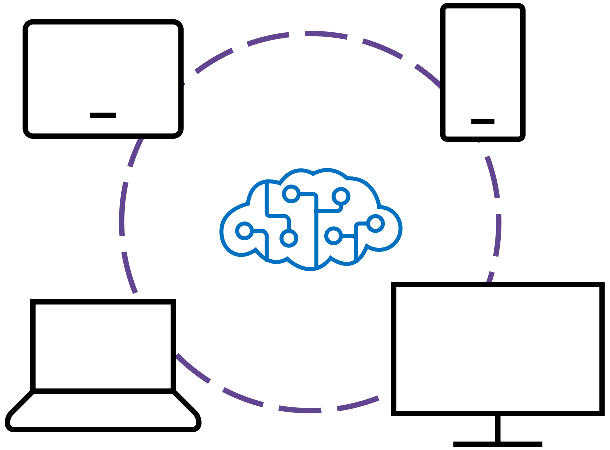
NN-Stretch:
Automatic Neural Network Branching for Parallel Inference on Heterogeneous Multi-Processors

Jianyu Wei[1,2], **Ting Cao**[1], Shijie Cao[1], Shiqi Jiang[1], Shaowei Fu[2], Mao Yang[1], Yanyong Zhang[2], Yunxin Liu[3]

Microsoft Research[1],
University of Science and Technology of China[2],
Institute for AI Industry Research, Tsinghua University[3]

On-device DNN Inference is Highly Needed

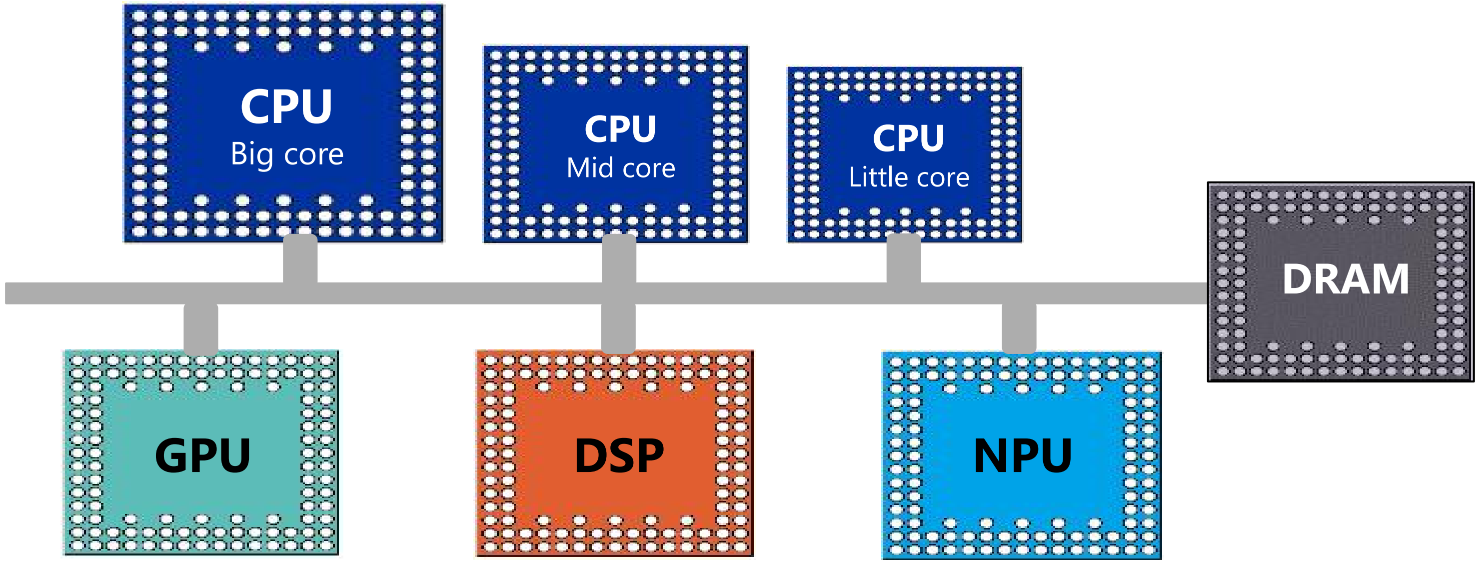




Cost of on-cloud inference strains model shipping

Inference on Edge devices
Low cost, quicker response time new scenarios and new market, …

