# 兼具高效能與節能的 Edge/Device AI 晶片技術

## ITRI deep Learning Accelerator design, system, tools, and applications

羅賢君 Shien-Chun Luo

工業技術研究院 Industrial Technology Research Institute (ITRI)

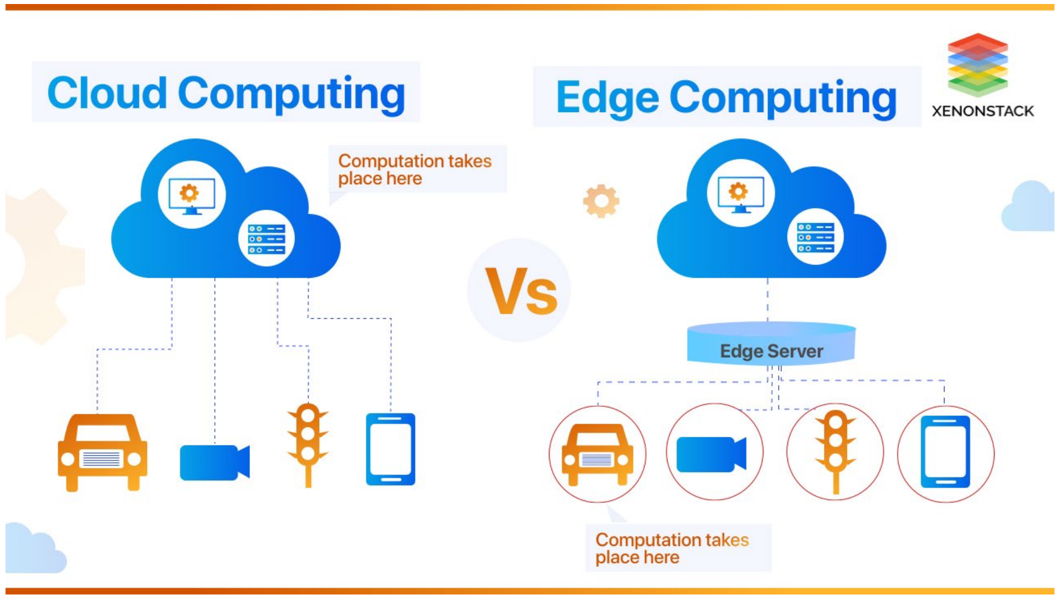
電光系統研究所 Electronic and Optoelectronic System Research Lab (EOSL)

感知運算晶片系統組 Integrated Perception and Computation Systems (P)

ITRI
Industrial Technology
Research Institute

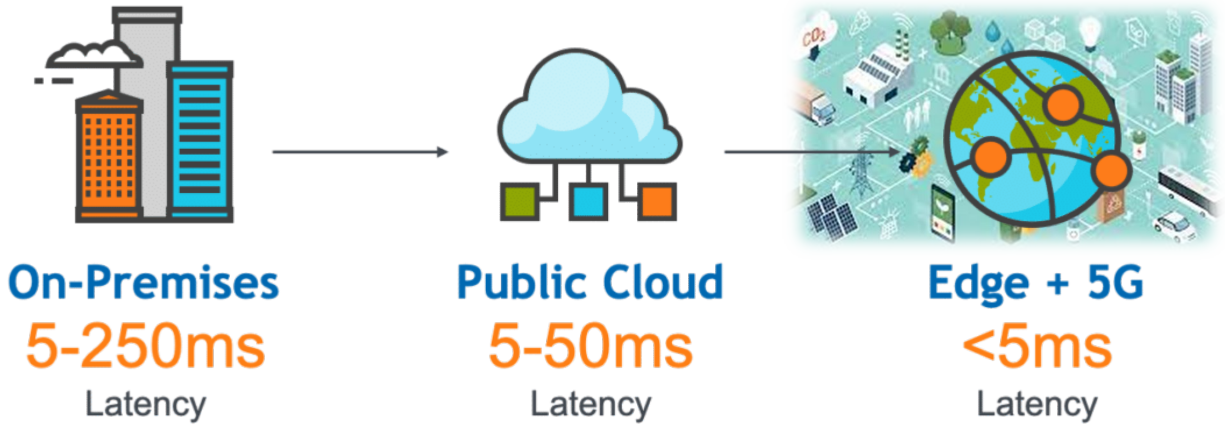
## What does AI Need ?

1. Massive **Data** acquire
2. Massive **Data** transfer
3. Massive **Data** compute



## Why Need Edge AI ?

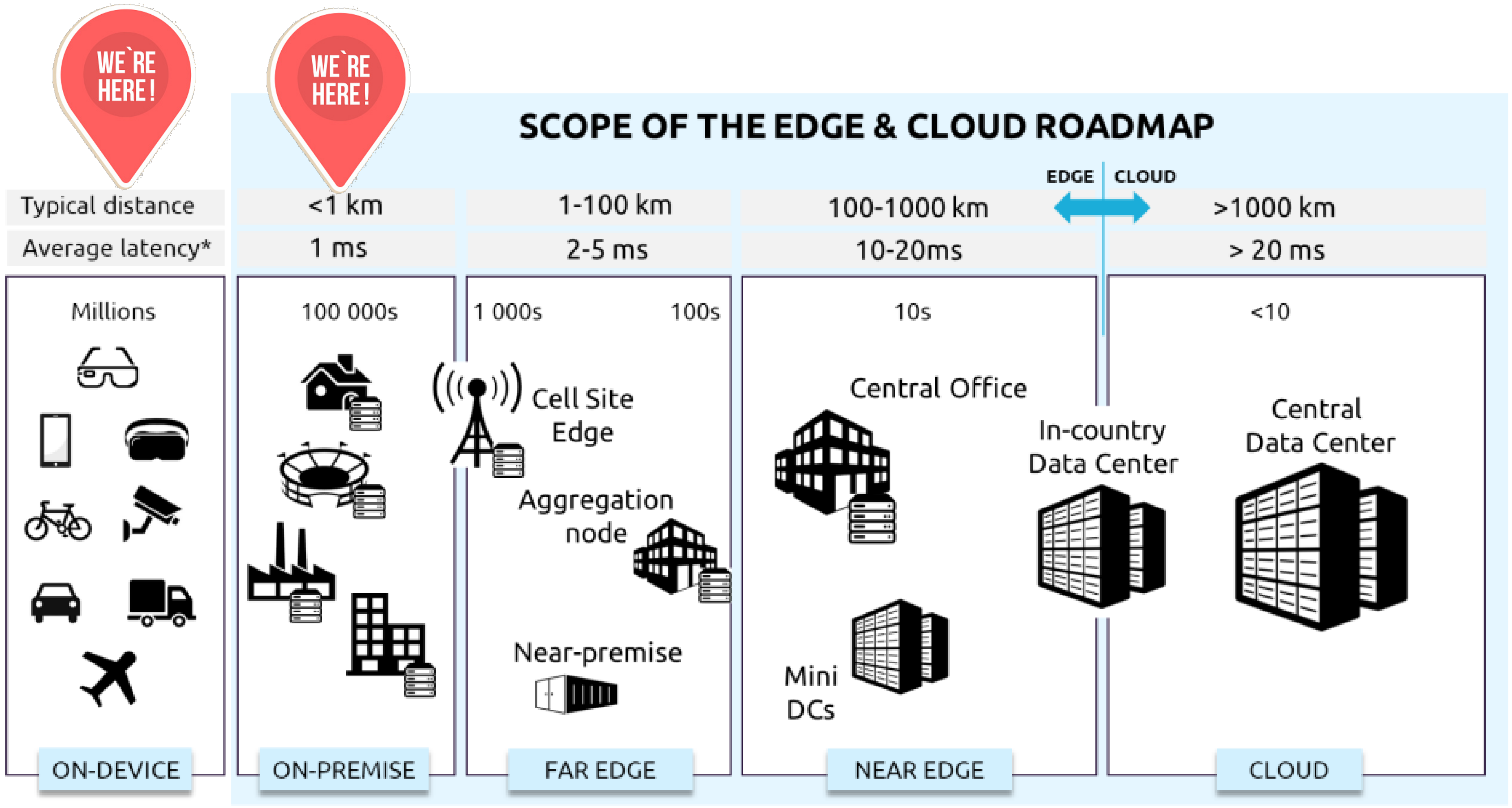
1. Latency
2. Privacy
3. Green



https://www.workspot.com/blog/the-public-cloud-drive-to-lowest-latency-infrastructure/

# Where is Our (ITRI-AI-Chip) in the Edge?

1. Device    2. Edge Server



Source： European industrial technology roadmap for the next generation cloud-edge offering

# Conventional and Deep-learning AI

## Rule-based, conventional machine learning

Inspired by experience, observations, priority relations

Analytical solution, fast and accurate for simple questions

Too many rules and low accuracy for complex classification

## Deep-learning machine learning

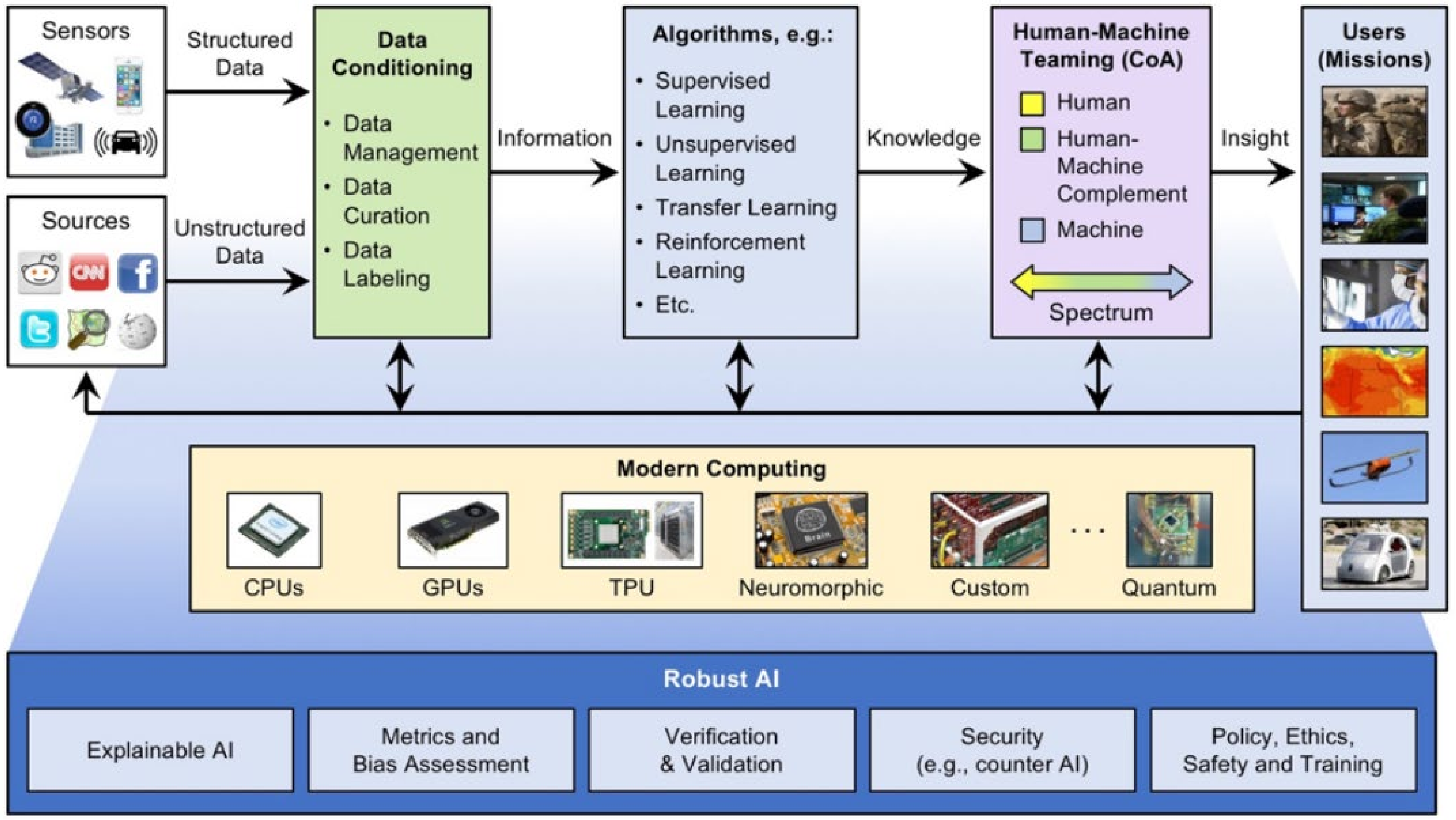
Summarize and regress massive data, may be labelled consistently

