FROM CHIPS TO SYSTEMS — LEARN TODAY, CREATE TOMORROW

DEC 5 - 9, 2021 ◆ San Francisco, California

NAAS: Neural Accelerator Architecture Search

Yujun Lin, Mengtian Yang, Song Han

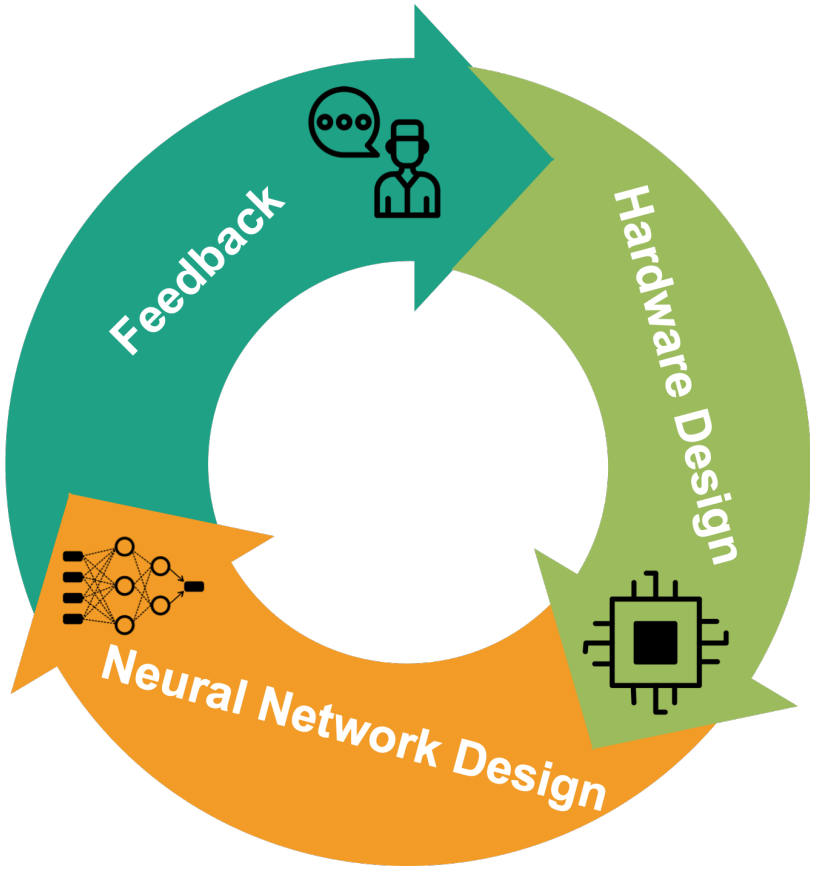
MIT

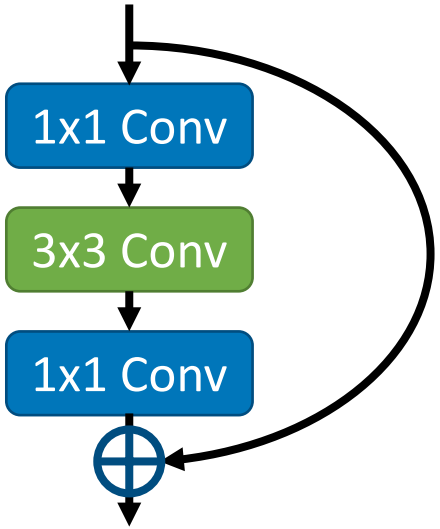
Bio

Yujun Lin is a Ph.D. student at MIT EECS, advised by Profesor Song Han. He received his M.S. degree from MIT in 2020 and B.Eng degree from Tsinghua University in 2018.

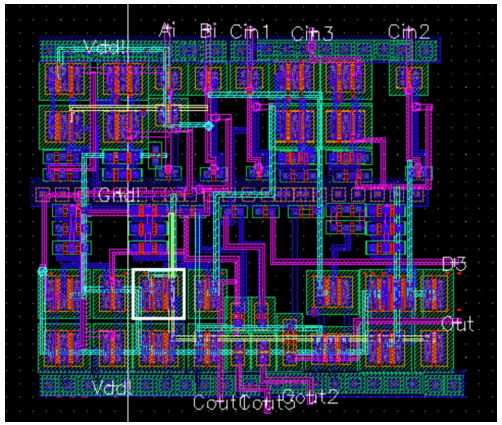
His research focuses on the intersection of efficient deep learning and accelerator architecture design.

Accelerating Deep Learning Computing





Neural Architecture



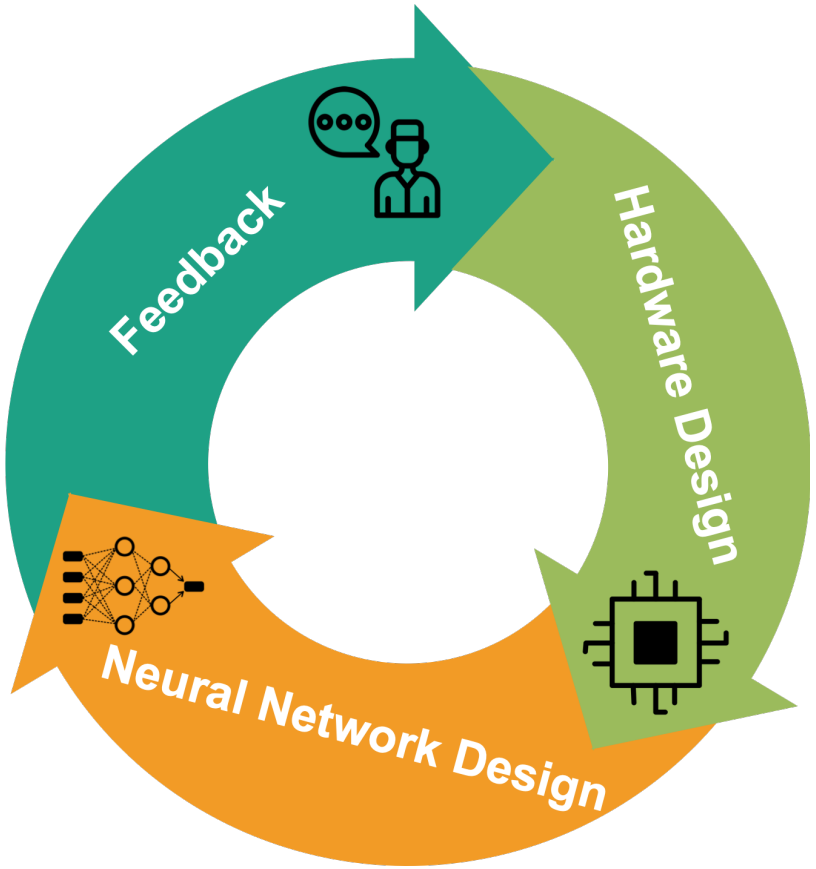
Accelerator Architecture

- Both neural architecture and accelerator architecture design are important to enable specialization and acceleration

Data Driven Approach is Desirable

- Given the huge design space, data-driven approach is desirable, where new architecture design evolves as new designs and rewards are collected
- Hardware-aware Neural Architecture Search (NAS) and auto compiler optimization (e.g., autoTVM)
  - Only focus on off-the-shelf hardware
  - Neglect the freedom in the hardware design space

Design Spaces



	Key Dimensions
Accelerator	Local Buffer Size, Global Buffer Size, #PEs
	Compute Array Size, PE Connectivity
Compiler	Loop Orders, Loop Tiling Size, Dataflow
Neural Network	#Layers, #Channels, Kernel Size, Bypass
	(Input / Weight) Quantization Precision

Design spaces are tightly entangled

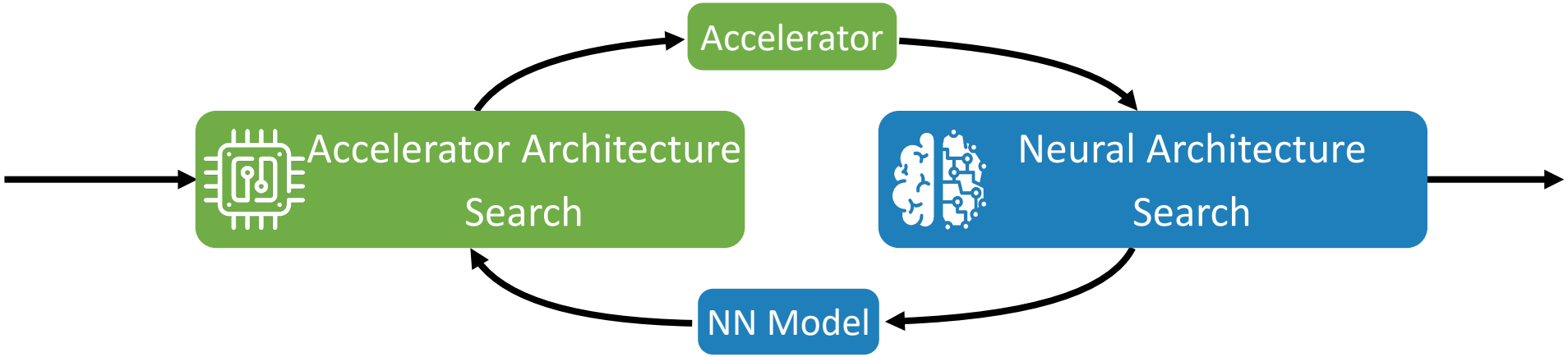
- Correlation between design spaces is complicated, and *varies from accelerator to accelerator*

Accelerator Parameter Space	Neural Architecture Search Space			
	# In Channels	# Out Channels	Kernel Size	Feature Map Size
Array #rows	○		●	
Array #cols		○		○
IBUF size	◎			◎
WBUF size	◎	◎	◎	
OBUF size		◎		◎

○ NVDLA ● Eyeriss

Joint Search Accelerator and Neural Network

- Searching accelerator and neural architecture in *one optimization loop* offers highly matched solutions



Architecture Design Spaces

	Key Dimensions	
Accelerator	Local Buffer Size, Global Buffer Size, #PEs	Architectural Sizing
	Compute Array Size, PE Connectivity	Connectivity Parameters
Compiler	Loop Orders, Loop Tiling Size, Dataflow	
Neural Network	#Layers, #Channels, Kernel Size, Bypass	
	(Input / Weight) Quantization Precision	

Architecture Design Spaces

	Key Dimensions	
Accelerator	Local Buffer Size, Global Buffer Size, #PEs	Architectural Sizing
	Compute Array Size, PE Connectivity	Connectivity Parameters
Compiler	**Loop Orders, Loop Tiling Size, Dataflow**	
Neural Network	#Layers, #Channels, Kernel Size, Bypass	
	(Input / Weight) Quantization Precision	

How to embed these design dimensions for searching?