Uniqueness of Edge Devices: Heterogeneous SoC

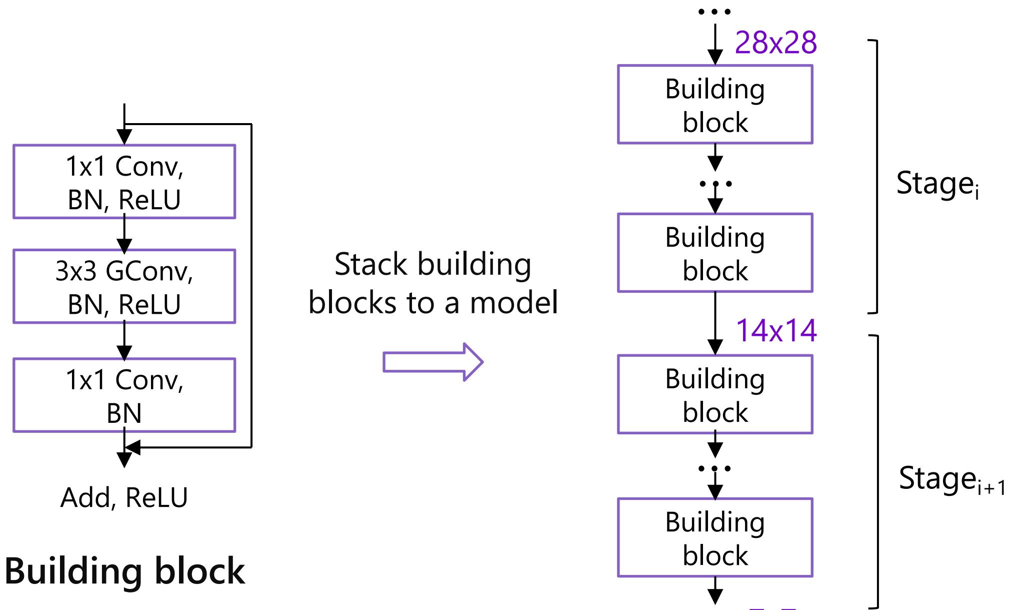


- **Shared memory**
- **Comparable processor performance**

Empower concurrent DNN inference

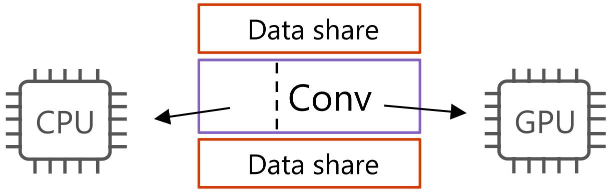
Sequential DNN Limits Concurrency

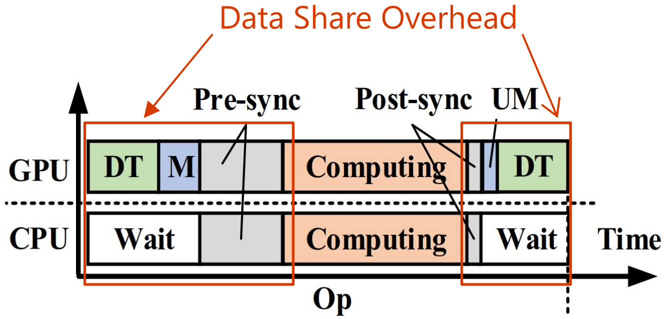
Most DNNs are composed of sequential operators



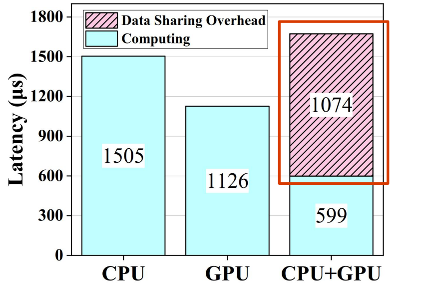
Example: RegNetX

Intra-operator parallelism needs high-FLOPS operators





Data share cost: Data transform, mapping, unmapping and sync



Data share cost example: 1x1 Conv <52,52,256,128>

Solution: change the model structure and enable inter-operator parallelism to reduce data sharing overhead

Mismatch between Heterogeneous Architecture and Model Structure

Model adaption techniques

- **Model scaling:** scale the depth, width of a given model;
- **Model compression:** removing the weight redundancy to compress a model into a smaller size;
- **Processor-tailored NN design:** designs NNs using hardware-efficient operators or hyperparameters.

They greatly facilitate the deployment of NN models to mobile devices

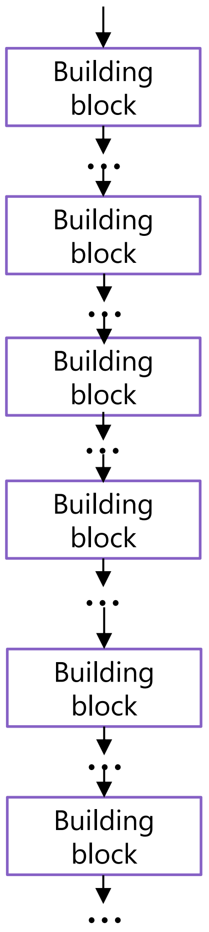
None of them has considered model adaption for the unique feature of mobile devices, i.e., CPUGPU heterogeneous computing.