High-order, deep (> 3) relation among filters and inputs

Including supervised or non-supervised learning

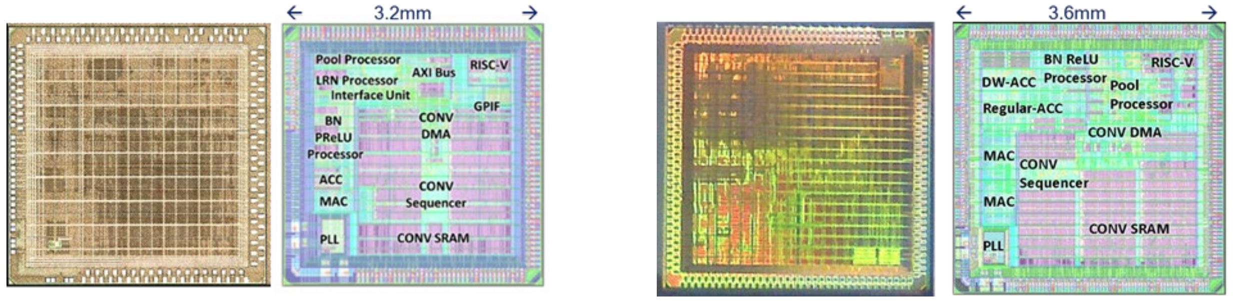
ITRI
Industrial Technology
Research Institute

# AI, Machine Learning Architecture



Survey and Benchmarking of Machine Learning Accelerators
https://arxiv.org/abs/1908.11348

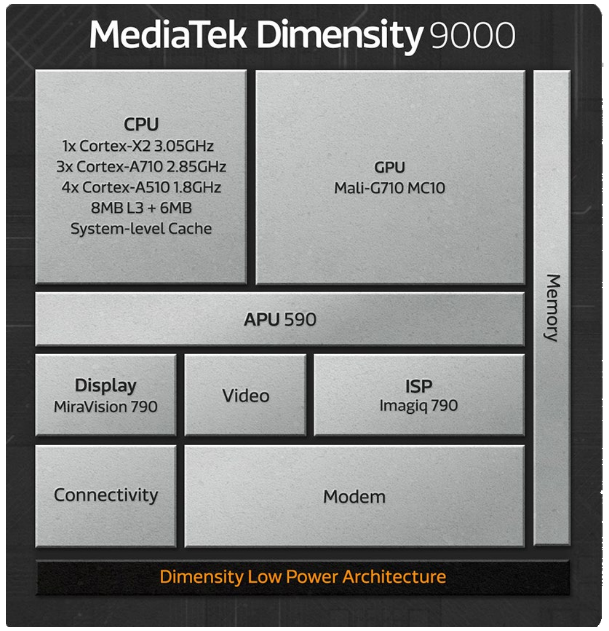
# AI Chip Products

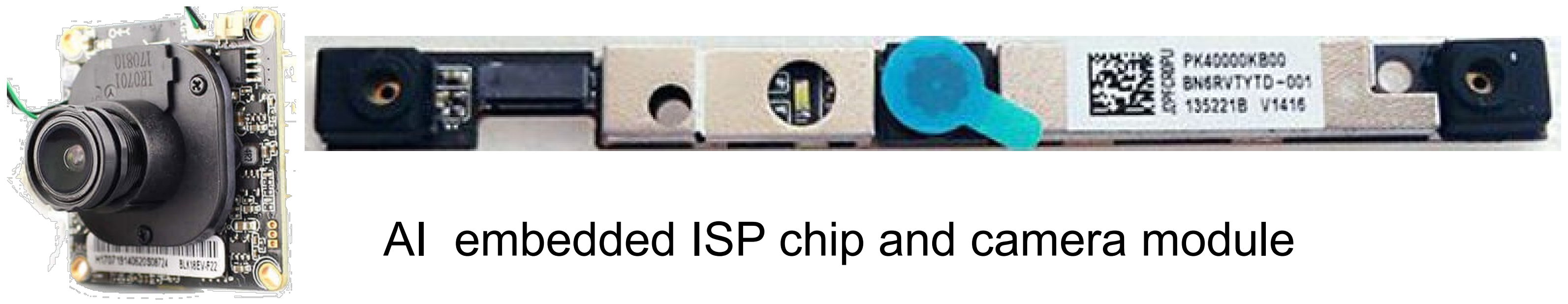


AI accelerator IP or IC

AI accelerator dongles or cards

AI embedded DEV boards

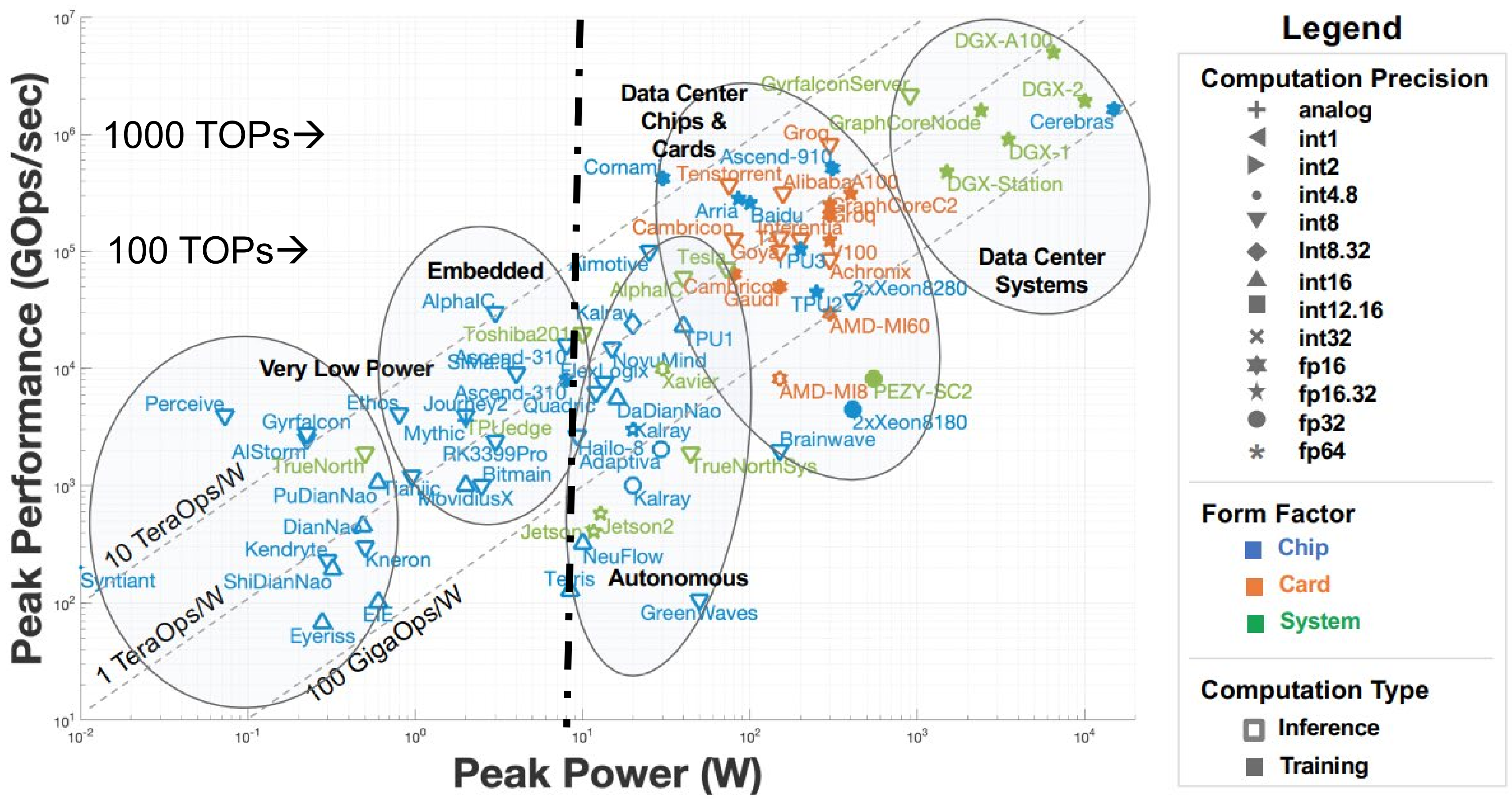




AI embedded ISP chip and camera module

AI embedded (APU, NPU…)
SOC CPU AP IC

# Performance Distribution of Machine Learning Hardware



## Machine Learning HW

## Performance Metrics

Performance index：

Tera-operators per second (TOPs)

Energy efficiency index ：

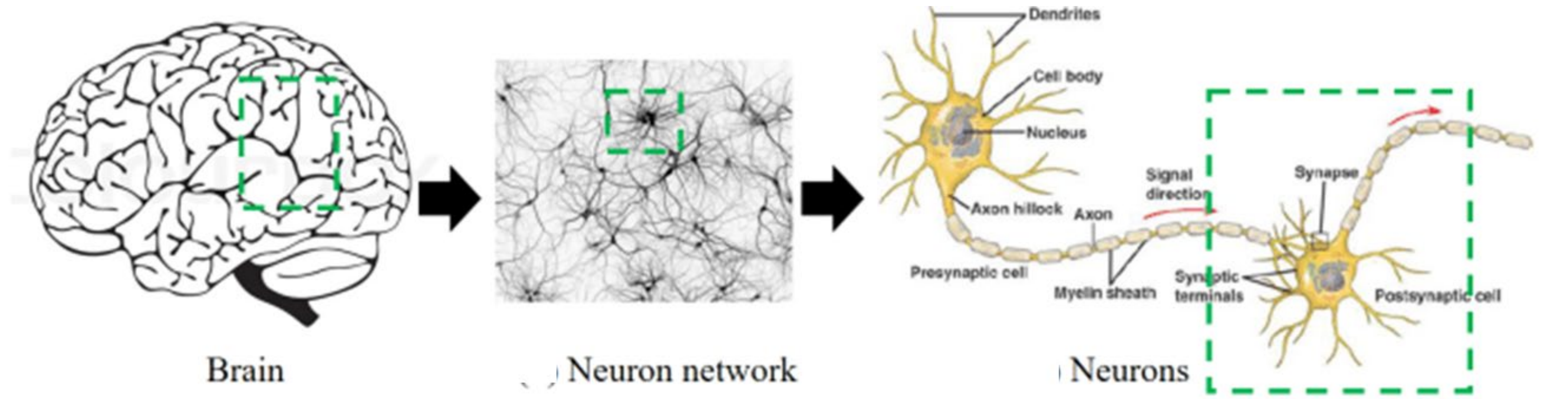
TOPs per Watt (TOPs/W)