Convolution Loop Nests

- Convolution loop nests can be divided into two parts: *temporal mapping* and *spatial parallelism*

Tensor Dimension	Notation
Batch	N
Output Channel	K
Input Channel	C
Input Row (Output Row)	Y (Y')
Input Column (Output Column)	X (X')
Kernel Row	R
Kernel Column	S

```
For _R in range(R / R):
 For _S in range(S / S):
  For _C in range(C / T_C):
   For _Y' in range(Y' / T_Y'):
    For _X' in range(X' / T_X'):
     For r in range(R):
      For s in range(S):
       For _k in range(K / 16):
        For _y' in range(T_Y'):
         For _x' in range(T_X'):
          For _c in range(T_C / 16):
           Parallel-For _m in range(16):
           Parallel-For _n in range(16):
            c = _C * T_C + _c * 16;
            k = _k * 16;
            y' = _Y' * T_Y' + _y';
            x' = _X' * T_X' + _x';
            y = y' + r - R;
            x = x' + s - S;
            psum[b,k,y',x'] += acts[b,x,y,x]
                               * wgts[k,c,y,x];
```

Mapping

HW

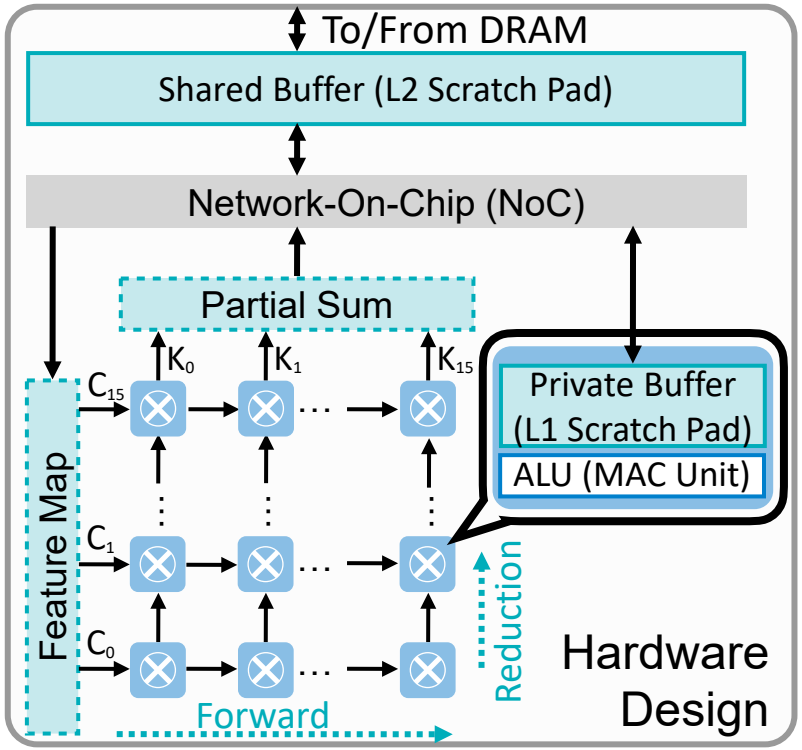
Loop Tiling

Loop Order

Hardware Parallelism

From Computation Loops To Hardware

- Spatial parallelism determines the PE connectivity
  - *e.g.*, C (in channels) indicates reduction of partial sum registers
  - *e.g.*, K (out channels) indicates forward of input feature registers

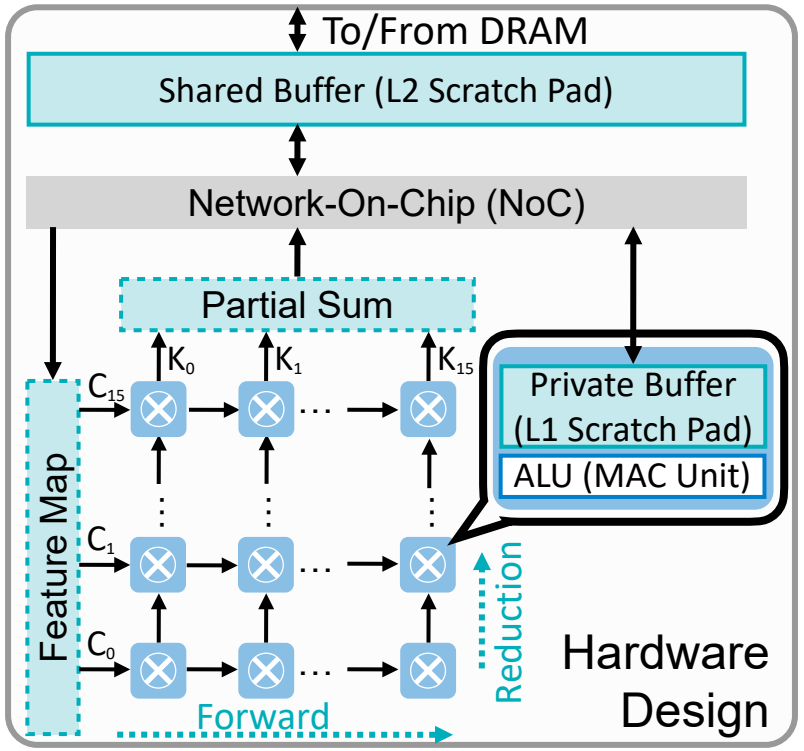



```
For _R in range(R / R):
 For _S in range(S / S):
  For _C in range(C / T_C):
   For _Y' in range(Y' / T_Y'):
    For _X' in range(X' / T_X'):
     For r in range(R):
      For s in range(S):
       For _k in range(K / 16):
        For _y' in range(T_Y'):
         For _x' in range(T_X'):
          For _c in range(T_C / 16):
           Parallel-For _m in range(16):
           Parallel-For _n in range(16):
            c = _C * T_C + _c * 16;
            k = _k * 16;
            y' = _Y' * T_Y' + _y';
            x' = _X' * T_X' + _x';
            y = y' + r - R;
            x = x' + s - S;
            psum[b,k,y',x'] += acts[b,x,y,x]
                               * wgts[k,c,y,x];
```

Mapping

HW

Encoding Accelerator and Mappings



Hardware Description

Architectural Sizing

- L2 Buffer Size
- L1 Buffer Size
- Number of PEs
- Bandwidth

Connectivity Parameters

- #Array Dimension
- Array Dim. Sizes
- PE Connection (Parallel Dim)

```
For _R in range(R / R):
 For _S in range(S / S):
  For _C in range(C / T_C):
   For _Y' in range(Y' / T_Y'):
    For _X' in range(X' / T_X'):
     For r in range(R):
      For s in range(S):
       For _k in range(K / 16):
        For _y' in range(T_Y'):
         For _x' in range(T_X'):
          For _c in range(T_C / 16):
           Parallel-For _m in range(16):
           Parallel-For _n in range(16):
            c = _C * T_C + _c * 16;
            k = _k * 16;
            y' = _Y' * T_Y' + _y';
            x' = _X' * T_X' + _x';
            y = y' + r - R;
            x = x' + s - S;
            psum[b,k,y',x'] += acts[b,x,y,x]
                               * wgts[k,c,y,x];
```

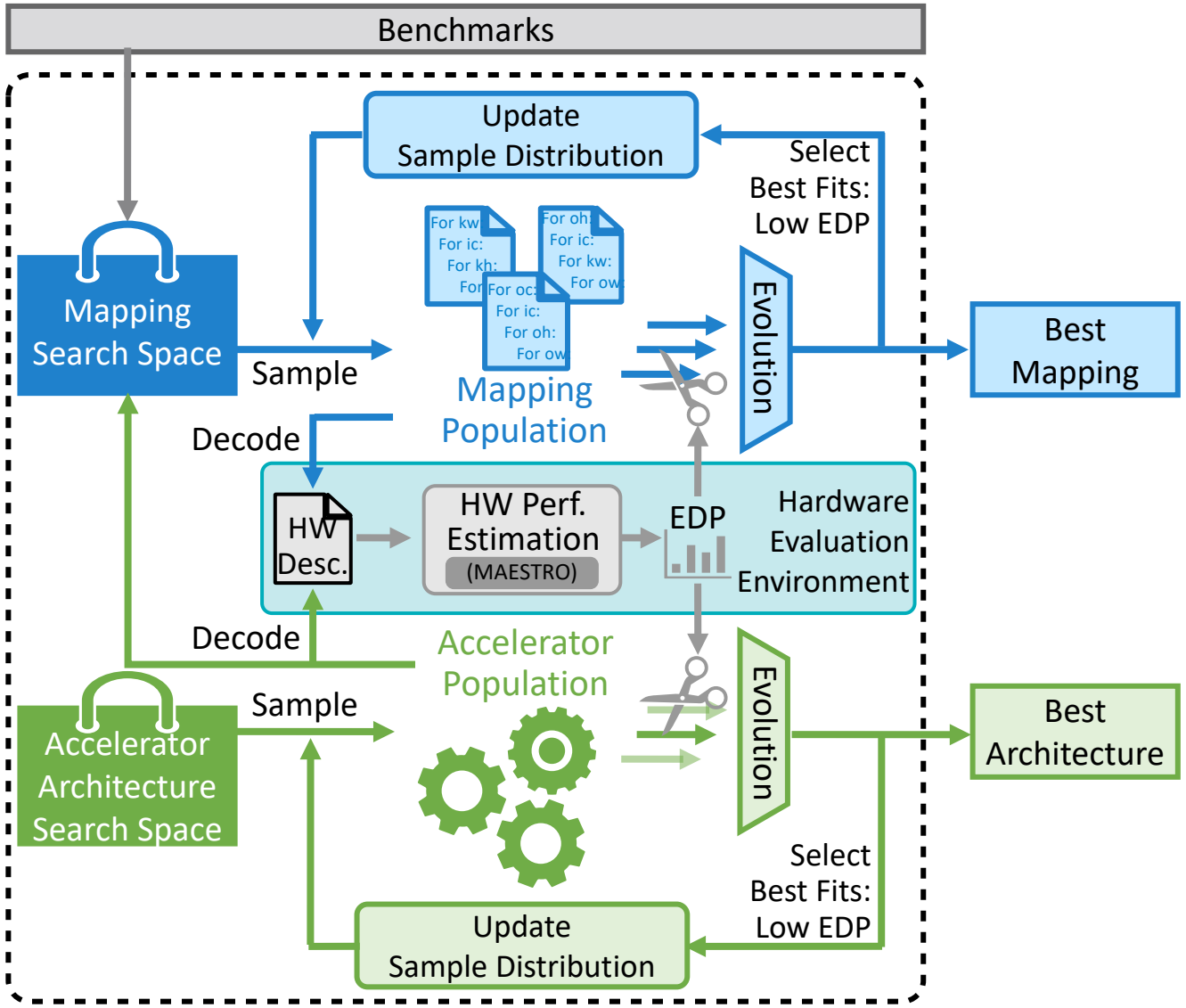
Hardware Encoding Vector

Architectural Sizing				Connectivity Parameters					
L2 Buffer Size	L1 Buffer Size	#PEs	Band-width	#Dim		Dim Sizes		Parallel Dims	