Call for General and Automatic Model Branching

Idea: automatically transform a given single-branch model to a balanced multibranch structure

Two questions:

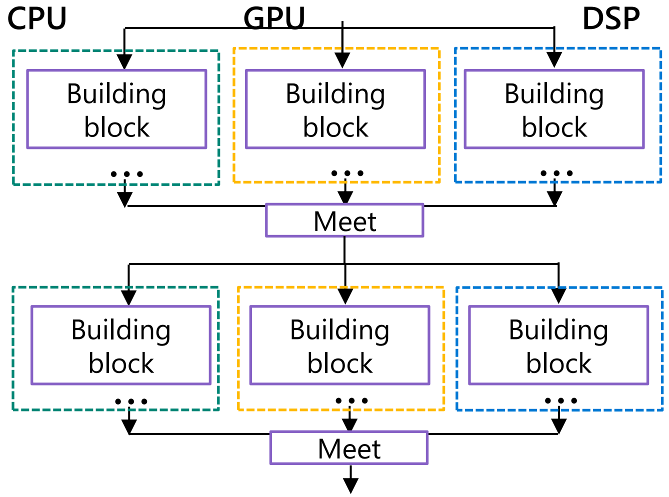
1. **Where to converge the branches, named as meeting point, to merge features extracted by each branch ?**
2. **How to scale each branch ?**



Transform

a long, seq DNN to

short, multibranch

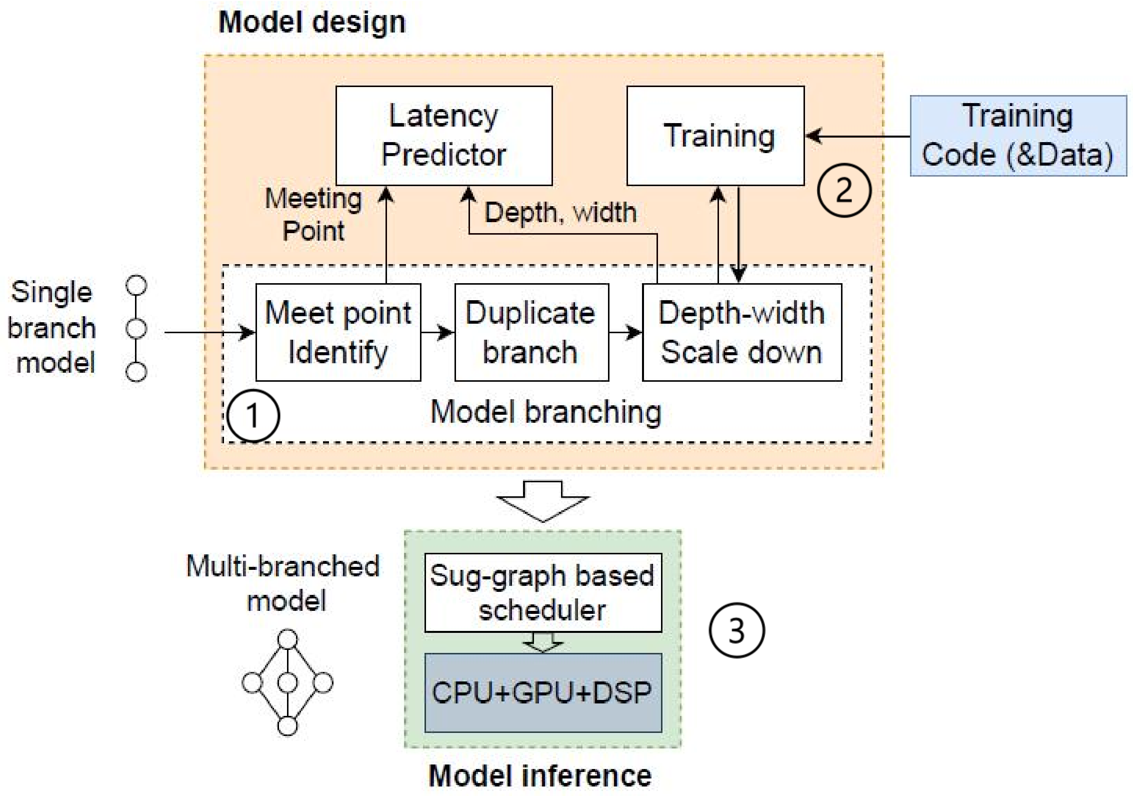


Multi-branch enables
inter-operator parallelism
to reduce latency

NN-Stretch: A New DNN Adaption Paradigm

- **Model design**
  - Input the a sequential model, out put the branched model;
  - Identify the meeting points, considering both latency and accuracy;
  - Duplicate a segment into multiple branches and then scales down the width and depth;
- **Model Inference**
  - Partition each branch into a sub-graph, as the basic scheduling unit to run on a processor;
- **Advantages of NN-Stretch**
  - Latency reduction without scarifying accuracy
  - No additional efforts from model designers
  - No more overhead than other model adaption techniques

NN-Stretch system overview



Model branching -> training -> Concurrent inference

Challenge: DNN Branching Space is Huge

Hyper Params	Potential Values
Number of meeting points N_{meet}	1 to number of layers
Number of branches N_{branch}	1 to number of processors
Meeting point location L_{meet}	1 to number of layers
Depth scale ratio R_{depth}	0.1 to 1.0 for each branch
Width scale ratio R_{width}	0.1 to 1.0 for each branch