## AI Chip Glossary

- operators： add, multiply, shift, arithmetic functions
- precision ： floating-point (FP16,32,64), integer (INT1,2,4,8,16)

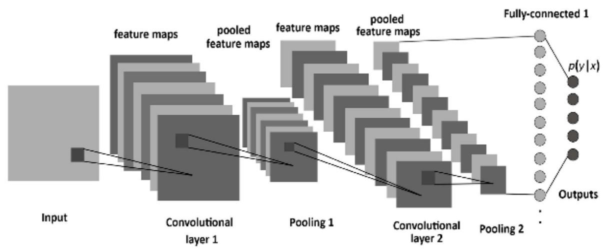
Survey and Benchmarking of Machine Learning Accelerators
https://arxiv.org/abs/1908.11348

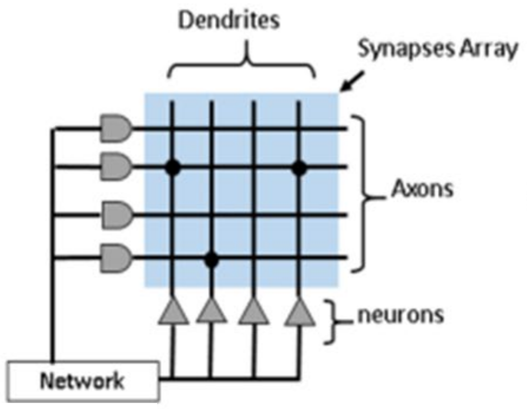
# From Neurons to Neural Networks

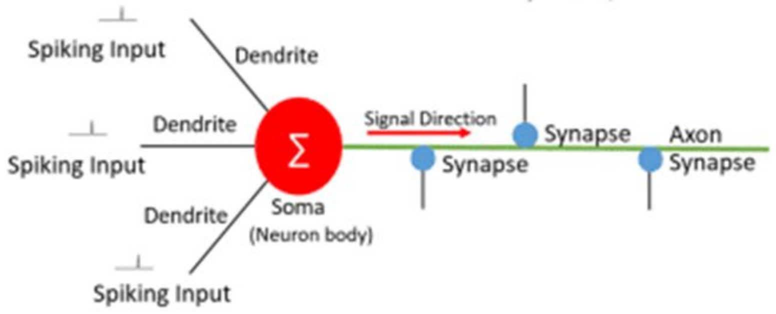


Digital

Convolutional Neural Network Solution





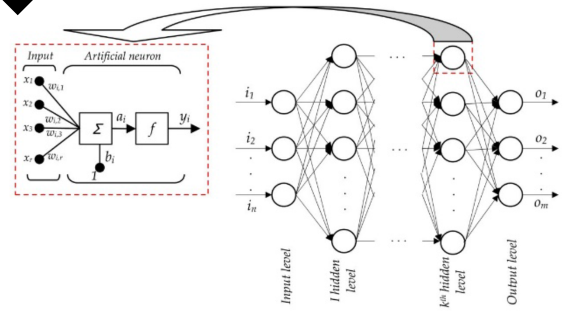


Neuron Structure

Neuron Network

Digital

Dense Neural Network Solution



Analog Mixed-mode

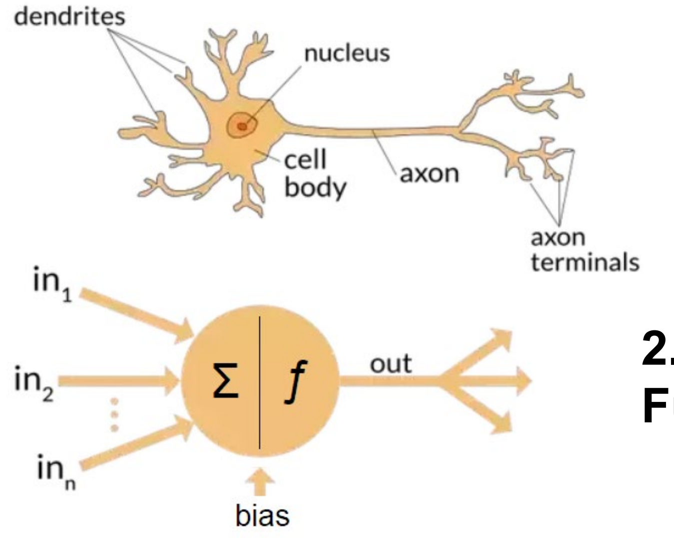
Neuromorphic, Processing in Memory

ITRI Industrial Technology Research Institute

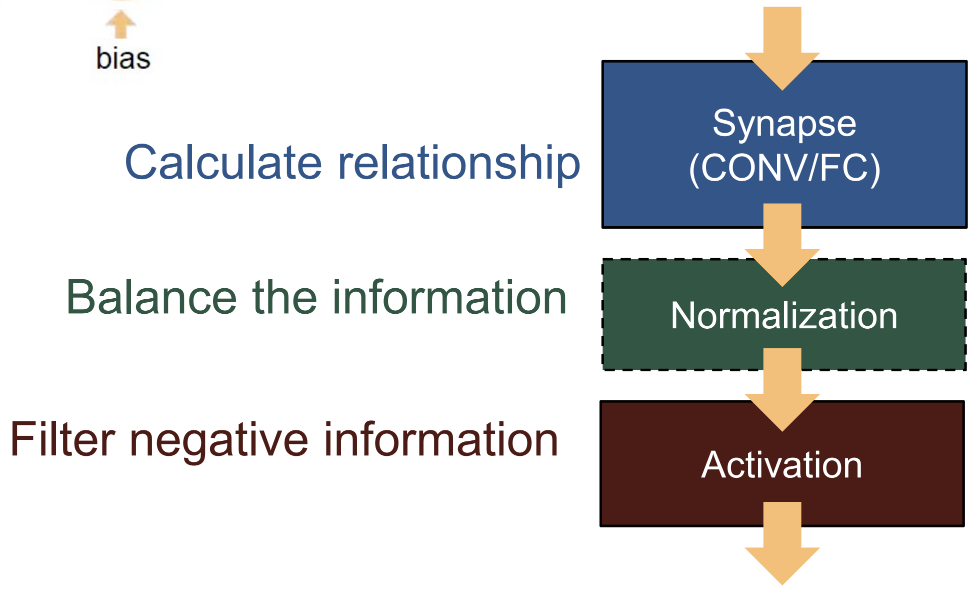
Opportunities and challenges on nanoscale 3D neuromorphic computing system
https://ieeexplore.ieee.org/document/8077906

# Neural Network Building Blocks and Graph

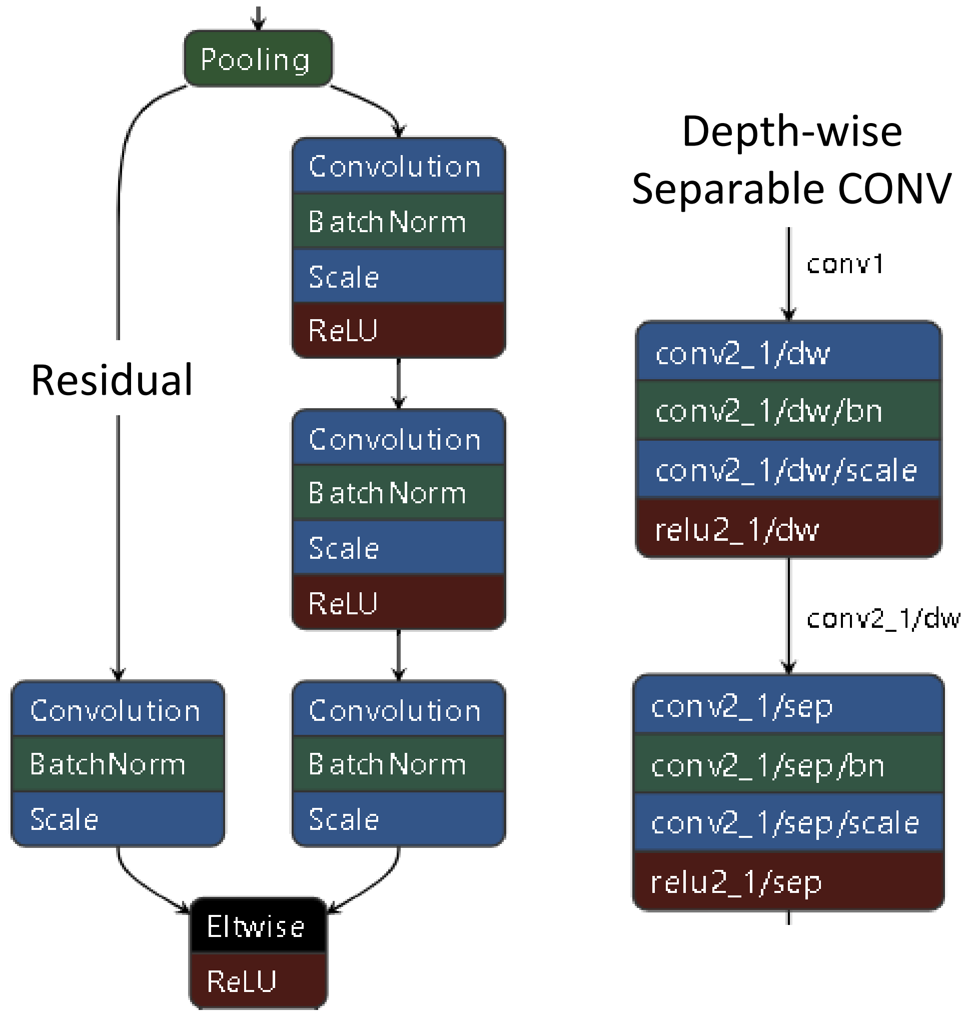
## 1. From Neurons to Functional Operators