Mapping Encoding Vector

Array Level			PE Level	
Loop Orders		Tiling Sizes	Loop Orders	

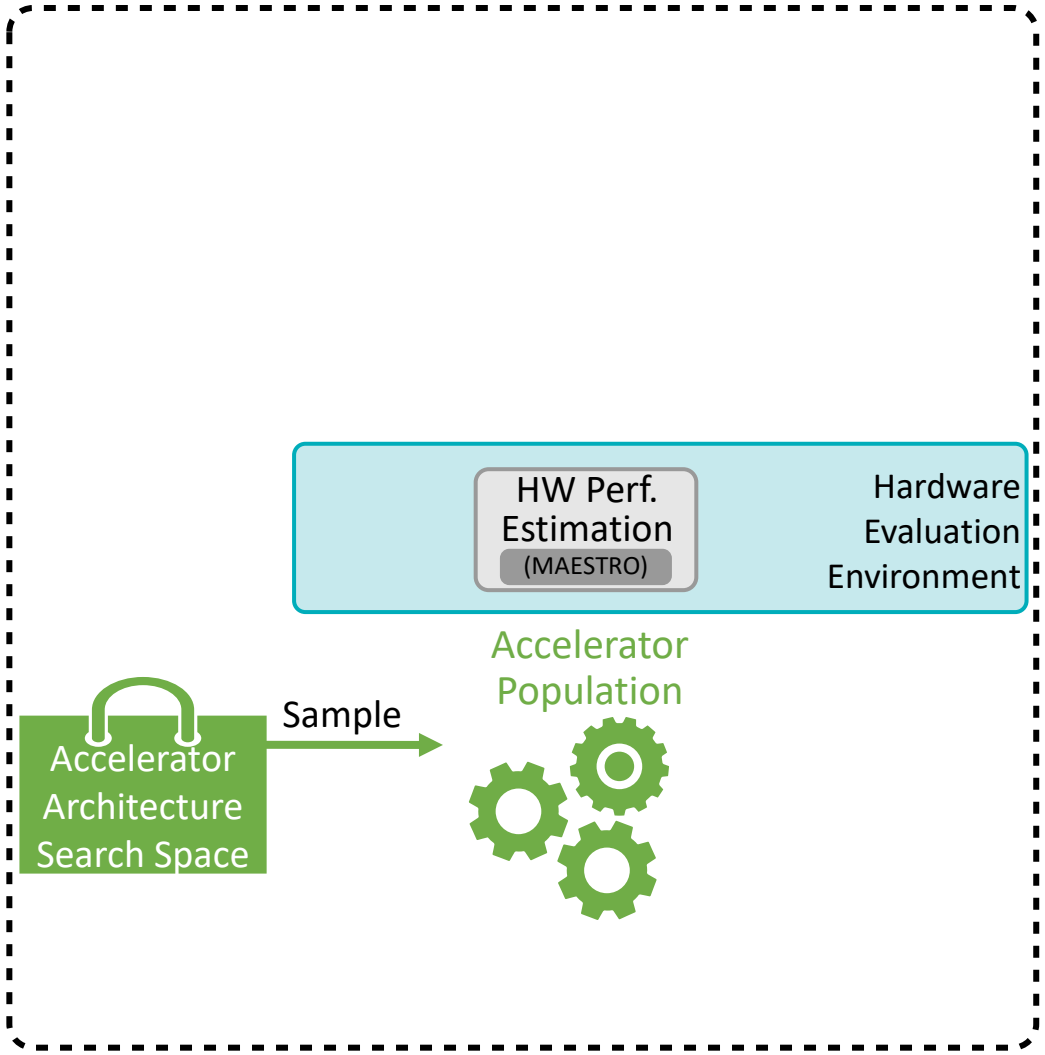
Neural Accelerator Architecture Search



Neural Accelerator Architecture Search

Accelerator Architecture Search

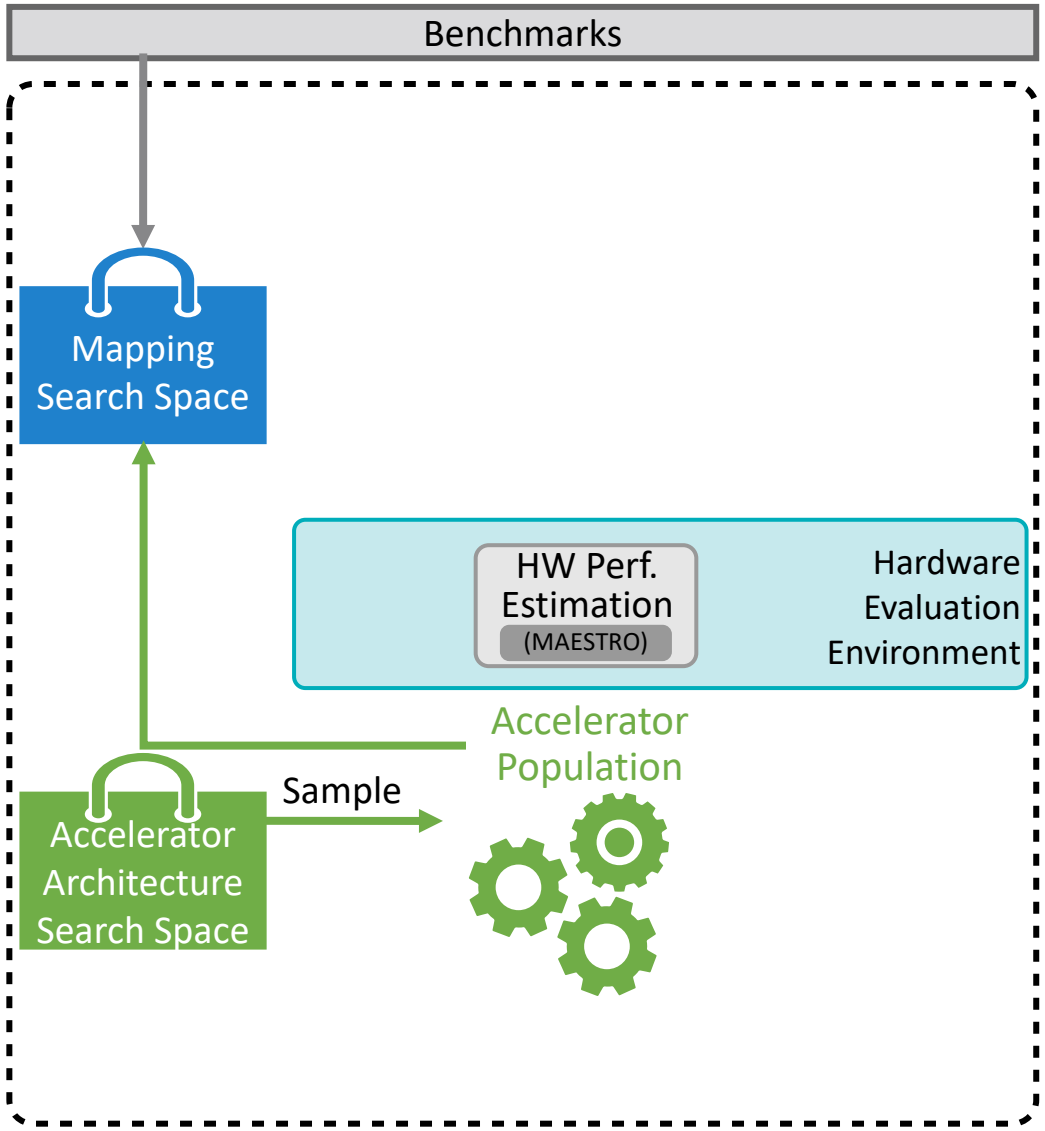
1. *Random sample* accelerator candidates based on multivariate normal distribution $N(\mu_A, \sigma_A, \Sigma_A)$



Neural Accelerator Architecture Search

Compiler Mapping Search

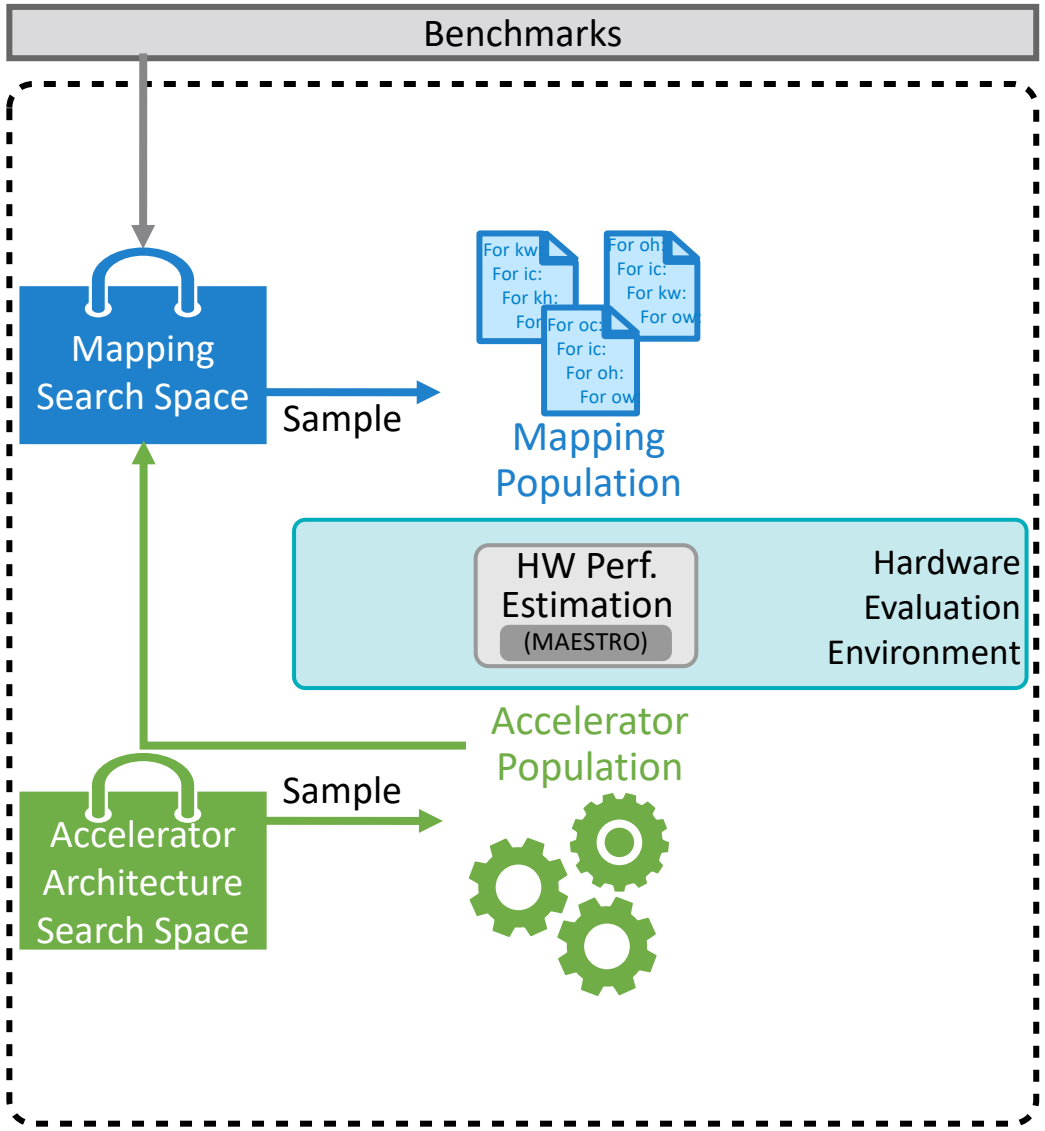
1. Determine mapping space for each accelerator candidates from NN benchmarks