Each setting results in different latency and accuracy tradeoff

Target:

$$\min_{N_{meet}, N_{branch}, L_{meet}, R_{depth}, R_{width}} Latency(G_{branch})$$
$$\text{s.t.} \quad Accuracy(G_{branch}) \geq target\_accuracy$$

- **Optimization problem**: to find the branching hyperparameter settings with min latency and no accuracy loss
  - Example: The space for ResNet-50 is $O(10^{50})$

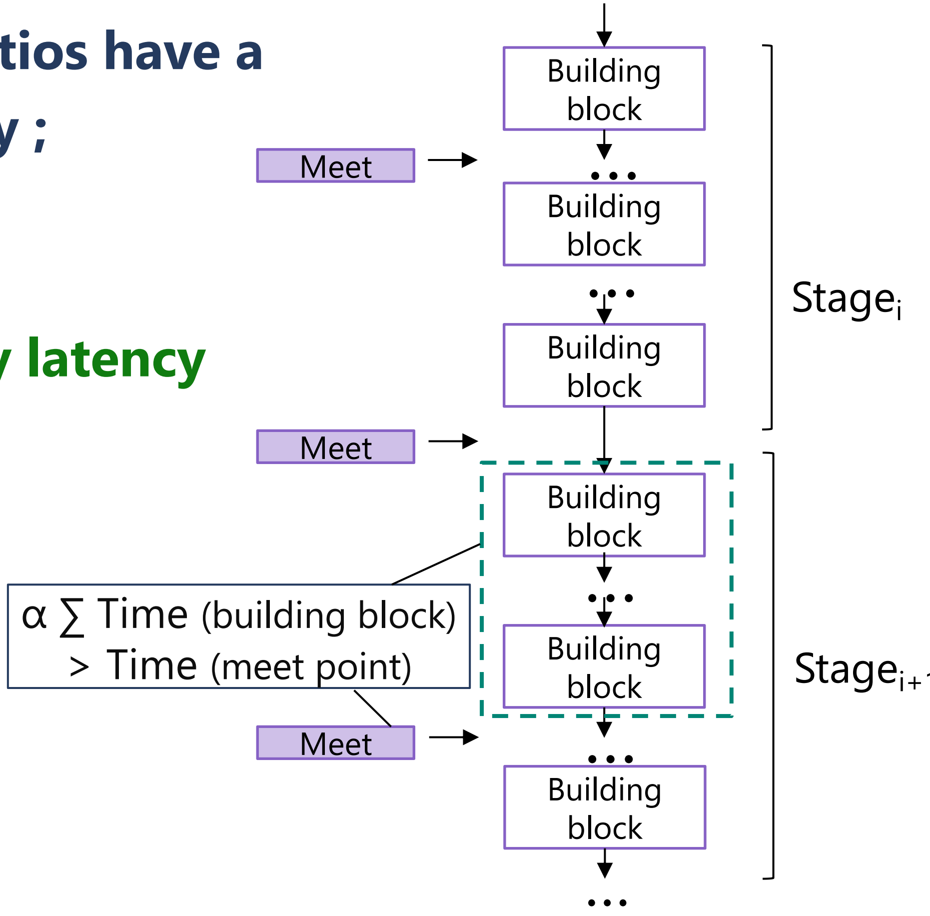
Narrowing Down Design Space from Inference Latency Perspectives

- **The hyper-parameters of meeting points and scaling ratios have a direct and statistical correlation to the inference latency ;**
- **The latency can be accurately and quickly predicted;**
- **Formulate the relationship between latency and hyper-parameters, pruning away solutions with unsatisfactory latency**

Cost-amortized meeting point identification

constraints on the number of and locations of meeting points

$$Latency(communication) <= \alpha \cdot \sum_{j=L[i]}^{L[i+1]-1} Latency(layer_j), \forall i \in [0, N_{meet} - 1)$$