## 2. From Operators to Functional Queues

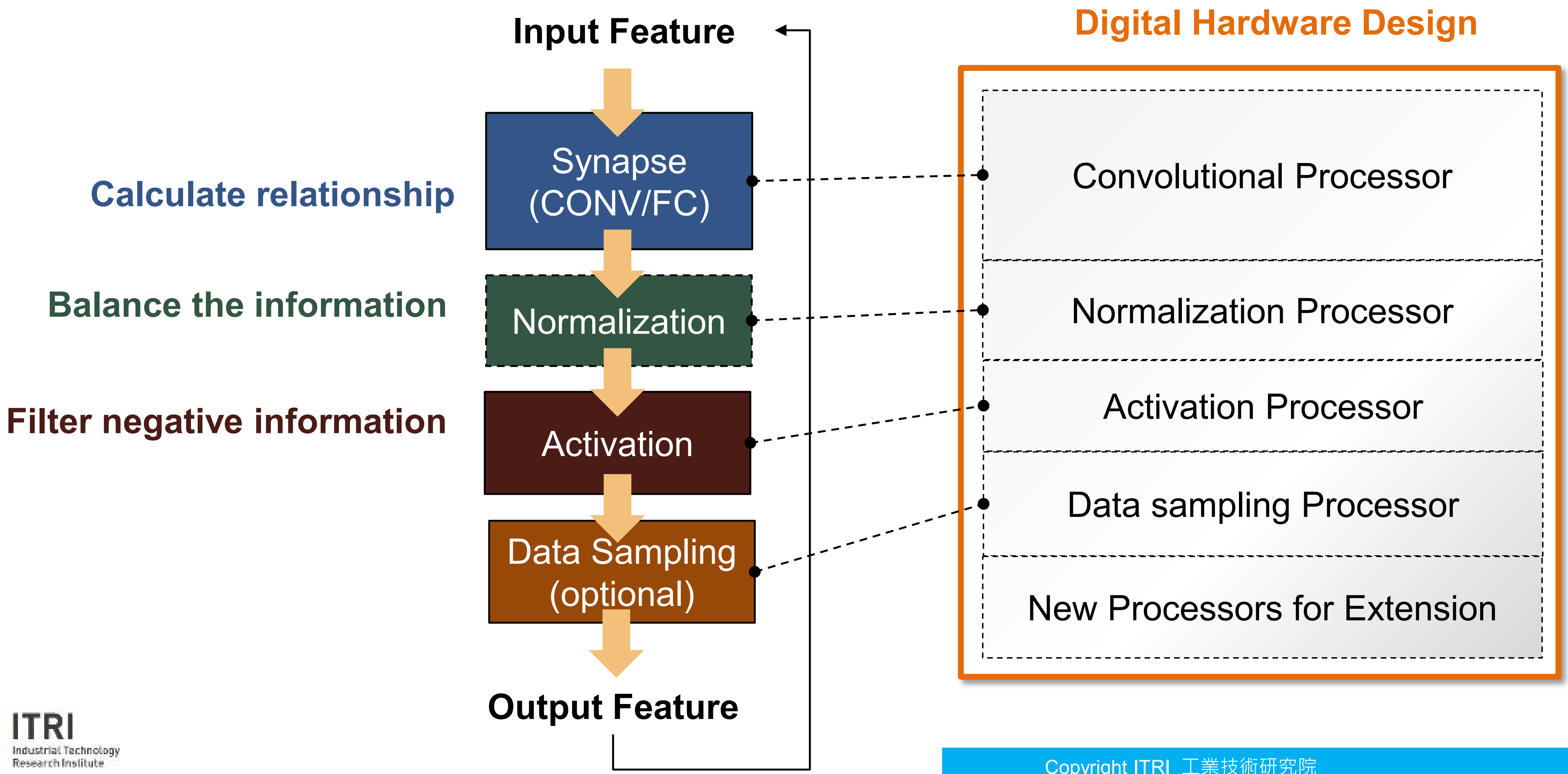


## 3. From Functional Queues to Neural Networks



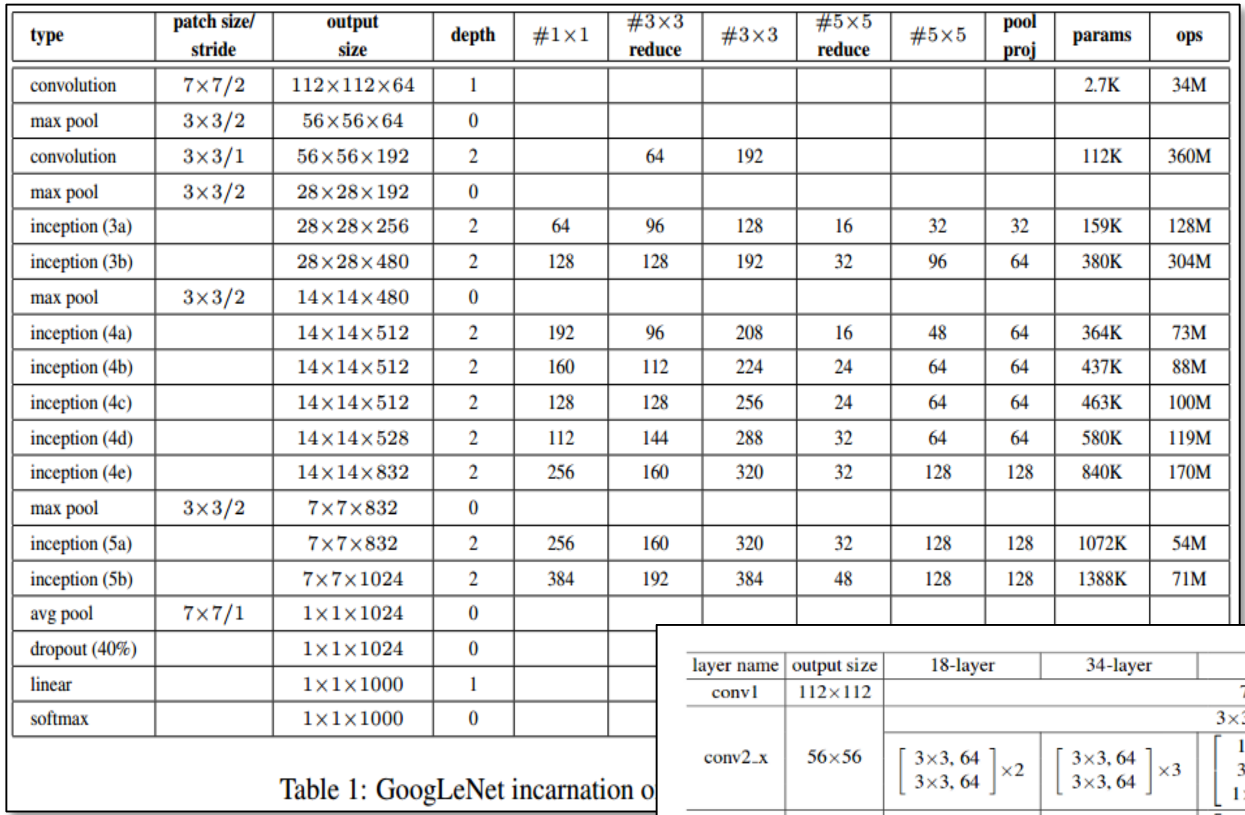
# This is How a DLA Architecture Formed

to construct a configurable HW queue that fits a generic neural operator queue



# Let's See Some Backbone CNN Models

What are those layer ?  What do you find ?  How to imagine ?



| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | | |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Table 1: GoogLeNet incarnation o

↑ Inception

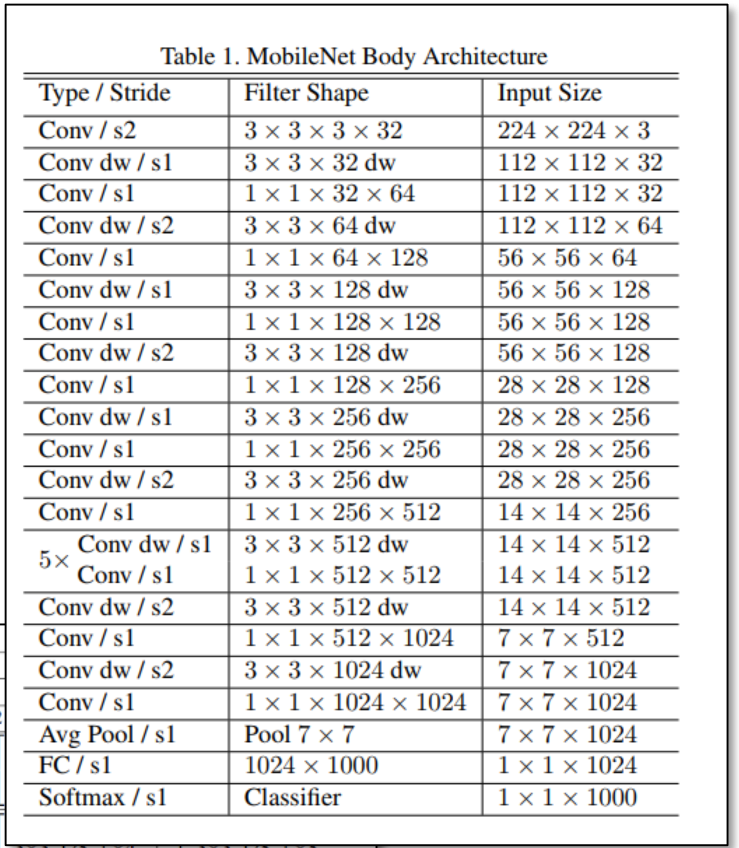
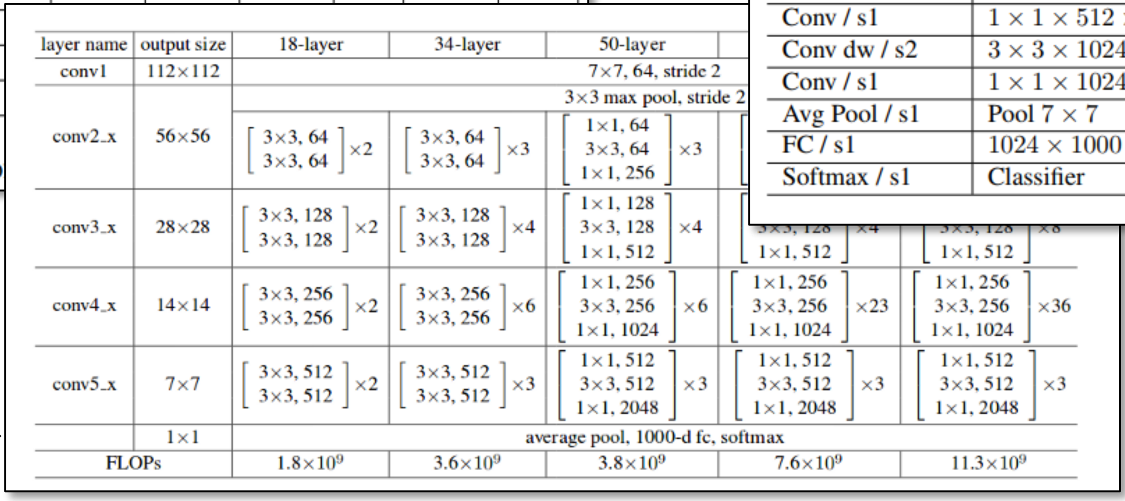


Table 1. MobileNet Body Architecture

| Type / Stride | Filter Shape | Input Size |
|---|---|---|
| Conv / s2 | 3 × 3 × 3 × 32 | 224 × 224 × 3 |
| Conv dw / s1 | 3 × 3 × 32 dw | 112 × 112 × 32 |
| Conv / s1 | 1 × 1 × 32 × 64 | 112 × 112 × 32 |
| Conv dw / s2 | 3 × 3 × 64 dw | 112 × 112 × 64 |
| Conv / s1 | 1 × 1 × 64 × 128 | 56 × 56 × 64 |
| Conv dw / s1 | 3 × 3 × 128 dw | 56 × 56 × 128 |
| Conv / s1 | 1 × 1 × 128 × 128 | 56 × 56 × 128 |
| Conv dw / s2 | 3 × 3 × 128 dw | 56 × 56 × 128 |
| Conv / s1 | 1 × 1 × 128 × 256 | 28 × 28 × 128 |
| Conv dw / s1 | 3 × 3 × 256 dw | 28 × 28 × 256 |
| Conv / s1 | 1 × 1 × 256 × 256 | 28 × 28 × 256 |
| Conv dw / s2 | 3 × 3 × 256 dw | 28 × 28 × 256 |
| Conv / s1 | 1 × 1 × 256 × 512 | 14 × 14 × 256 |
| 5× Conv dw / s1 | 3 × 3 × 512 dw | 14 × 14 × 512 |
| 5× Conv / s1 | 1 × 1 × 512 × 512 | 14 × 14 × 512 |
| Conv dw / s2 | 3 × 3 × 512 dw | 14 × 14 × 512 |
| Conv / s1 | 1 × 1 × 512 × 1024 | 7 × 7 × 512 |
| Conv dw / s2 | 3 × 3 × 1024 dw | 7 × 7 × 1024 |
| Conv / s1 | 1 × 1 × 1024 × 1024 | 7 × 7 × 1024 |
| Avg Pool / s1 | Pool 7 × 7 | 7 × 7 × 1024 |
| FC / s1 | 1024 × 1000 | 1 × 1 × 1024 |
| Softmax / s1 | Classifier | 1 × 1 × 1000 |