Neural Accelerator Architecture Search

Compiler Mapping Search

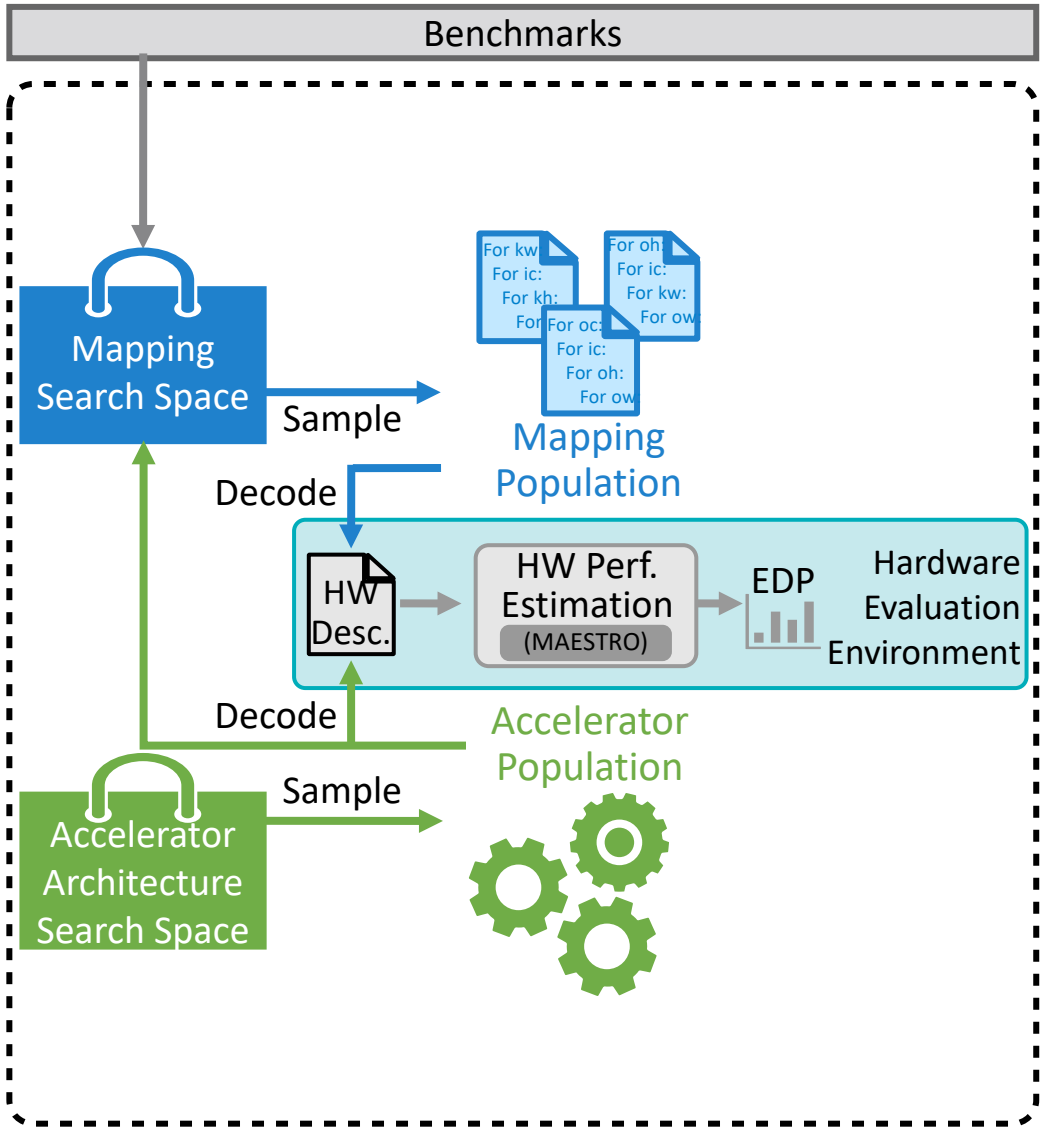
2. *Random sample* mapping candidates based on multivariate normal distribution $N(\mu_M, \sigma_M, \Sigma_M)$



Neural Accelerator Architecture Search

Compiler Mapping Search

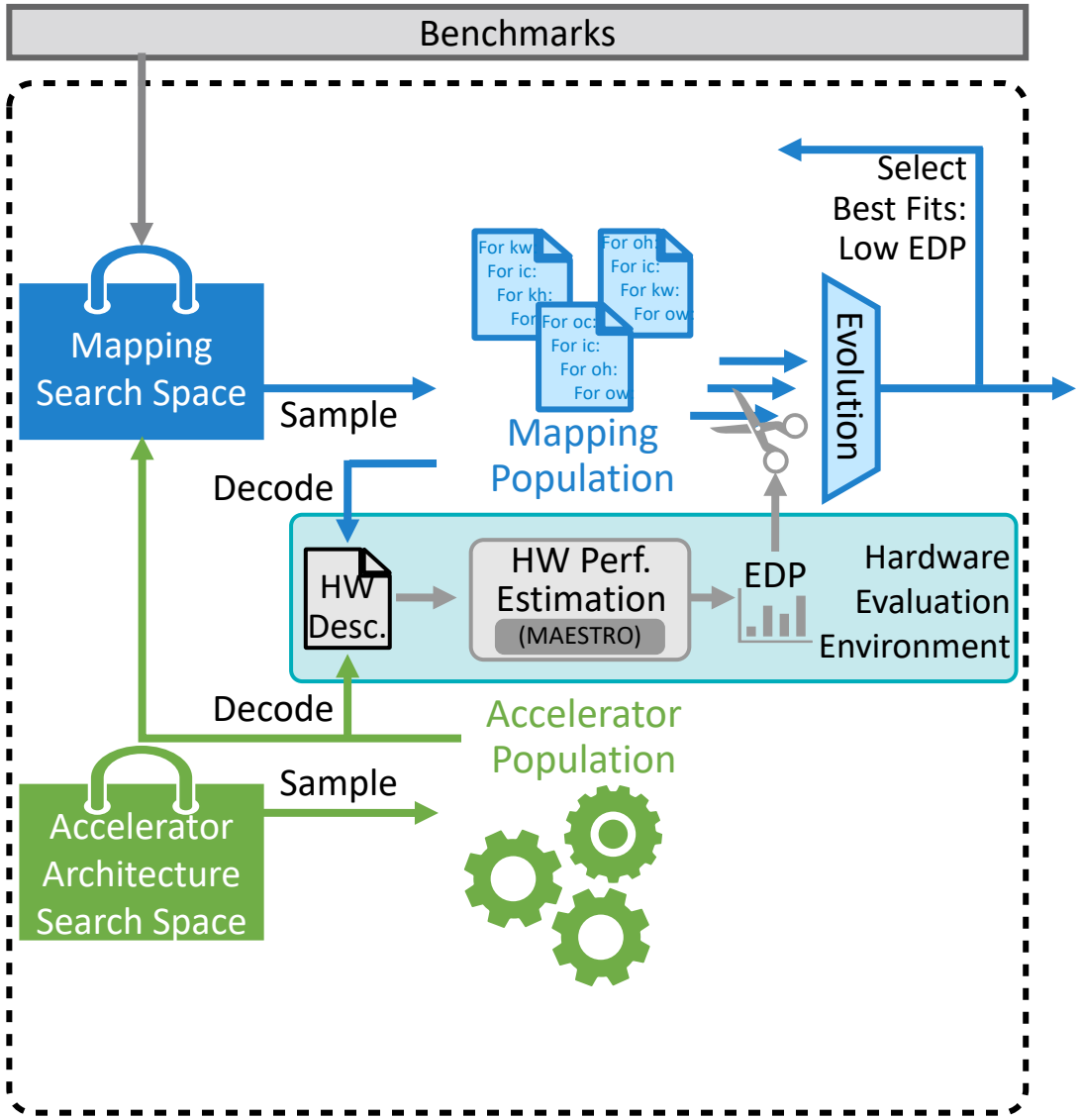
3. Decode encoding vectors to hardware description;
4. *Evaluate* Energy-Delay-Product (EDP) for *each pair of* accelerator candidate and its mapping candidate