Sequential Model

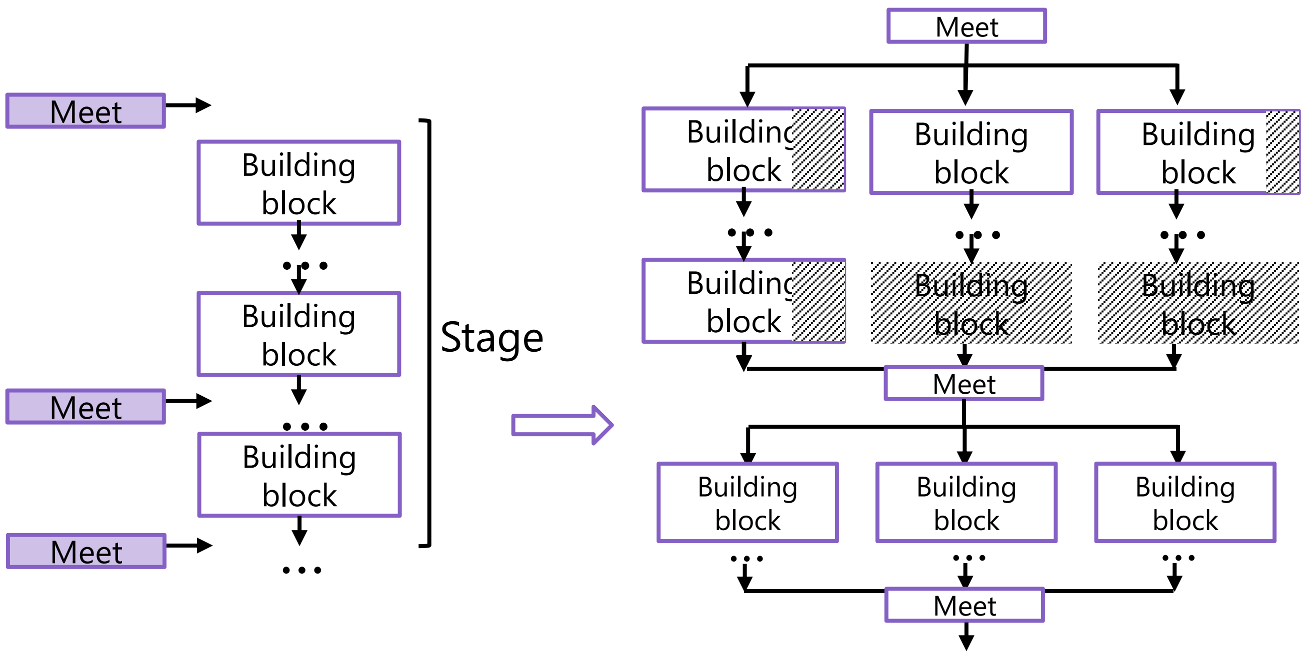
Narrowing Down Design Space from Inference Latency Perspectives

Heterogeneity-aware depth-width scaling

- set lower bounds on the depth/width-scaling ratios to avoid straggler branches that can increase the overall latency

$$Latency_{proc_i}(branch_i) <= \beta \cdot Latency_{proc_j}(original\_segment), \forall i \in [1, N_{branch}]$$

- GPU, DSP scales depth first as wider ops can saturate the computation units



Identify meet point → Duplicate branches → Depth-width scale for each branch

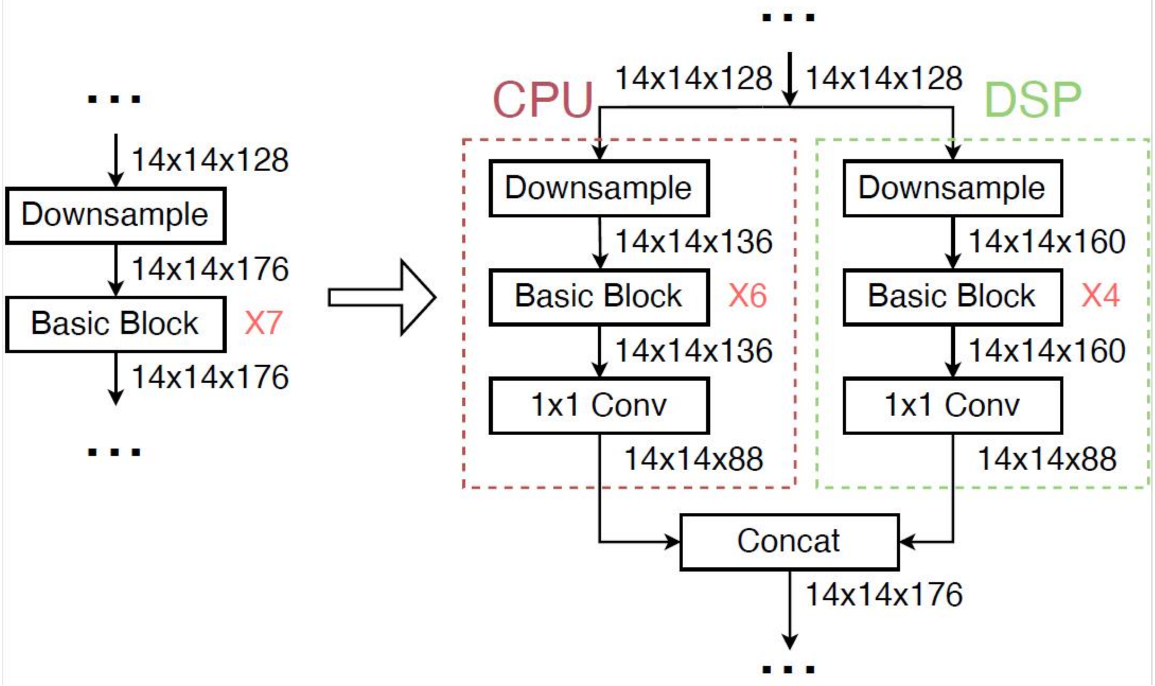
Narrowing Down Design Space from Model Accuracy Perspectives