↑ MobileNet

Resnet →



| layer name | output size | 18-layer | 34-layer | 50-layer | | |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | [3×3, 64; 3×3, 64] ×2 | [3×3, 64; 3×3, 64] ×3 | [1×1, 64; 3×3, 64; 1×1, 256] ×3 | | |
| conv3_x | 28×28 | [3×3, 128; 3×3, 128] ×2 | [3×3, 128; 3×3, 128] ×4 | [1×1, 128; 3×3, 128; 1×1, 512] ×4 | [3×3, 128; 1×1, 512] ×4 | [3×3, 128; 1×1, 512] ×8 |
| conv4_x | 14×14 | [3×3, 256; 3×3, 256] ×2 | [3×3, 256; 3×3, 256] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×6 | [1×1, 256; 3×3, 256; 1×1, 1024] ×23 | [1×1, 256; 3×3, 256; 1×1, 1024] ×36 |
| conv5_x | 7×7 | [3×3, 512; 3×3, 512] ×2 | [3×3, 512; 3×3, 512] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 | [1×1, 512; 3×3, 512; 1×1, 2048] ×3 |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

ITRI Industrial Technology Research Institute

# This is How Our DLA Architecture Construct

**High Performance Techniques**

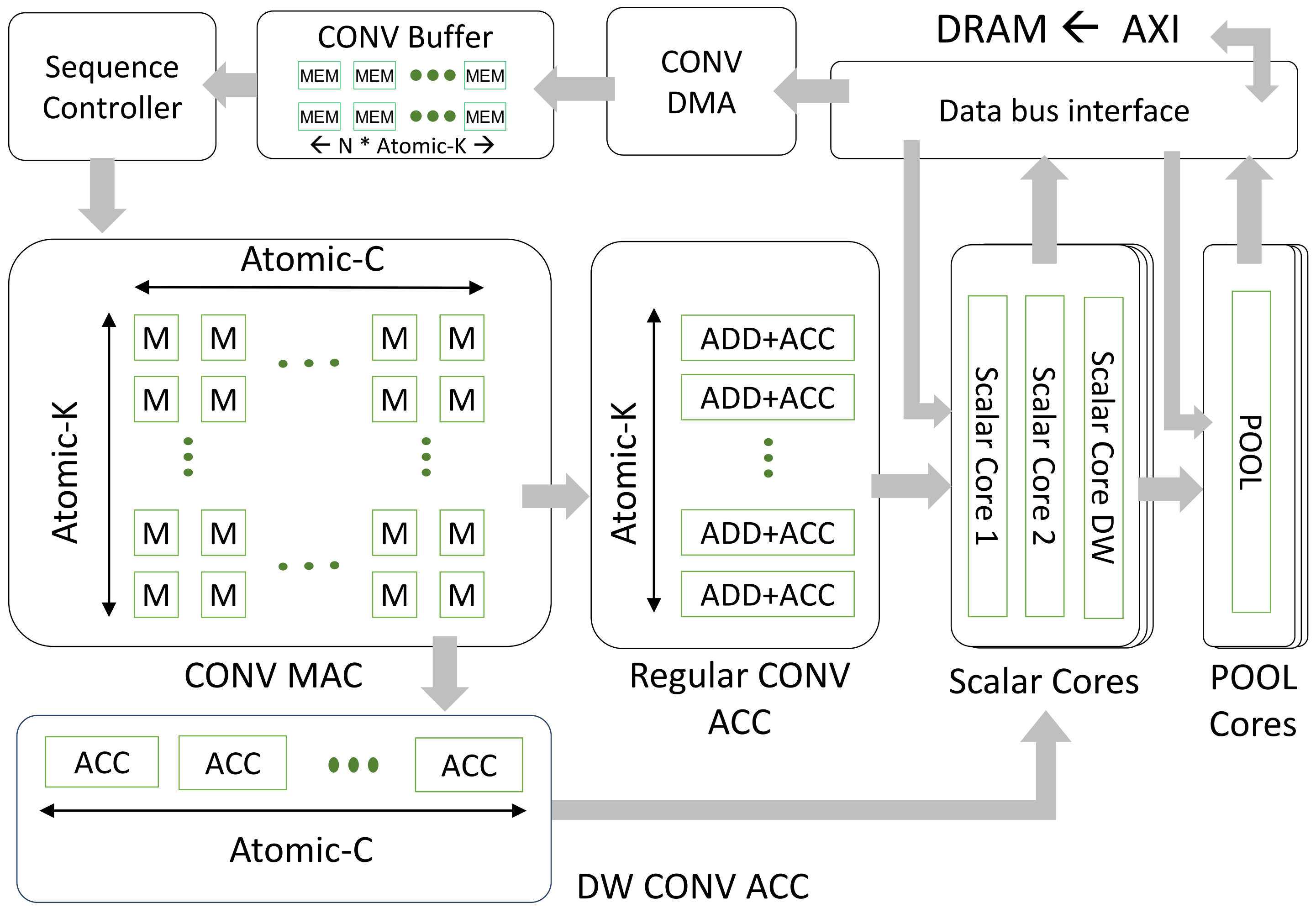
1. $2^C$ X $2^K$ Multiplier array
2. Aligned CONV buffer array
3. Aligned accumulator array

**Low-energy Techniques**

1. Fused operators for saving data
2. Multiple scalars for OP swap
3. Identical IN & OUT data format

**Ultra-low Power Techniques**

1. Adaptive gating design
2. Data pipeline retiming

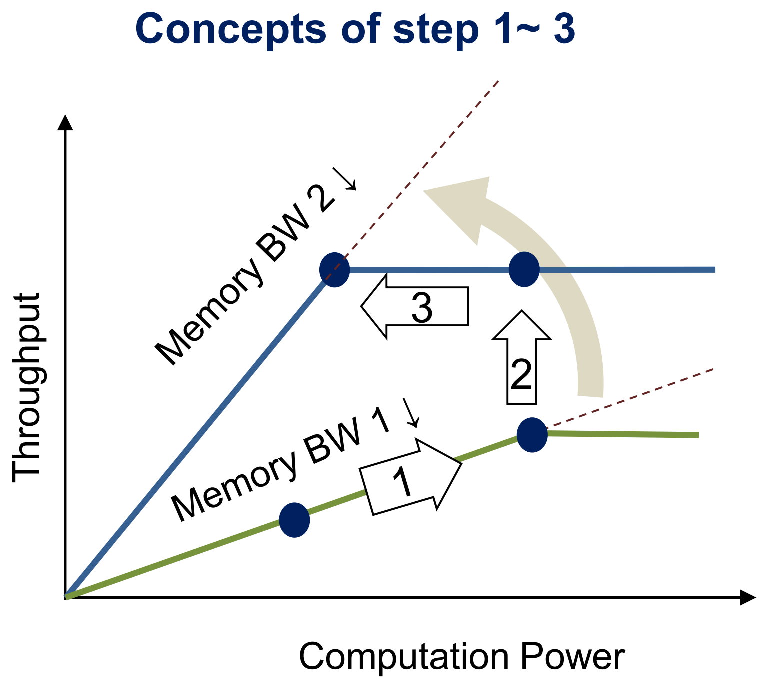


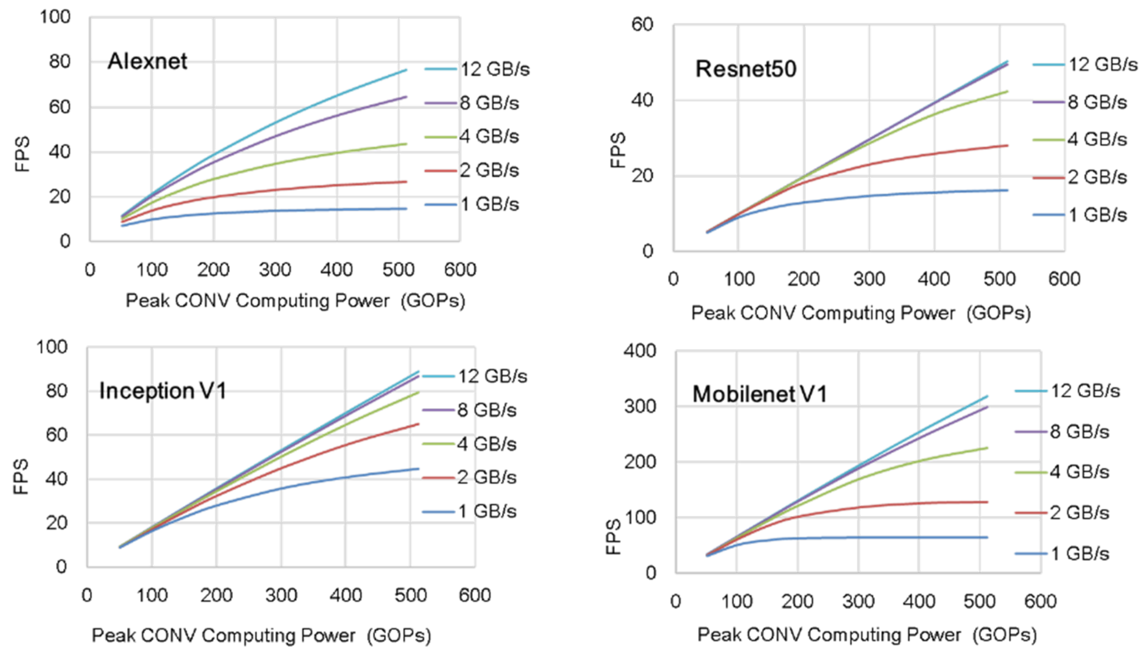
ITRI Industrial Technology Research Institute

# Optimize and Customize (1) ： CNN Accelerator

Convolution contains many independent MACs and data overlap

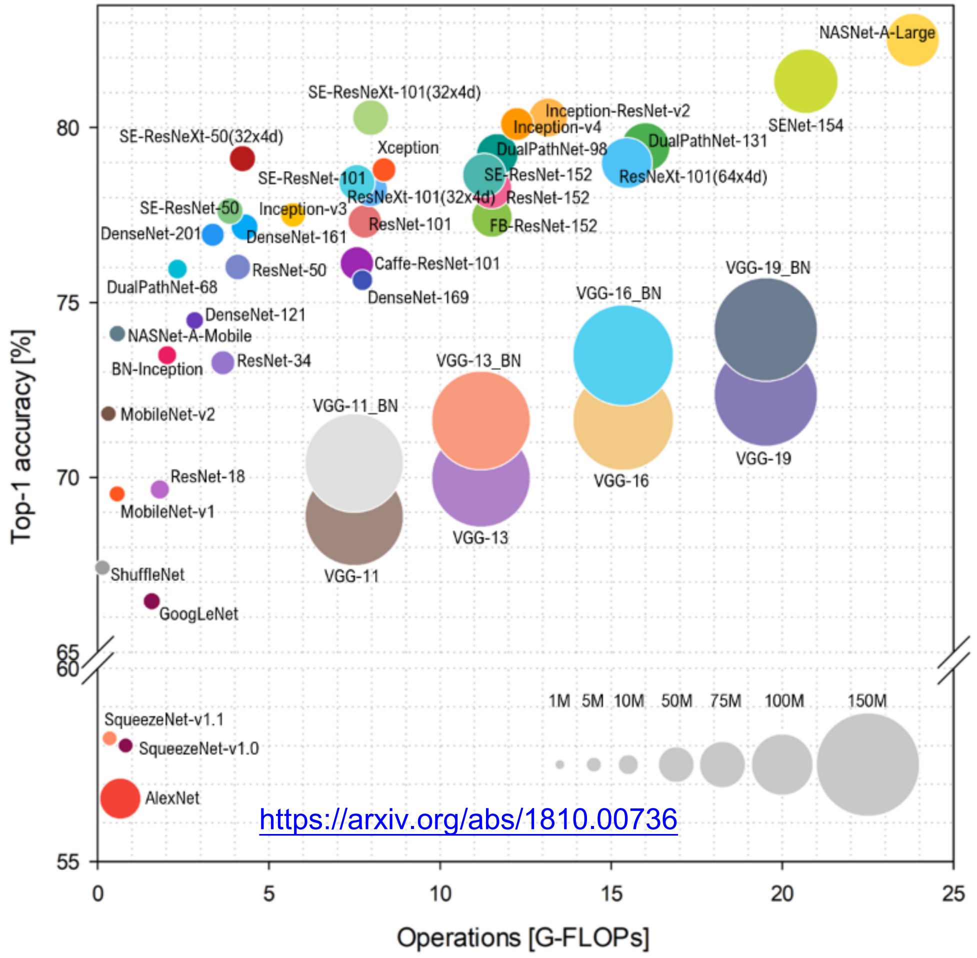
1. Increase MAC PEs with high parallelism
2. Ensure the data supplement to those PEs
3. Improve energy efficiency, adaptive to the models