Neural Accelerator Architecture Search

Compiler Mapping Search

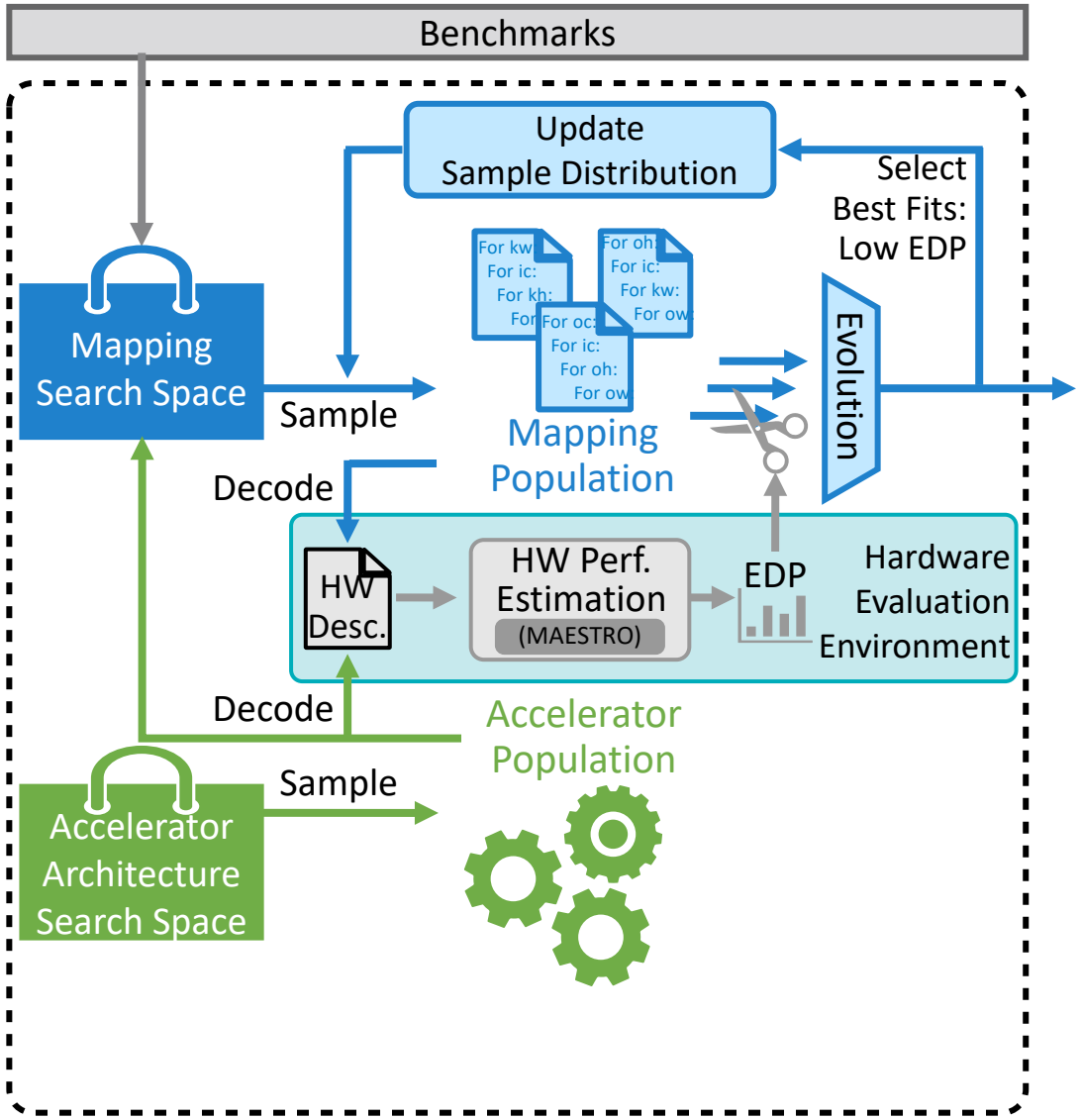
5. *Select best fits* with lowest EDP



Neural Accelerator Architecture Search

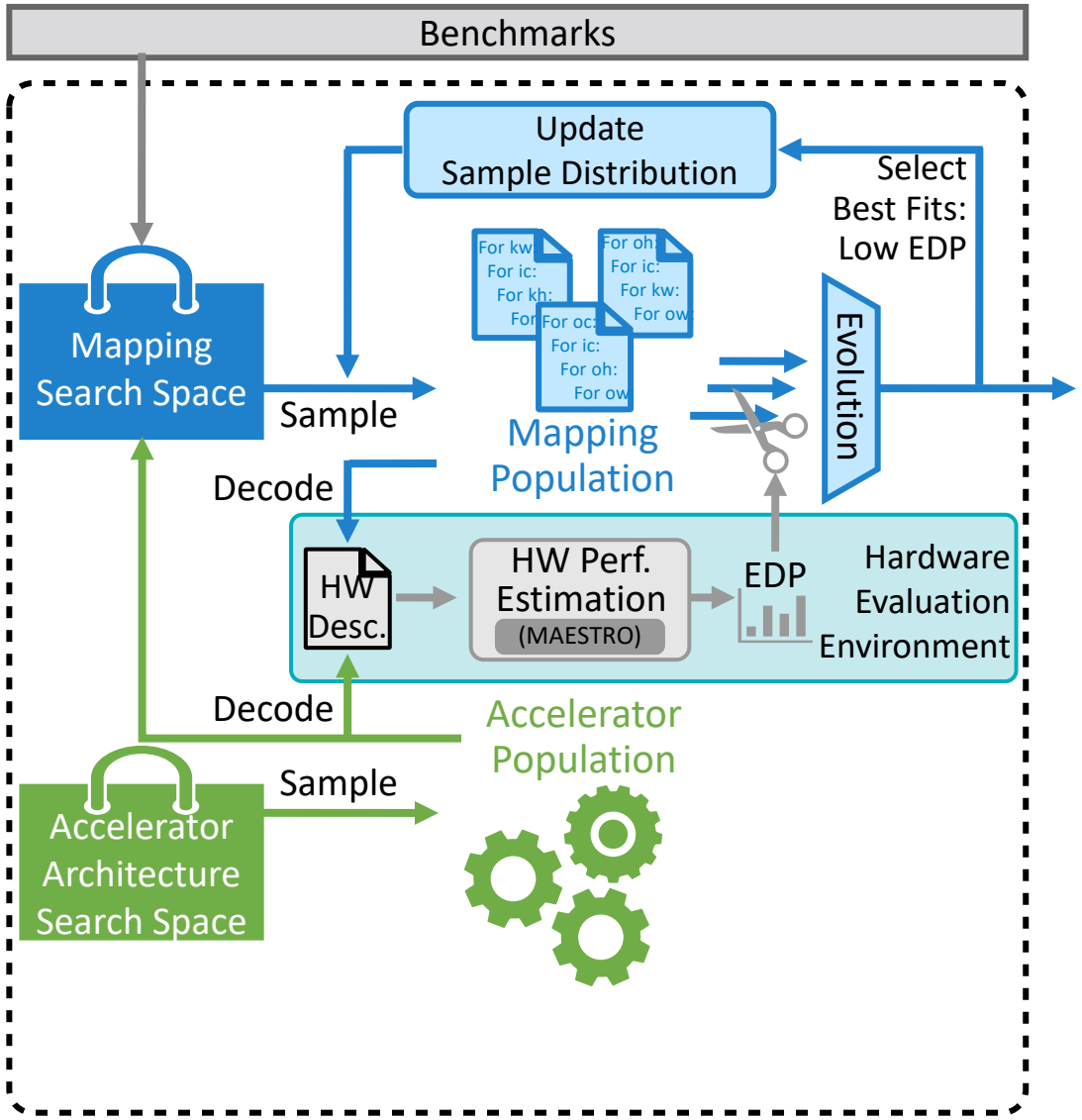
Compiler Mapping Search

5. Update $\mu_M, \sigma_M, \Sigma_M$ to *increase the likelihood* around best fits

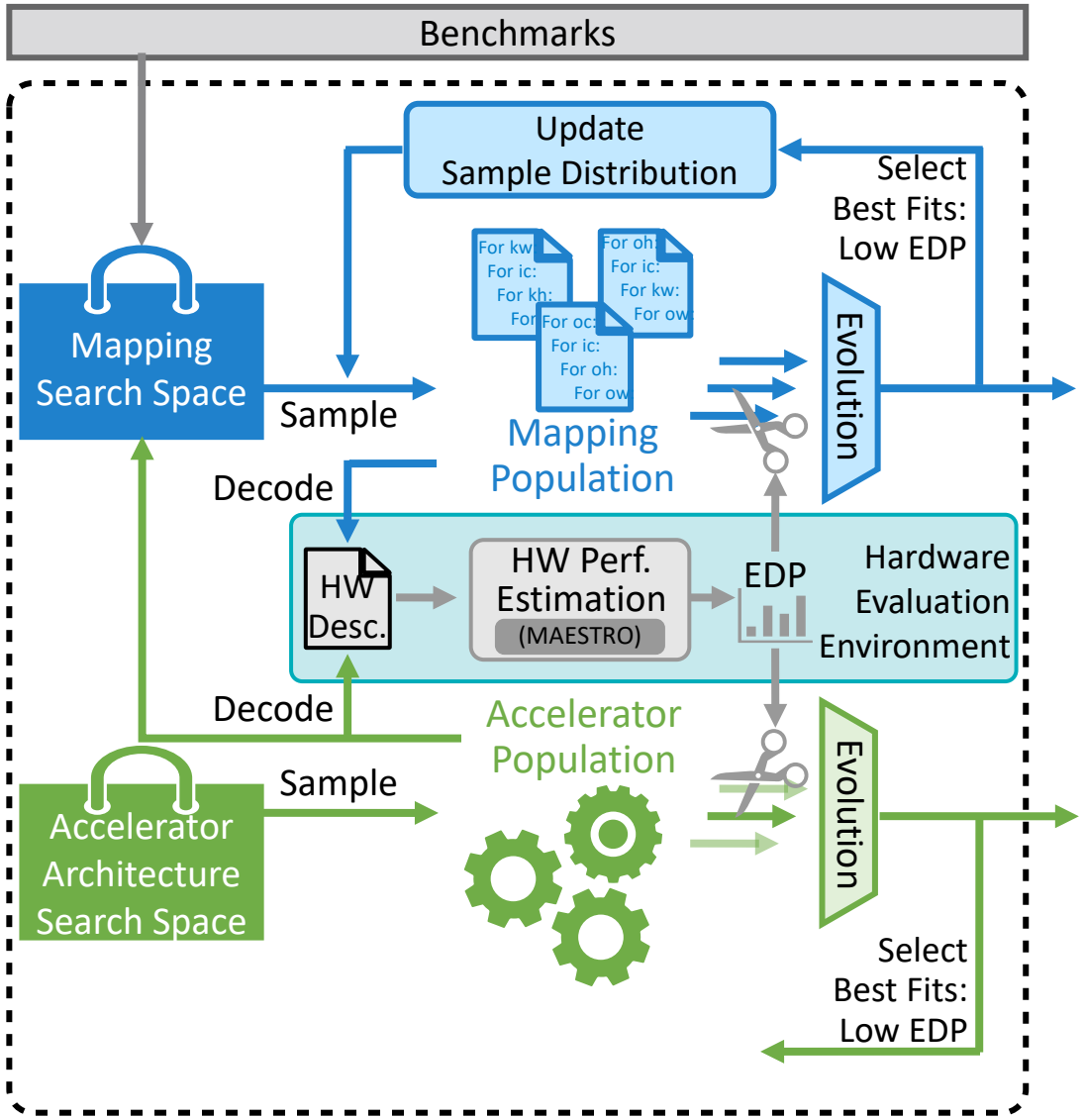


Neural Accelerator Architecture Search

Compiler Mapping Search
(iteratively optimizing mappings)