Capacity-guaranteed depth-width scaling

- Previous works point out that extremely **shallow** or **narrow** networks have low accuracy;

$Branch_1$	$Branch_2$	Top-1 Acc.	Top-5 Acc.	GFLOPs	#Params (M)
(0.5,1)	**(0.75,0.4)**	**72.81**	**92.91**	0.83	1.38
(0.5,1)	(0.3,1)	71.52	92.21	0.90	1.50
(1,0.5)	(1,0.3)	70.39	91.80	0.83	1.37

- Lower bounds on scaling ratios can be obtained through preliminary experiments for different models



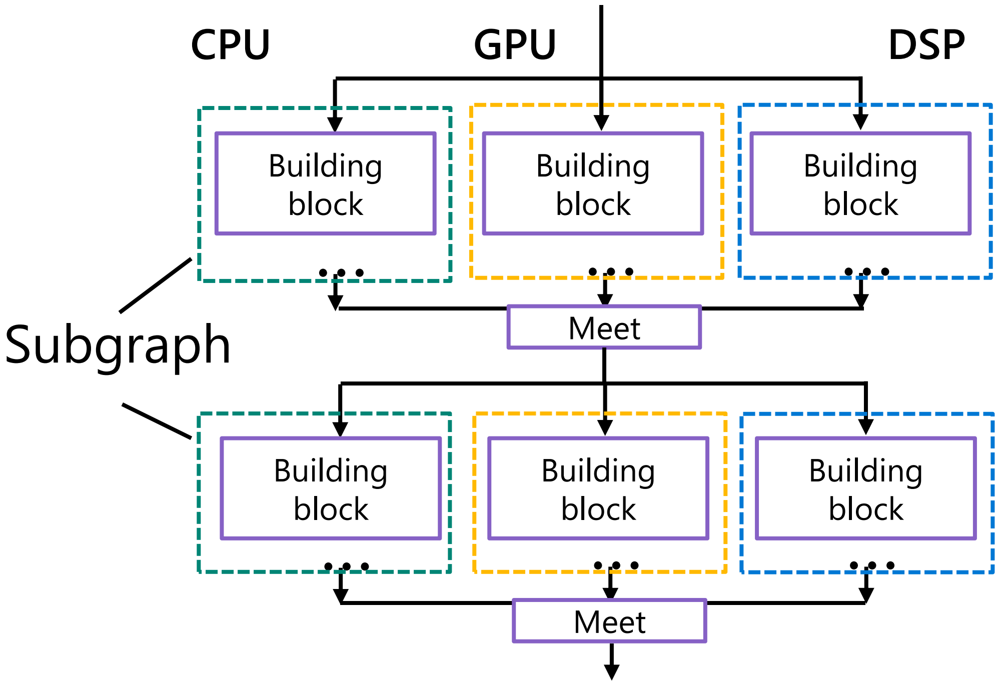
Example: branched EfficientNet (Stage 5)

"Hard" Latency & "Soft" Accuracy Constraints

Space is reduced from 10^50 to O(10) by the constraints.

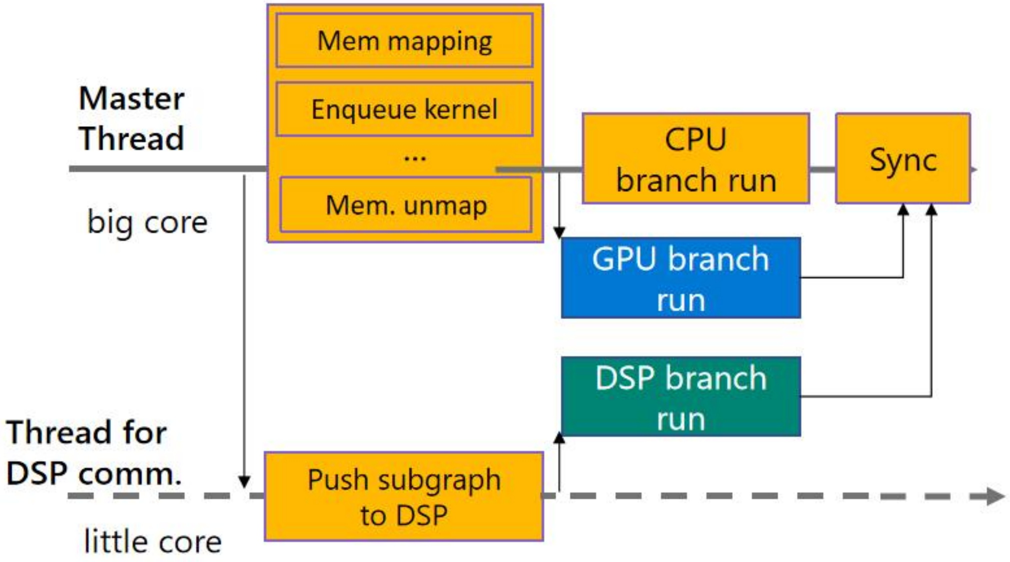
Multi-branch Inference: Sub-graph-based Scheduler