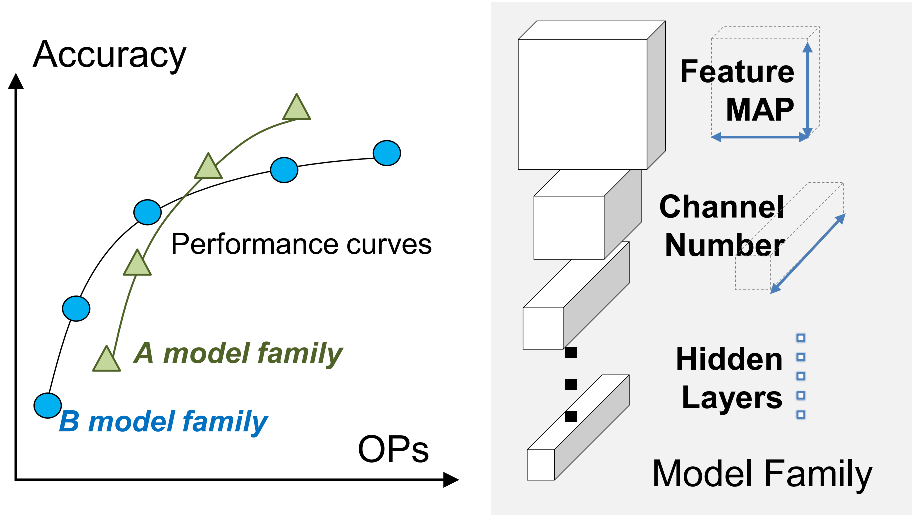
# Optimize and Customize (2) ： CNN Models

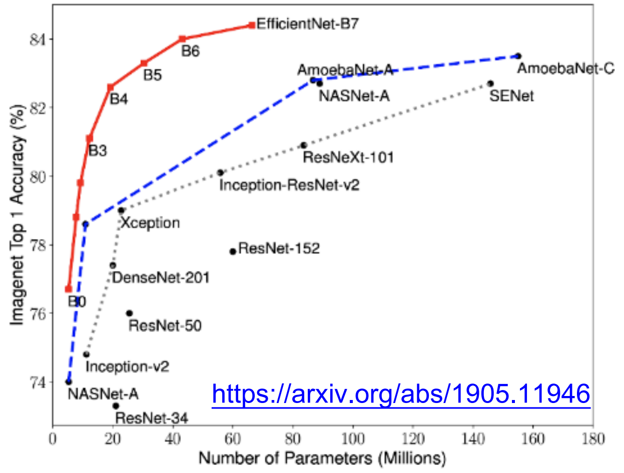
1. Select NN Model Backbone



https://arxiv.org/abs/1810.00736

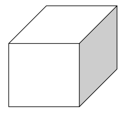
2. Search for the best configuration





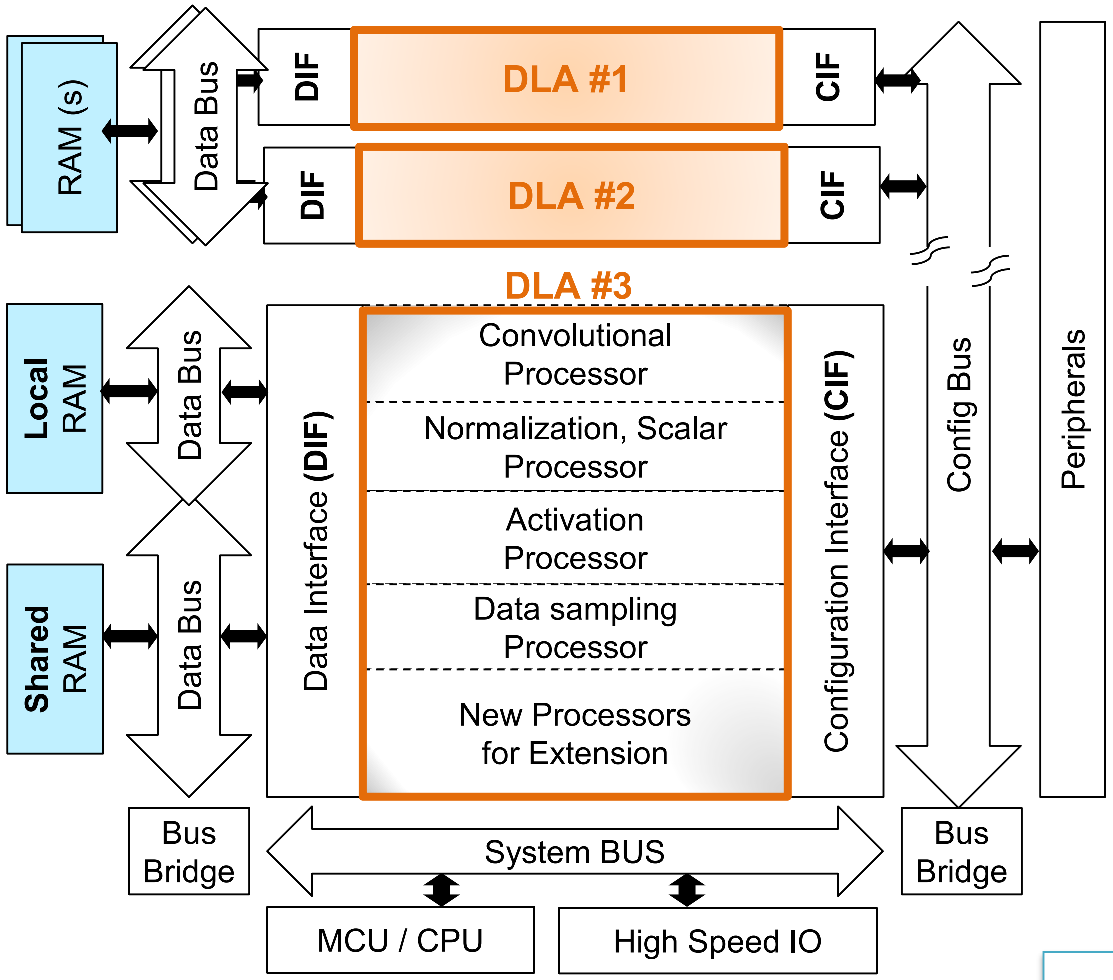
https://arxiv.org/abs/1905.11946

3. Adapt to HW effective, HW specific features



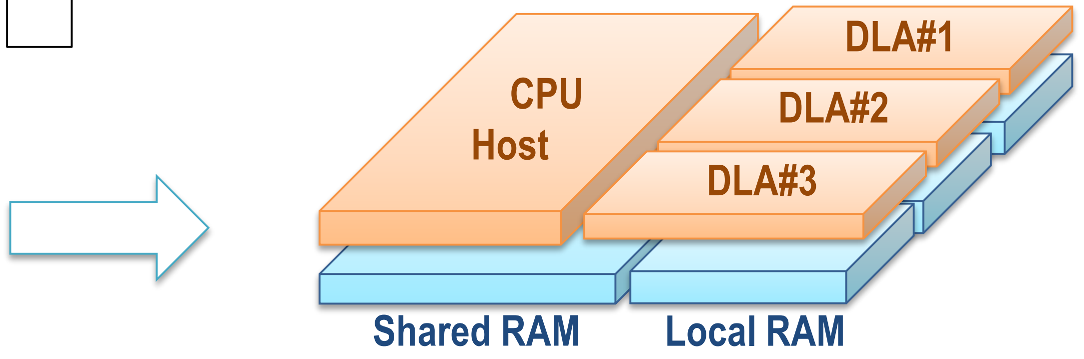
HW accelerates 3x3x8 most effectively

# ITRI Configurable and Scalable DLA Architecture

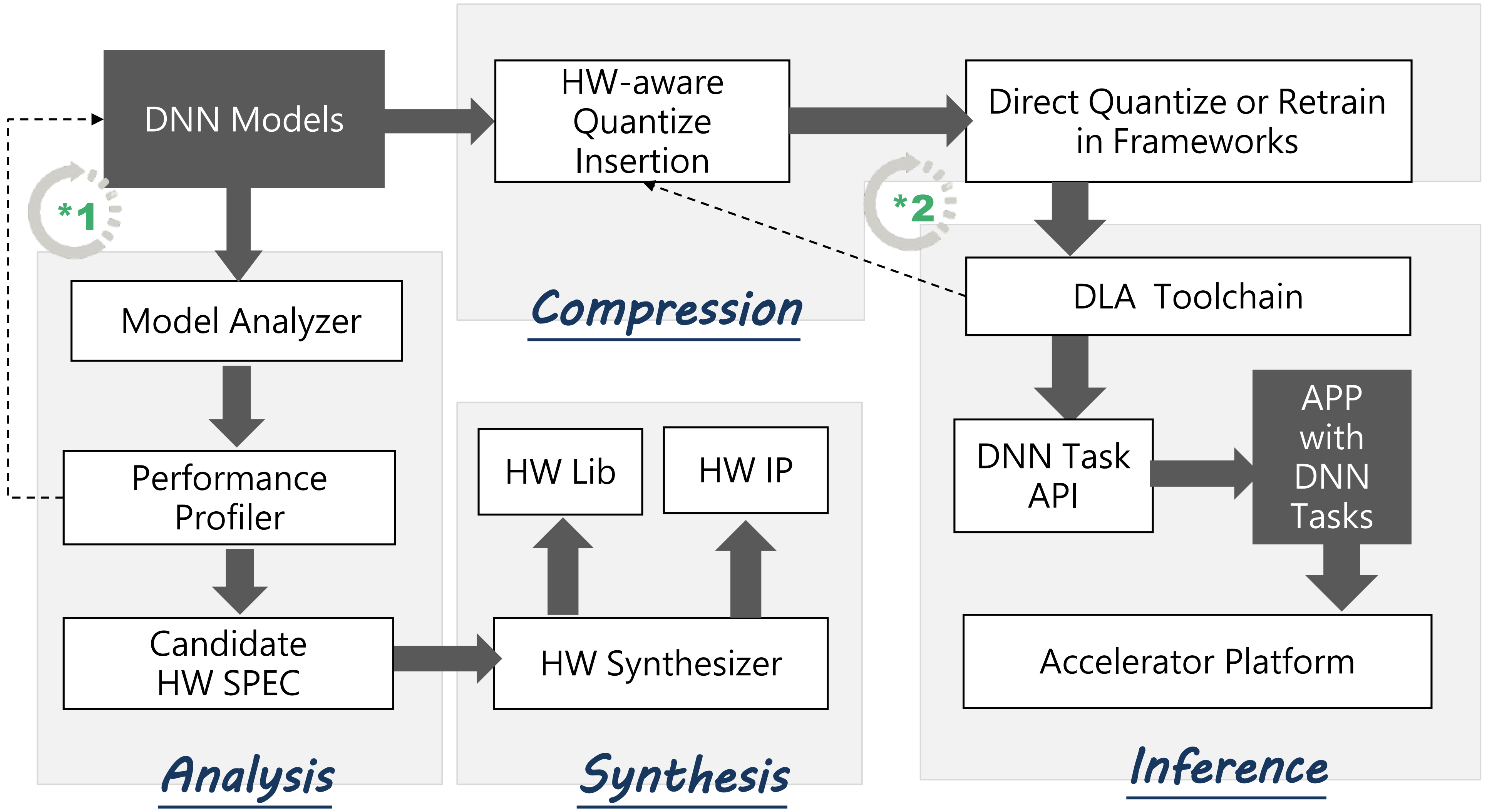


Our structure inherits NVDLA, with various extensions

1. Configurable DLA number
2. Configurable MAC number
3. Configurable scalar cores for normalization, scale, quantize
4. Optional activation functions
5. Optional data sampling processors
6. Optional new processors
7. Custom CPU or MCU as the host