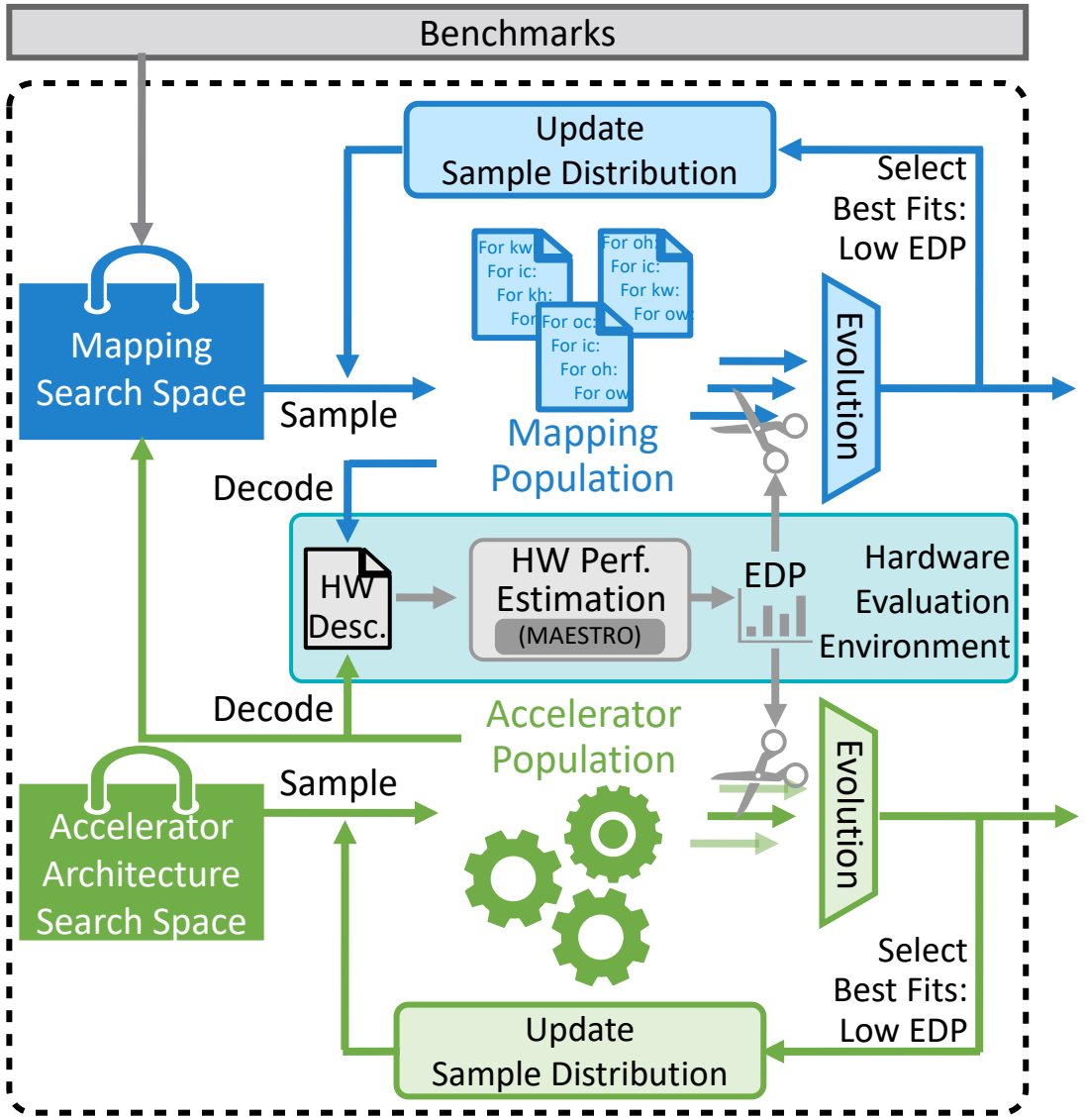
Neural Accelerator Architecture Search



Accelerator Architecture Search

2. Select best fits based EDP using corresponding searched mappings

Neural Accelerator Architecture Search



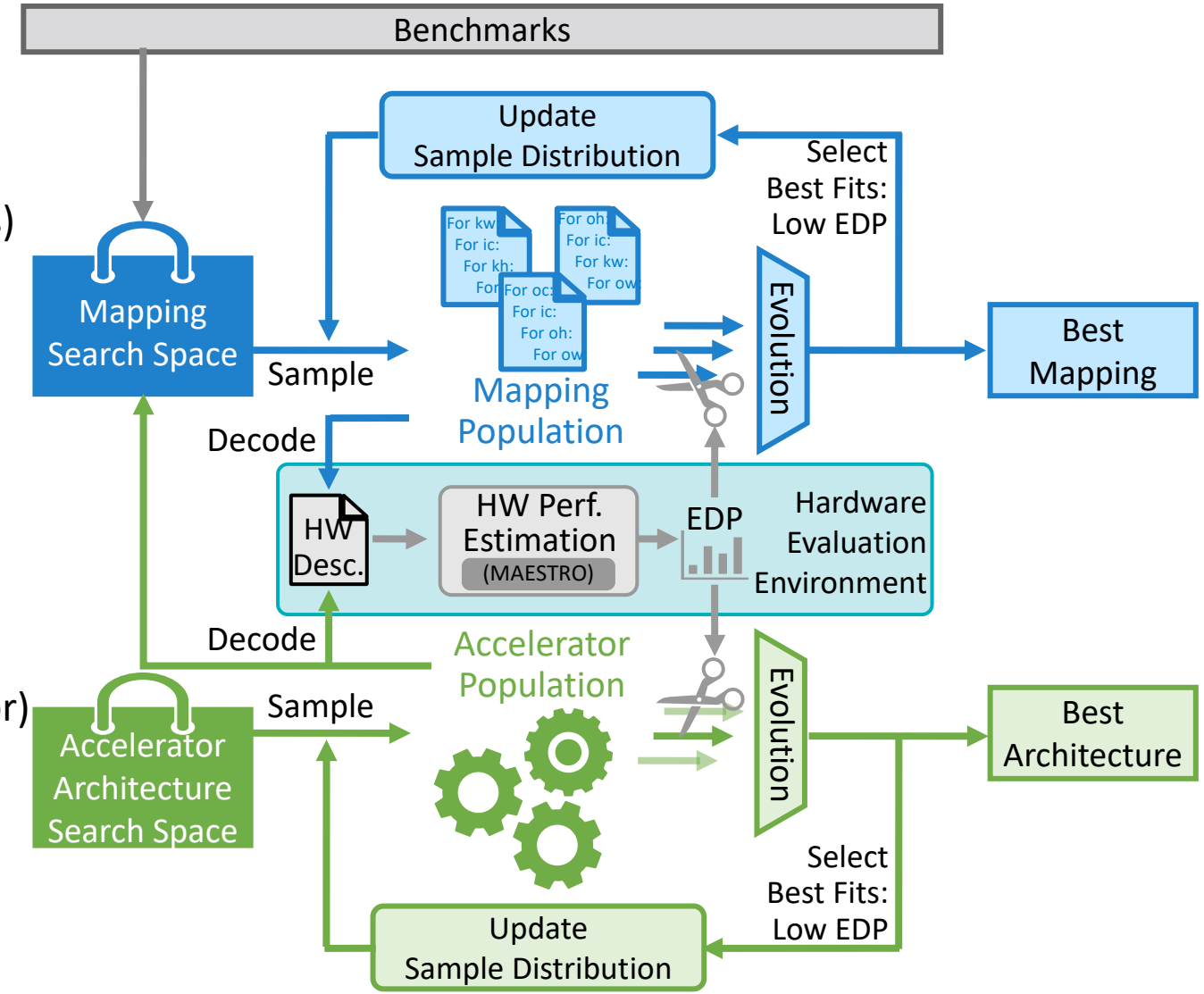
Accelerator Architecture Search

3. Update $\mu_A, \sigma_A, \Sigma_A$ to *increase the likelihood* around best fits

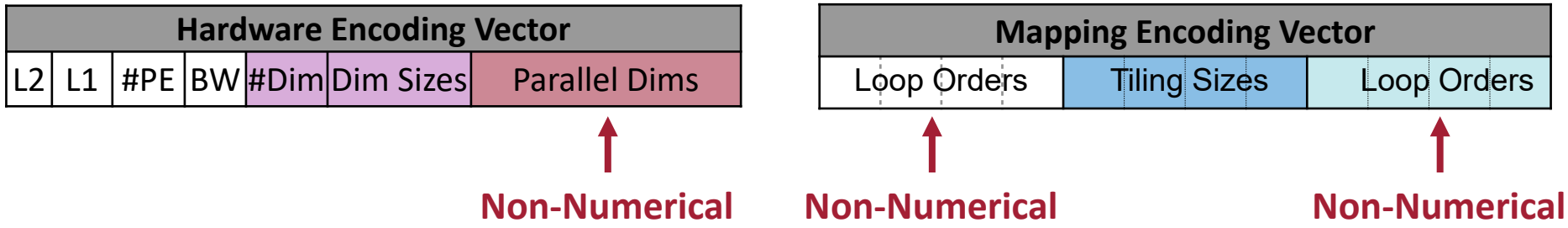
Neural Accelerator Architecture Search

Compiler Mapping Search
(iteratively optimizing mappings)

Accelerator Architecture Search
(iteratively optimizing accelerator)



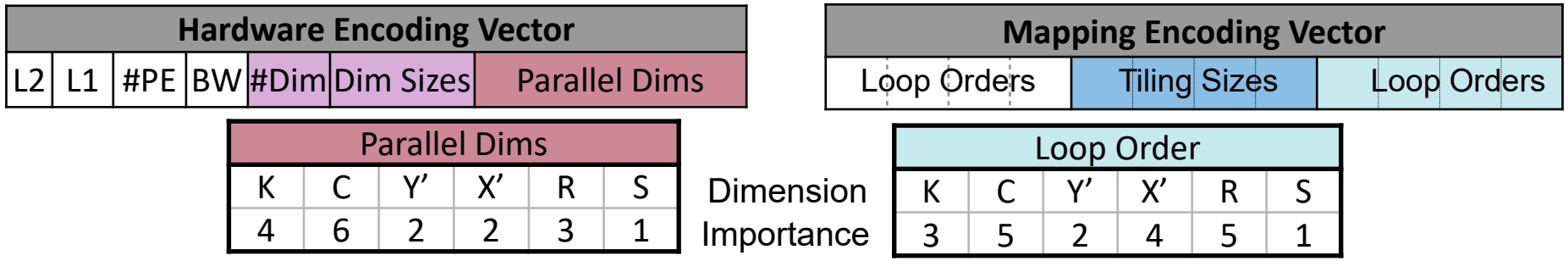
Encoding Non-numerical Parameters



- Index-based Encoding
  - Increment/Decrement of index value does not convey any physical information

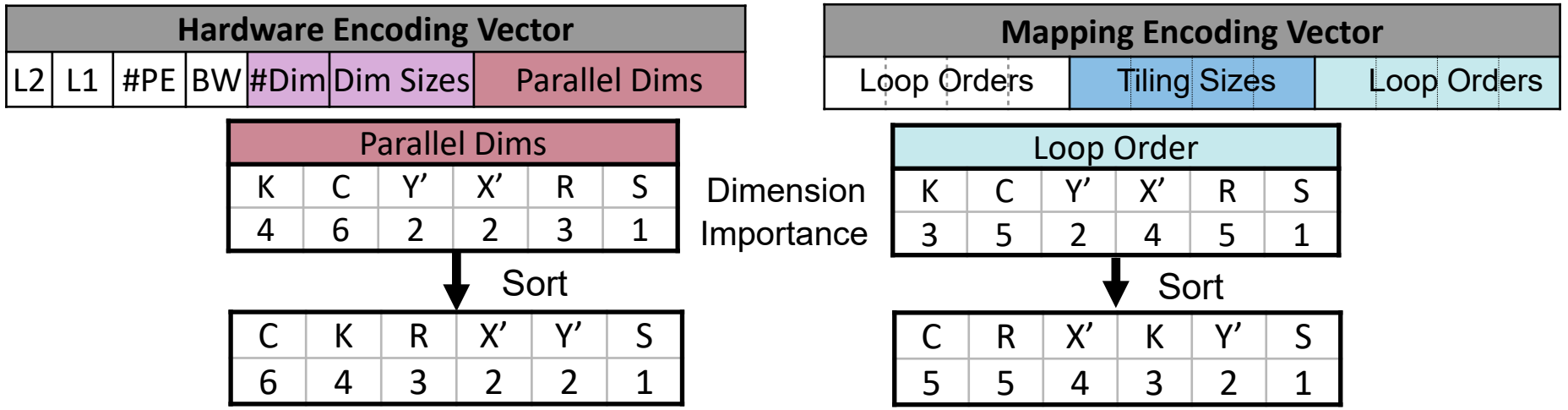
Non-Numerical Parameter Loop Orders	Numerical Encoding Value Index
CRXKYS	0
CXYRSK	1
…	…

Encoding Non-numerical Parameters



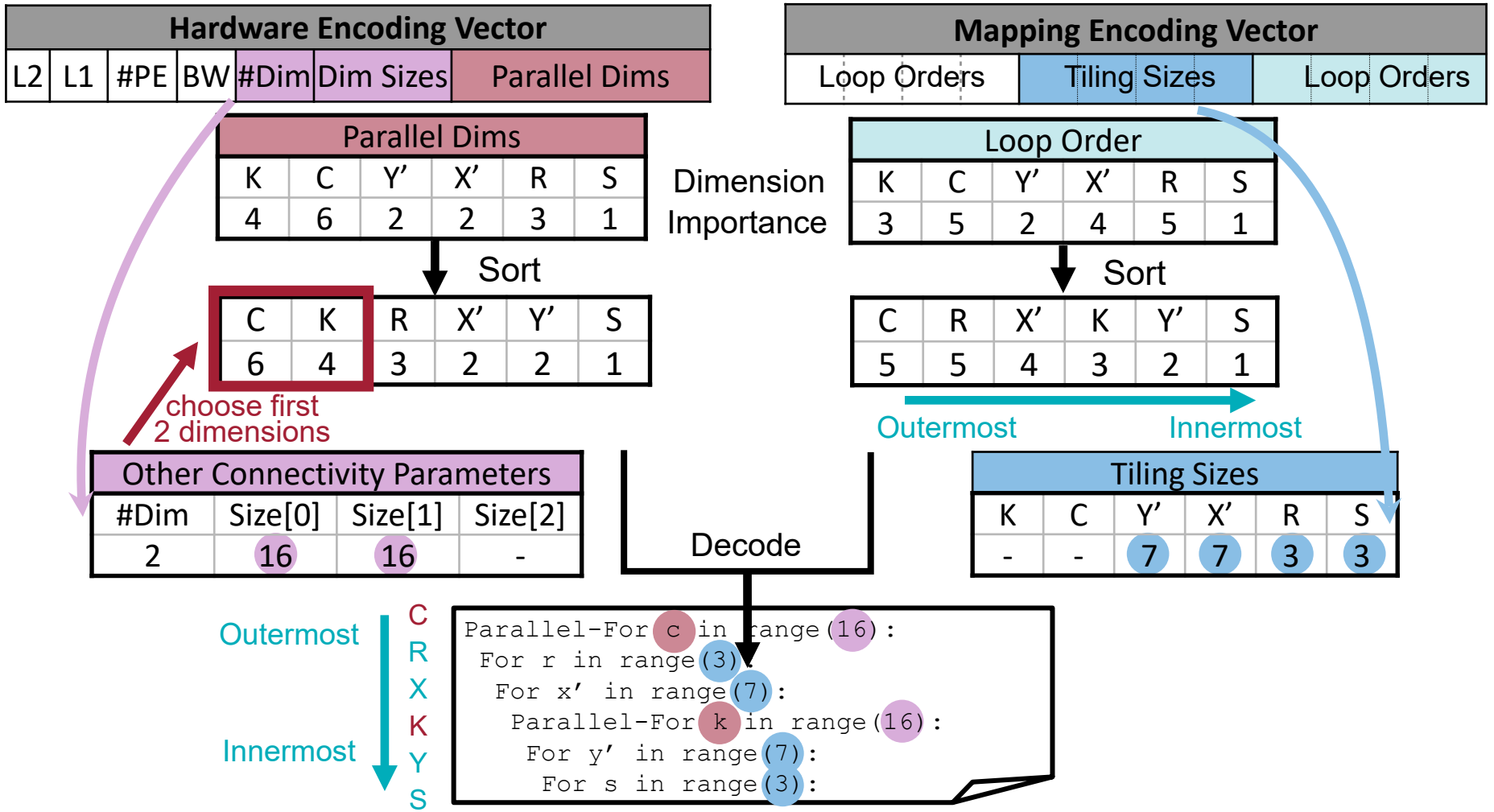
- Importance-based Encoding
  1. Fix the dimension position in the encoding vectors
  2. Optimizer assigns numerical *importance* to these dimensions
     - by random sampling based on multivariate normal distribution, the same as other numerical parameters such as array sizes