- **Inference design:** A branch is mapped as a sub-graph, as the scheduling unit among processors
- **Definition:** A sub-graph is a sequence of ops with only the first one depending on the output from other processors



Multi-branch Inference: Sub-graph-based Scheduler

- **Challenge:**
  - Different communication mechanisms among processors
  - GPU: low-level APIs supporting synchronous or asynchronous run to the CPU
  - DSP: high-level operator-level APIs only supporting synchronous run
- **Threadpool implementation:**
  - A new thread on little core for DSP
  - The master thread for asynchronous communicate with GPU



Algorithm 1: Sub-graph-based spatial scheduler

```
for subgraph ∈ TopoSorted(Subgraphs) do
    if subgraph.isMeetingPoint then
        ExecuteOnCpu(CpuSubgraph);
        OpenCLQueue.Finish();
        DspFuture.Wait();
        Execute(subgraph);
    else
        ParallelExecute(subgraph);
    end
end
Function ParallelExecute(subgraph):
    if subgraph.isCpuSubgraph then
        CpuSubgraph = subgraph;
    else if subgraph.isGpuSubgraph then
        EnqueueInputMapBuffer();
        EnqueueInputTransformKernel();
        EnqueueGpuKernels(subgraph);
        EnqueueOutputTransformKernel();
        EnqueueOutputMapBuffer();
    else if subgraph.isDspSubgraph then
        DspFuture = ThreadPool.push(subgraph);
```

Experiment Setting

- Models:

Model	FLOPs (G)	Params (M)
ResNet34	3.7	21.8
ResNet50	4.1	25.6
RegNetX-1.6GF	1.6	11.2
RegNetX-4GF	4	20.6
EfficientNet-Lite4	2.6	13
EfficientNet-B5	10.3	30.4

- Training codebase:

pycls image classification in PyTorch

- Devices:

	Xiaomi 9	Pixel 6	Xiaomi 11
SoC	Snapdragon 855	Google Tensor	Snapdragon 888
CPU	Kyro 485	Cortex-X1/A76/A55	Kyro 680
GPU	Adreno 640	Mali-G78	Adreno 660
DSP	Hexagon 690	-	Hexagon 780

- Inference codebase: TFLite 2.8.0
- Inference precision: INT8

Results - Accuracy

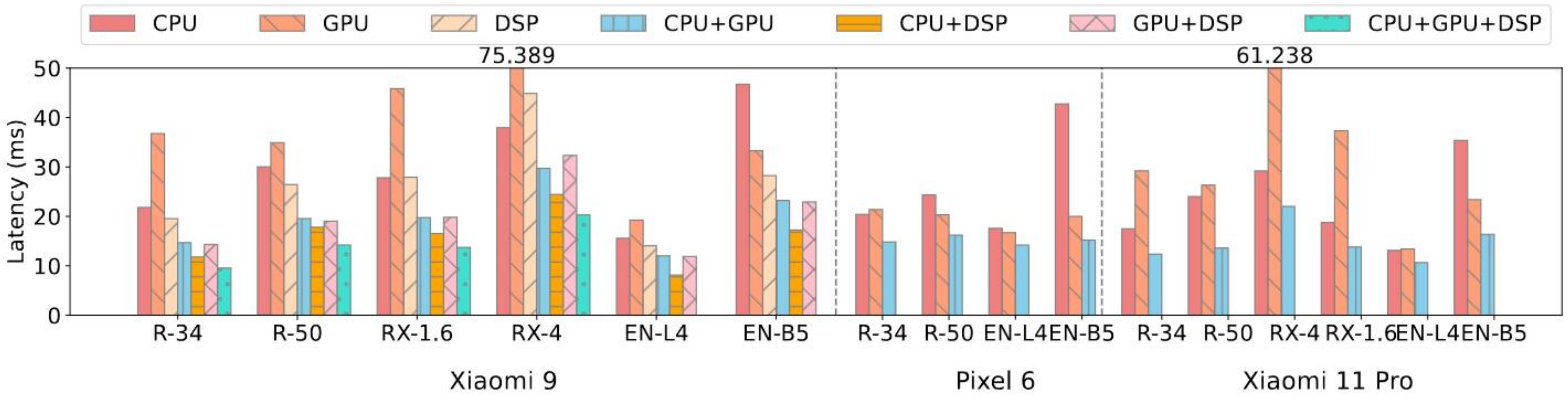
Branched models can maintain the original accuracy