# ITRI Service, Develop Flow of the DLA Platform

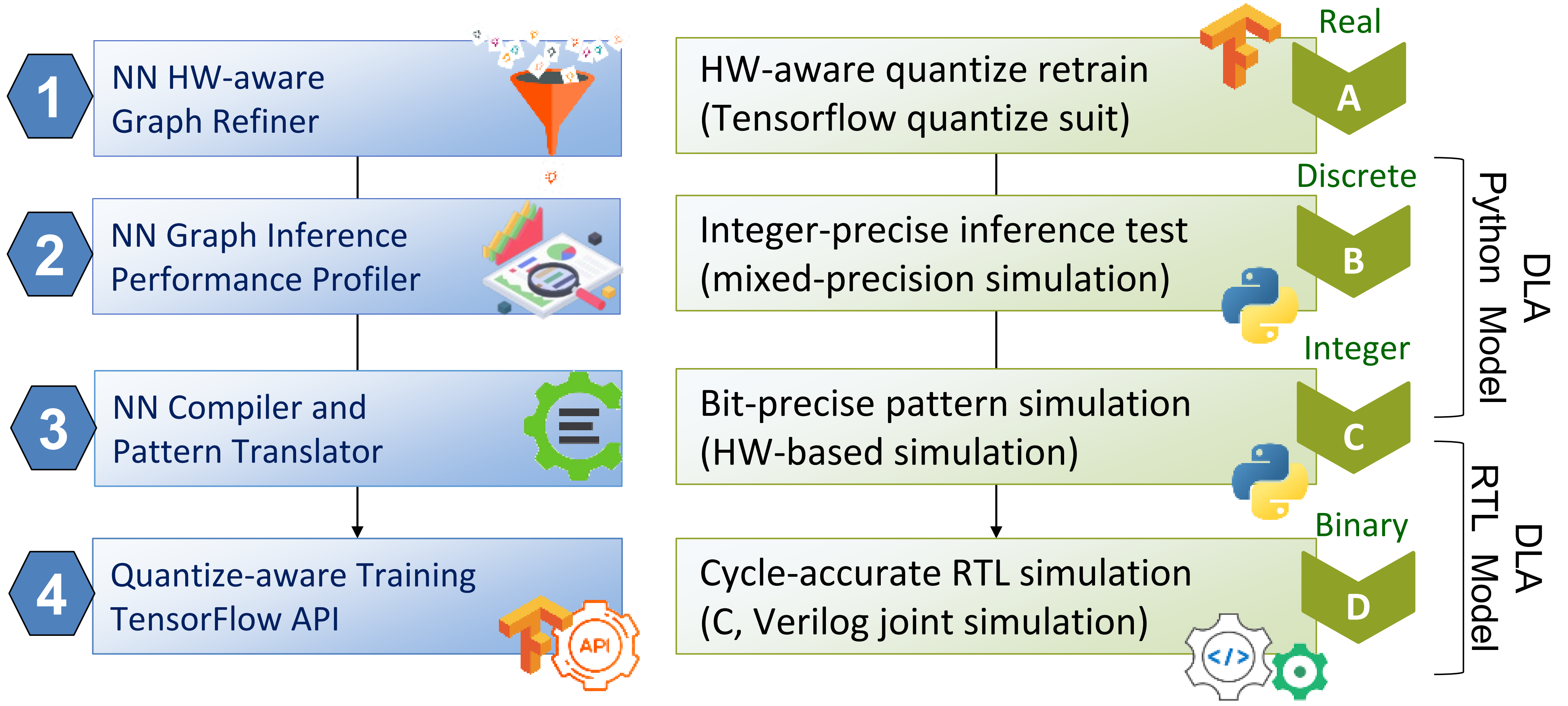


*1. Network architecture search  *2. HW-precise quantize modification

# ITRI DLA Tool Chain and Verification Flow

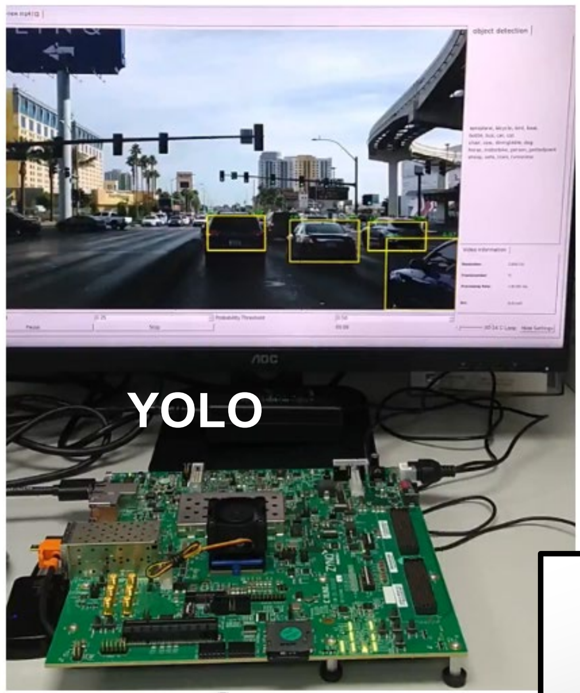
from QAT, Profile, Bare-metal Compile, Simulate, to Deploy

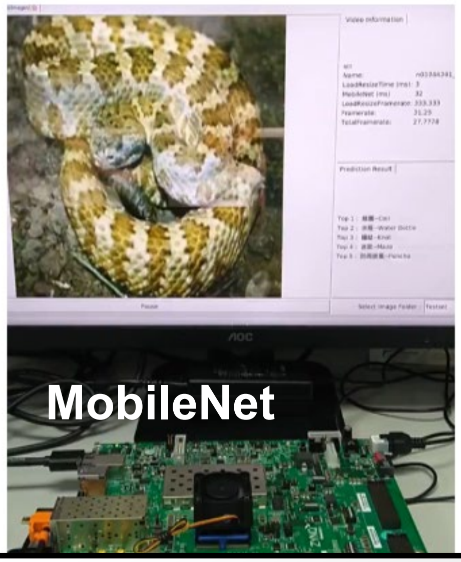
4 Main Tools + 4 Numerical Steps : linking from training to inference

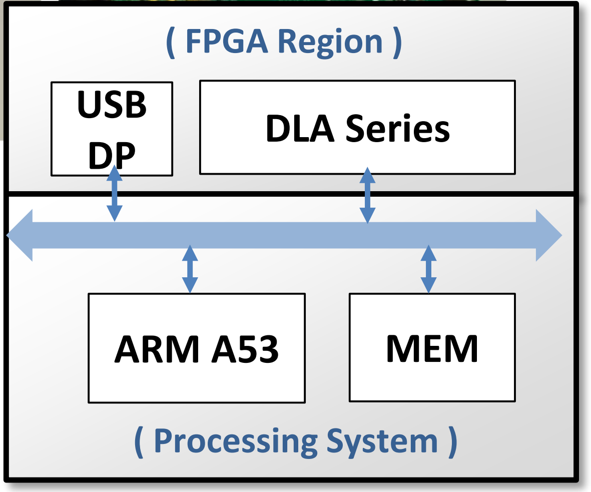

1
NN HW-aware Graph Refiner
2
NN Graph Inference Performance Profiler
3
NN Compiler and Pattern Translator
4
Quantize-aware Training TensorFlow API
API
HW-aware quantize retrain (Tensorflow quantize suit)
Real
A
Integer-precise inference test (mixed-precision simulation)
Discrete
B
Bit-precise pattern simulation (HW-based simulation)
Integer
C
Cycle-accurate RTL simulation (C, Verilog joint simulation)
Binary
D
&lt;/&gt;
DLA Python Model
DLA RTL Model

# ITRI DLA IP and Standalone FPGA Prototype

| DLA Series | DLA256M | DLA2048M |
|---|---|---|
| MAC Number | 256 | 2048 |
| CONV SRAM | 128KB | 512KB |
| Norm. Engine | 1X | 8X |
| Pool Engine | 1X | 4X |
| AXI Data BW | 1X | 8X |
| FPGA Prototype | 43% Logic, 3%DSP | 78% Logic, 86% DSP |
| FPGA Frequency | 200 MHz | 125 MHz |
| **Reference Model Speed (fps)** | | |
| Tiny YOLO v1 | 14.4 | 28.7 |
| Tiny YOLO v2 | 7.5 | 28.7 |
| Tiny YOLO v3 | 12.3 | 32.3 |
| Full YOLO v3 | 1.07 | 4.5 |
| Full YOLO v4 | 1.04 | 4.4 |
| Resnet50 | 6.4 | 17.6 |
| Mobilenet | 33 | 110 |