Encoding Non-numerical Parameters



- Importance-based Encoding
  3. *Sort* the dimensions by the importance value in decreasing order

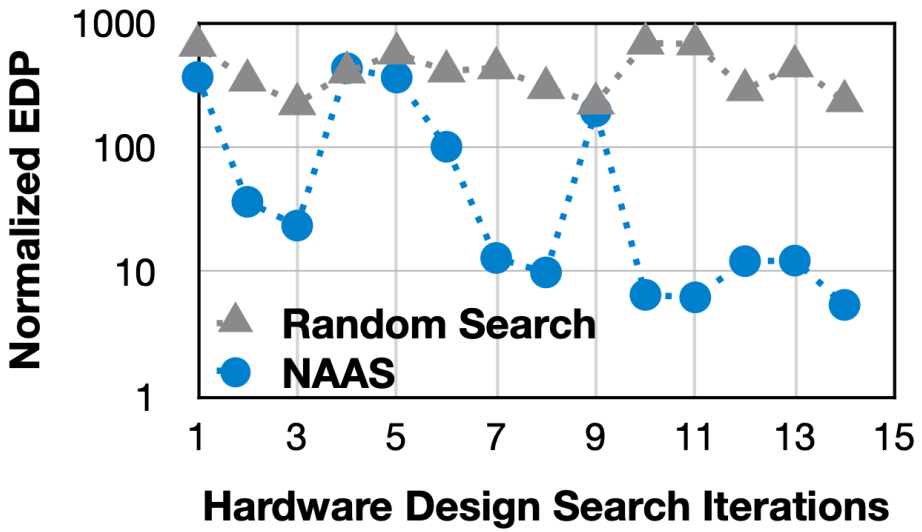
Encoding Non-numerical Parameters



Evaluation

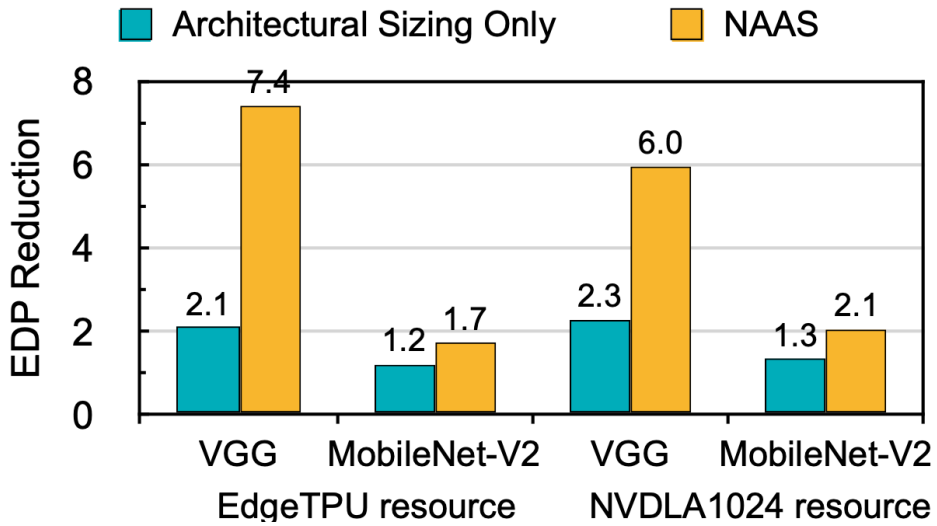
- Design Spaces of NAAS
  - 4 resource constraints: EdgeTPU, NVDLA, Eyeriss, ShiDianNao
  - NAAS searches #PEs at stride of 8, buffer sizes at stride of 16B, array sizes at stride of 2
- CNN Benchmarks
  - Classic large-scale networks: VGG16, ResNet50, UNet
  - Light-weight mobile networks: MobileNetV2, SqueezeNet, MNasNet
- Evaluation Settings
  - Large-scale NN with more hardware resources (EdgeTPU, NVDLA with 1024 PEs)
  - Light-weight NN with limited hardware resources (ShiDianNao, Eyeriss, NVDLA with 256 PEs)

Learning Curves



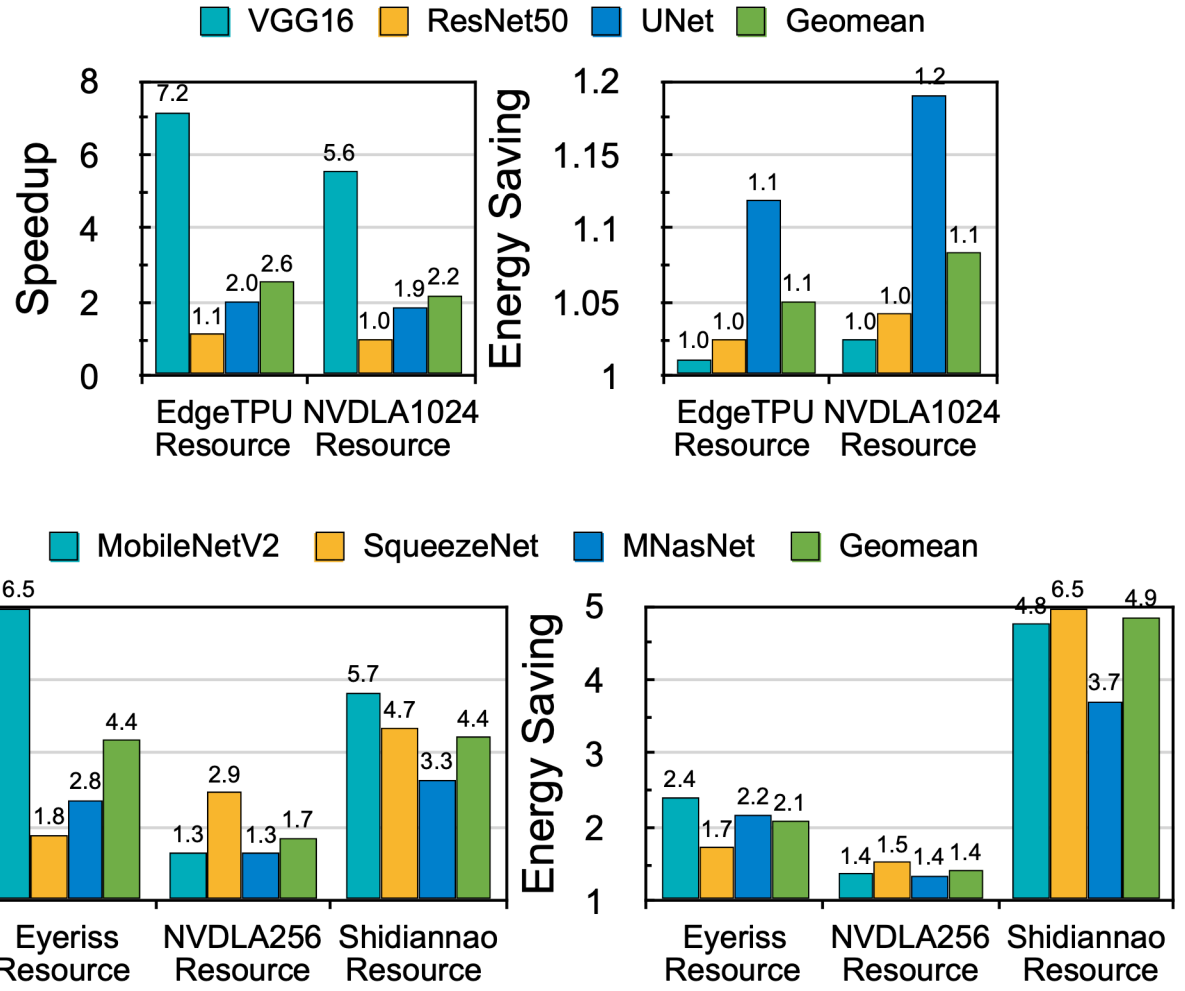
- As the optimization continues, the EDP mean of NAAS candidates decreases.
- NAAS gradually improves the range of hardware selections.

Search Beyond Architecture Sizing

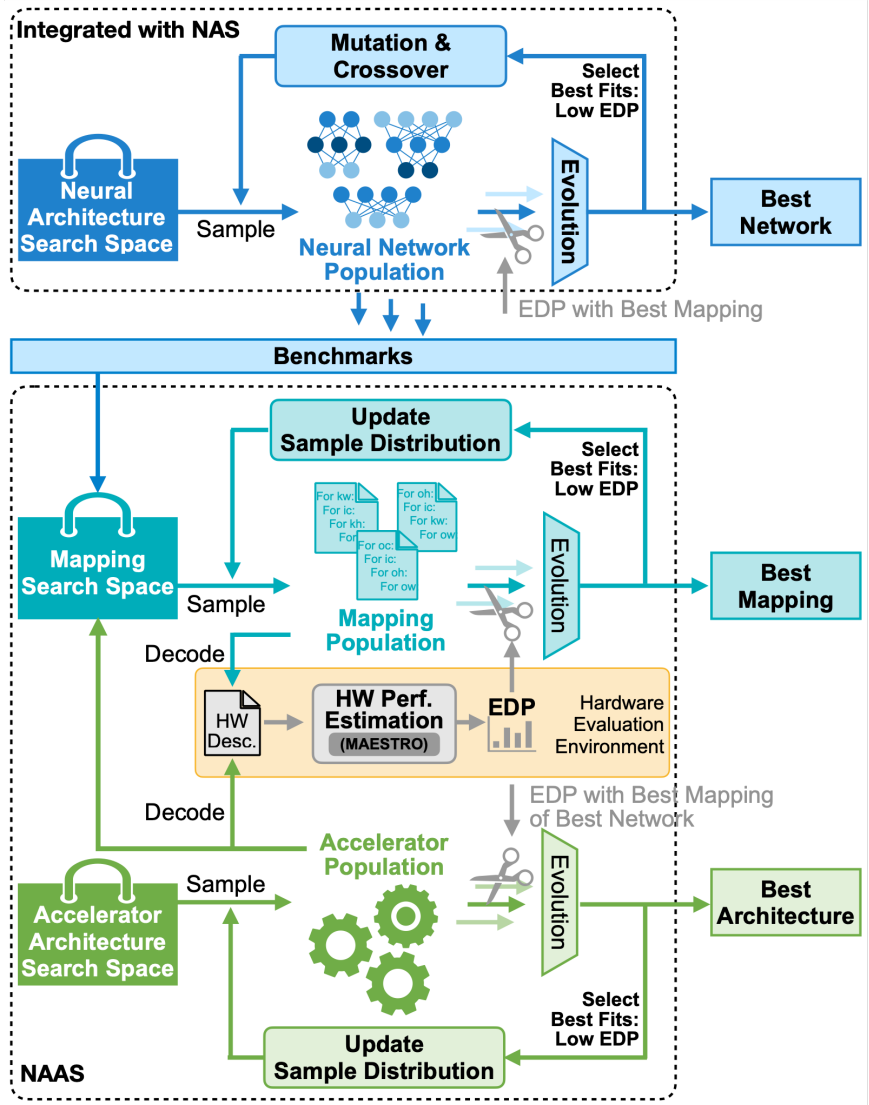


- Compared to searching the architectural sizing only (e.g., NASAIC, NHAS), searching the connectivity parameters and mapping strategies as well achieves considerable EDP reduction.