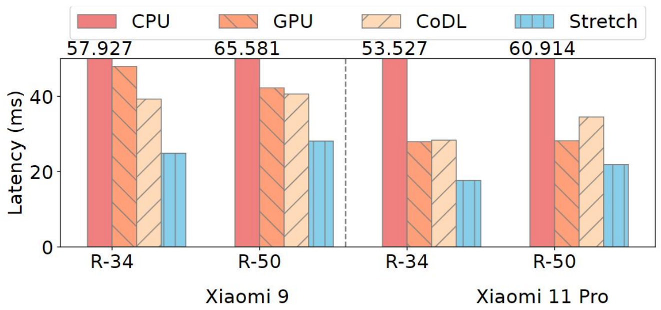
	R-34	R-50	RX-1.6	RX-4	EN-L4	EN-B5
Original	73.3	**76.7**	77	78.6	77.8	**78**
Mi9/C+G	72.8	76.2	76.9	78.4	**77.9**	77.8
Mi9/C+D	73.2	76.2	**77.5**	**78.9**	77.6	77.9
Mi9/G+D	73.3	76.6	76.9	78.4	77.5	77.8
Mi9/C+G+D	72.9	76.1	76.6	78.3	-	-
Pixel6/C+G	**73.6**	76.5	-	-	77.6	77.9
Mi11/C+G	72.8	76.2	76.9	78.4	**77.9**	77.8

Results - Latency

- The first work to enable flexible combination of different processors for inference
  - Concurrent run achieve 2.2x, 1.4x, 1.8x speedup compared to the fastest single processor

Latency comparison of different processor combinations

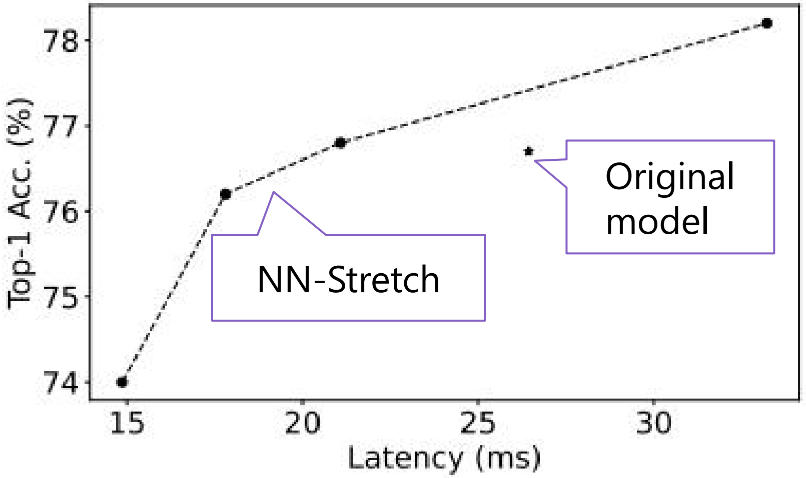




Missing bars are not supported by the inference framework

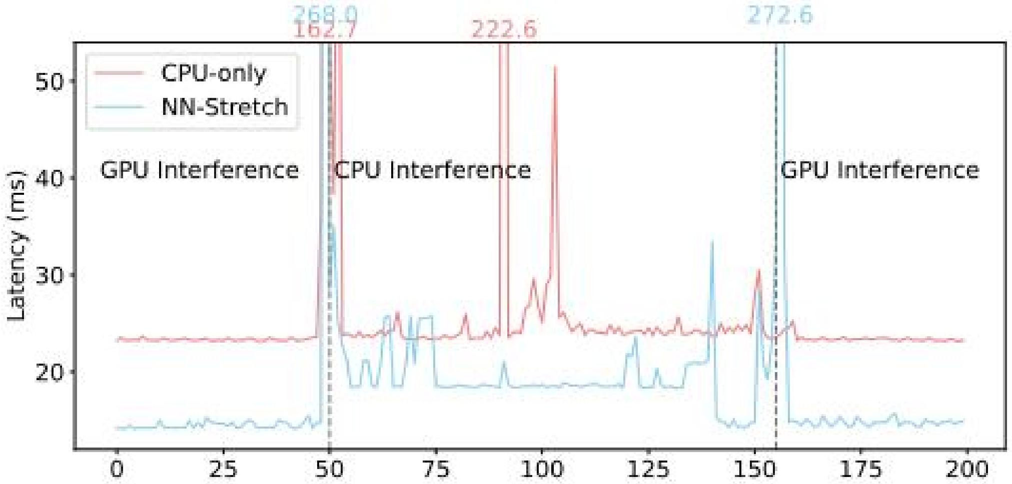
Results

- NN-Stretch generate models with different latency requirement



NN-Stretch improves the accuracy-latency tradeoff (marked as dots) compared to the original ResNet-50 on Mi9/C+D (marked as the star)

- NN-Stretch adjusts inference processors based on availability



Latency of ResNet-34 inferences on Mi9.
Interference run on GPU [0,50), CPU [50,155), and GPU [155,200). Baseline always runs on CPU. NN-Stretch runs on CPU+DSP, GPU+DSP, CPU+DSP

Summary

- NN-Stretch is a new model adaption paradigm, that enables concurrent inference by algorithm and system co-design.
- It improves inference performance and flexibility for diverse AI-applications on edge devices.
- Opensource link: GitHub - caoting-dotcom/multiBranchModel: Multi-branch model for concurrent execution

Thank you!