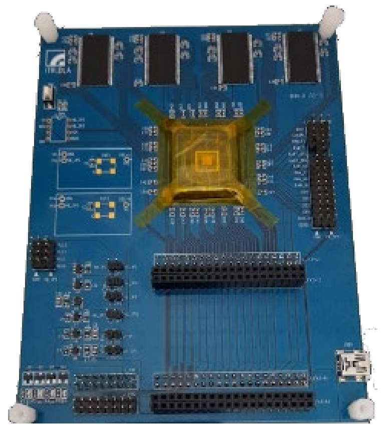


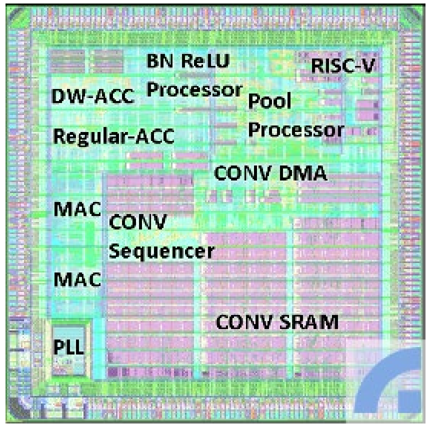




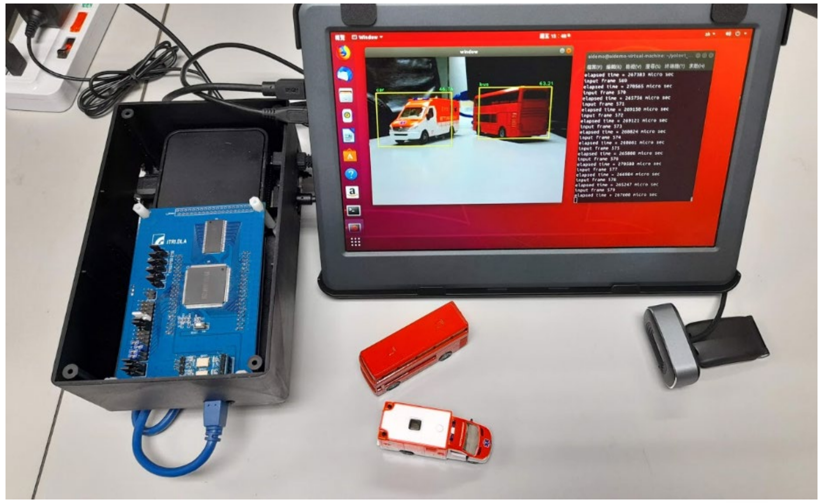
Standalone Xilinx ZCU102 FPGA Prototype

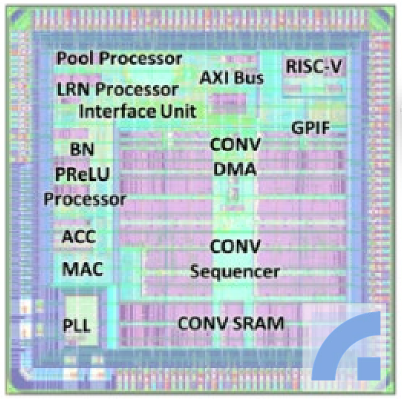
# DLA + RISC-V as a USB Accelerator

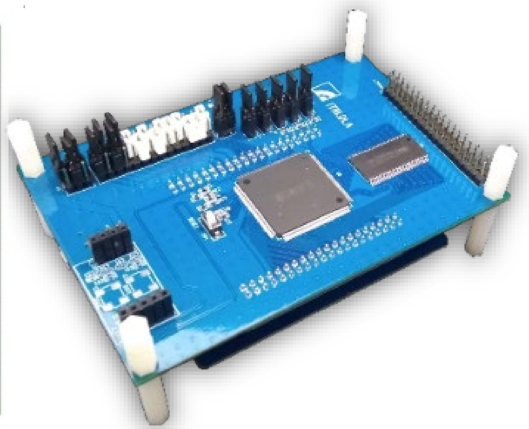




**DLA 256M** @65nm, 200 GOPs,
80 mW, 2.5 TOPs/W

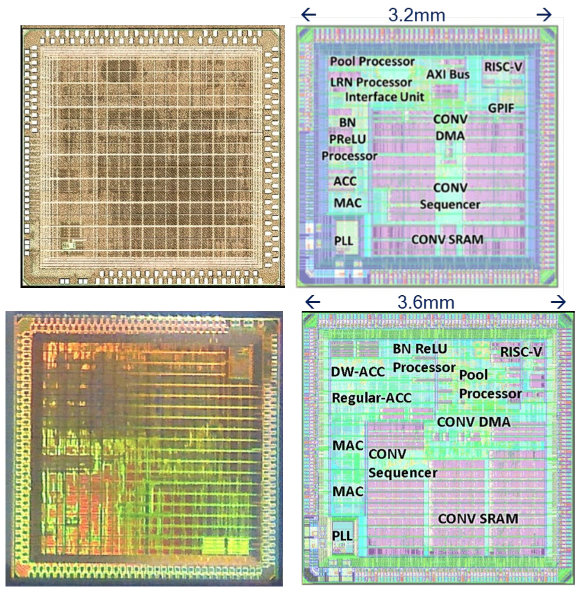




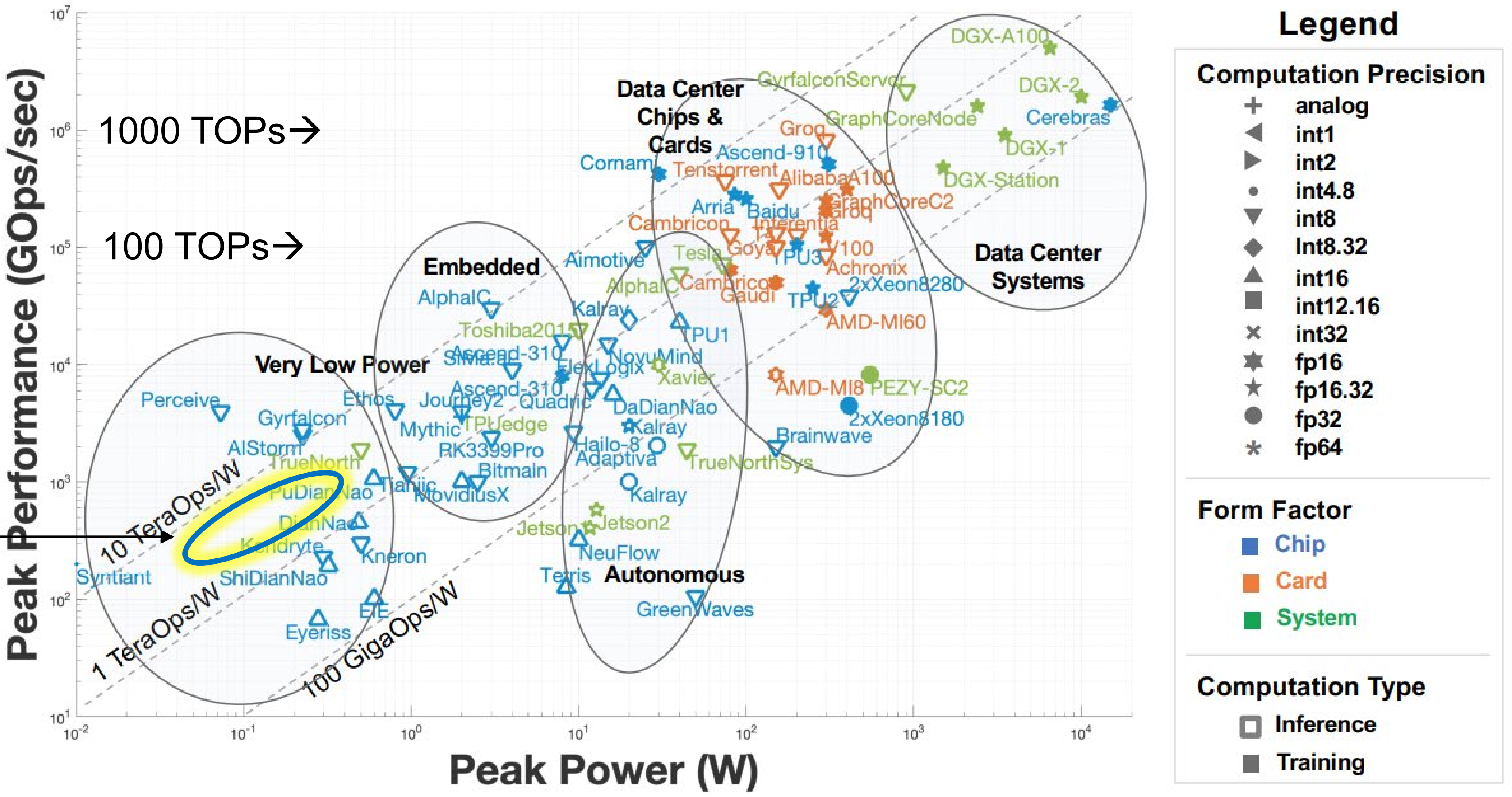


**DLA 64M** @65nm, 50 GOPs,
60 mW, 0.83 TOPs/W

# ITRI DLA AI Chip Position on the Accelerator Map



256MAC @ 65nm,
200 GOPs/ 80mW,
2.5 TOPs/W



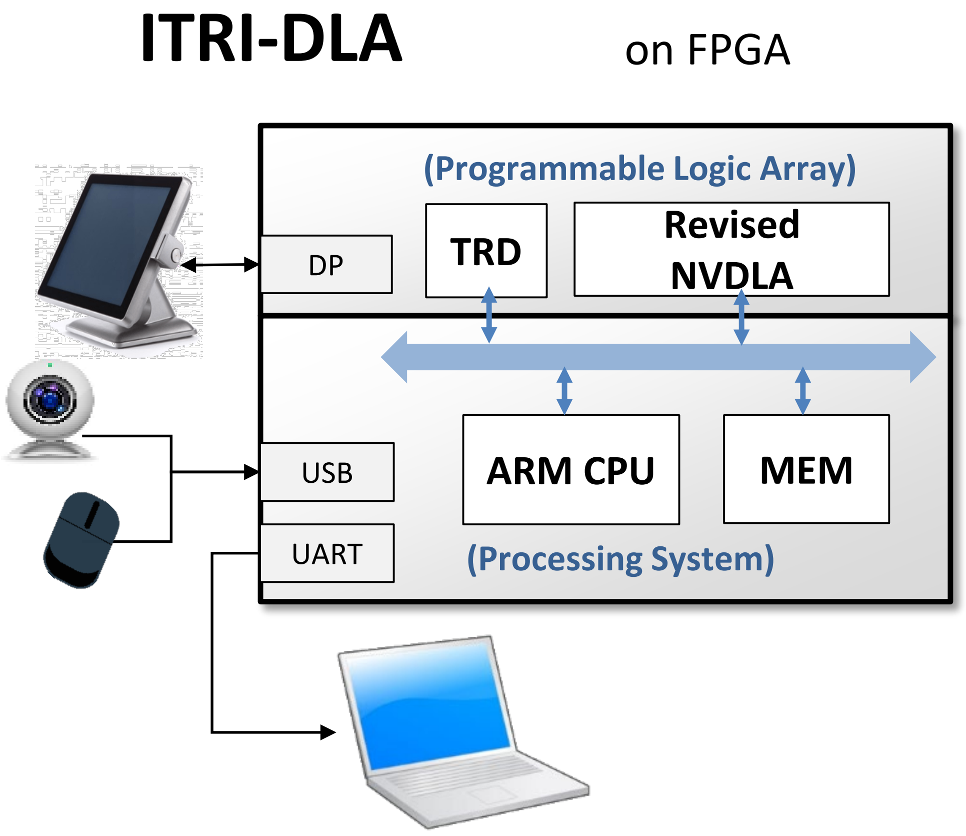
Survey and Benchmarking of Machine Learning Accelerators
https://arxiv.org/abs/1908.11348

## Summaries

1. Introduction of the need of edge and device AI.
2. Introduction of the ITRI DLA AI chip.
3. Like CPU/GPU/DSP, AI accelerator need its own SDK.
4. AI chip’s SW works actually >> HW works.
5. AI chips can be embedded or plug-in dongles.
6. No heat sink nor fan makes AI edge and device fly.
7. In-situ AI decision saves massive raw data movement$\rightarrow$ green.

# ITRI-OpenDLA FPGA for Trial Evaluation

https://github.com/SCLUO/ITRI-OpenDLA



- DLA RTL fo FPGA
- MPSOC FPGA Support
- Standard CNN Apps
- Live camera / video / image Modes

## THANK YOU!

Questions and Comments?

ITRI-OpenDLA

https://github.com/SCLUO/ITRI-OpenDLA

DLA Perf Profiler

https://github.com/SCLUO/Open-DLA-Performance-Profiler