NAAS offers better solution than baseline



Jointly Optimize NN, Mapping, Accelerator

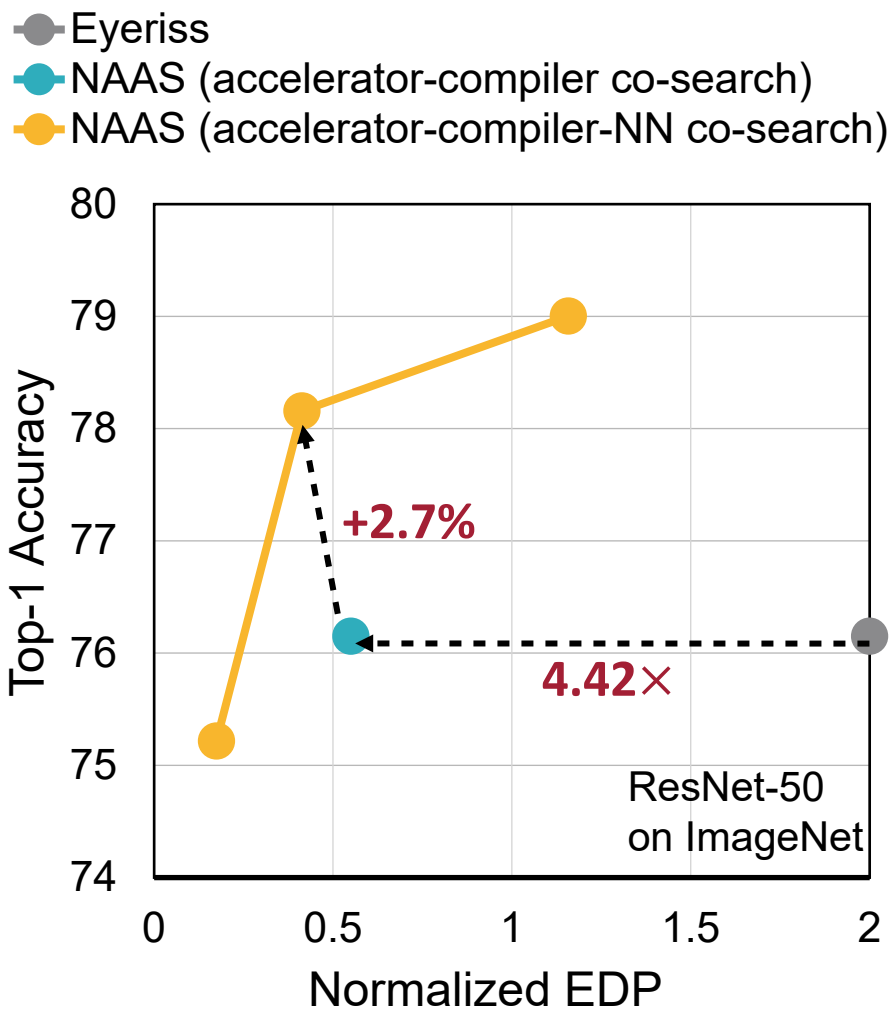


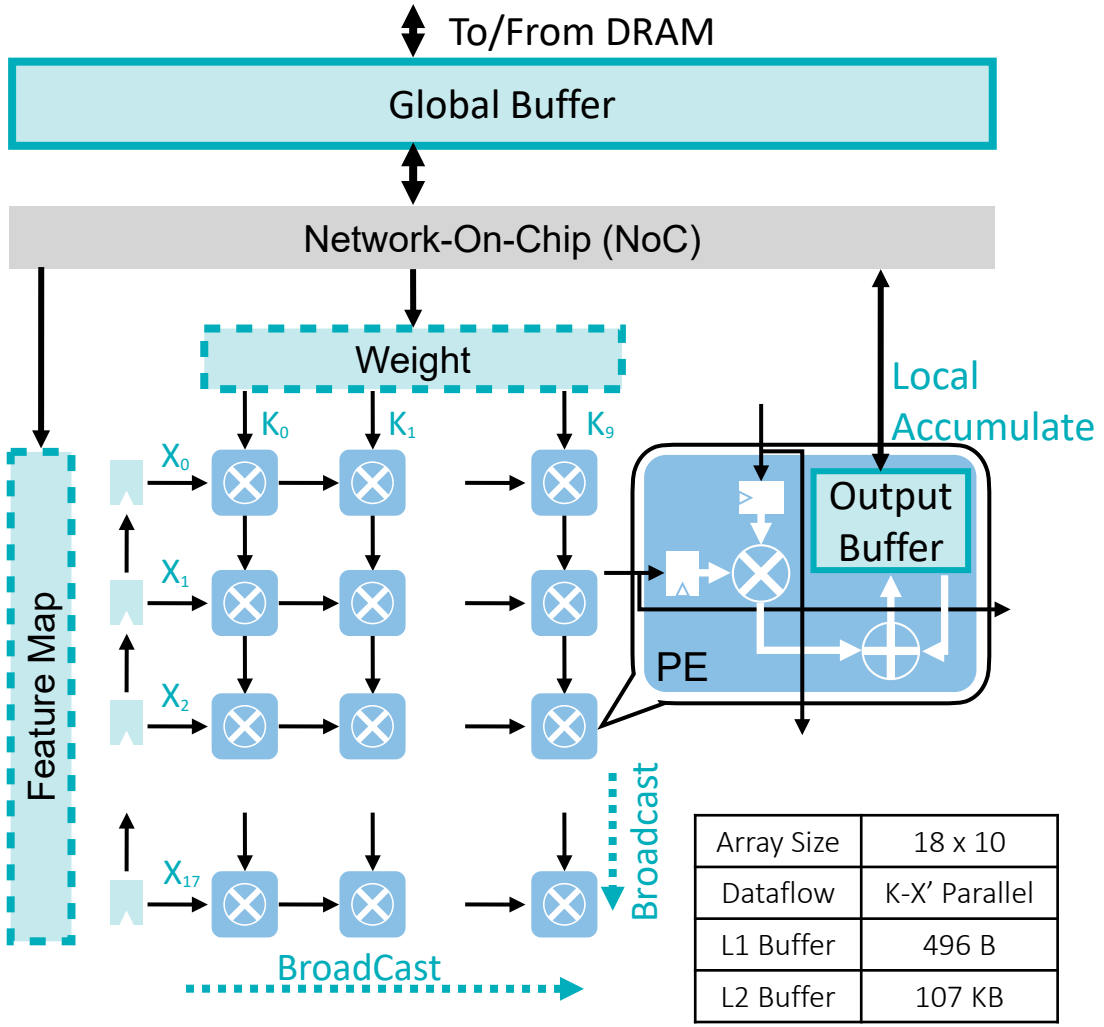

```
For epoch_naas in range(max_naas_epochs):
  accelerators = NAAS_generate_hardware()
  For hw in accelerators:
    For epoch_ofa in range(max_ofa_epochs):
      networks = OFA_generate_networks(accuracy)
      For nn in networks:
        map = NAAS_optimize_mappings(hw, nn)
        edp = NAAS_get_edp(hw, nn, map)
        OFA_update_optimizer(nn, edp)
        best_nn, best_map, best_edp = OFA_update_best(nn, map, edp)
    NAAS_update_optimizer(hw, best_nn, best_map, best_edp)
```

Evaluation

- Design Space of NAS
  - Once-For-All ResNet NAS
  - 3 width multiplier choices: 0.65, 0.8, 1.0
  - 18 residual blocks at maximum
  - 3 reduction ratios in each residual block: 0.2, 0.25, 0.35
  - Input image size ranges from 128 to 256 at strid of 16

Top-1 Accuracy vs. Normalized EDP





Array Size	18 x 10
Dataflow	K-X' Parallel
L1 Buffer	496 B
L2 Buffer	107 KB

Compared to NASAIC

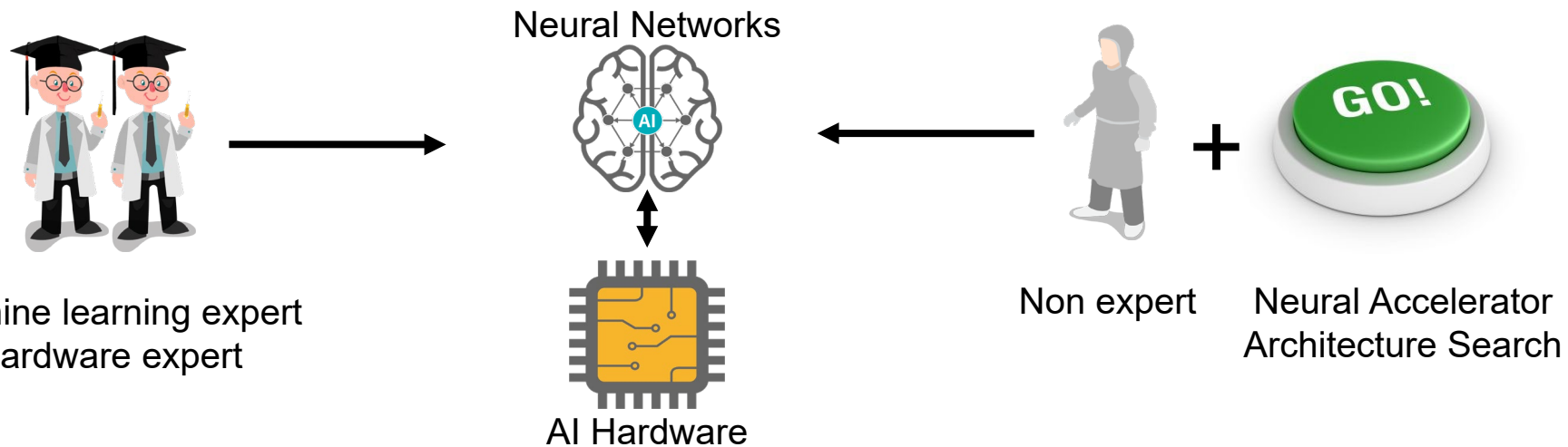
Search Approach	Arch	Cifar-10 Accuracy	Latency (cycles)	Energy (nJ)	EDP (cycles-nJ)
NASAIC	NVDLA	93.2	3e5	1e9	3e14
	ShiDianNao	91.1			
NAAS	NVDLA	93.2	8e4	2e9	2e14

Search Approach	Co-Search Cost (Gds)	NN Training Cost (Gds)	Total Cost (Gds)	AWS Cost	CO2 Emission
NASAIC	6000N	16 N	6000N	$ 441, 000N	41, 000N lbs
NHAS	12+4N	16 N	12+20N	$ 1, 500N	150N lbs
NAAS	<0.25N	50	< 50 + 0.25N	< $ 18N	< 2N lbs

- Gds: GPU days. N: the number of deployment scenarios.
- AWS cost $75/Gd, CO2 emission is 7.5 lbs/Gd.

Neural Accelerator Architecture Search

- Design spaces of hardware, compiler, and neural networks are tightly entangled, joint-optimization is better than separate optimization.
- Optimize both numerical parameters and non-numerical parameters, such as PE connectivity and loop order. Importance-based encoding helps optimize non-numerical parameters.



https://tinyml.mit.edu