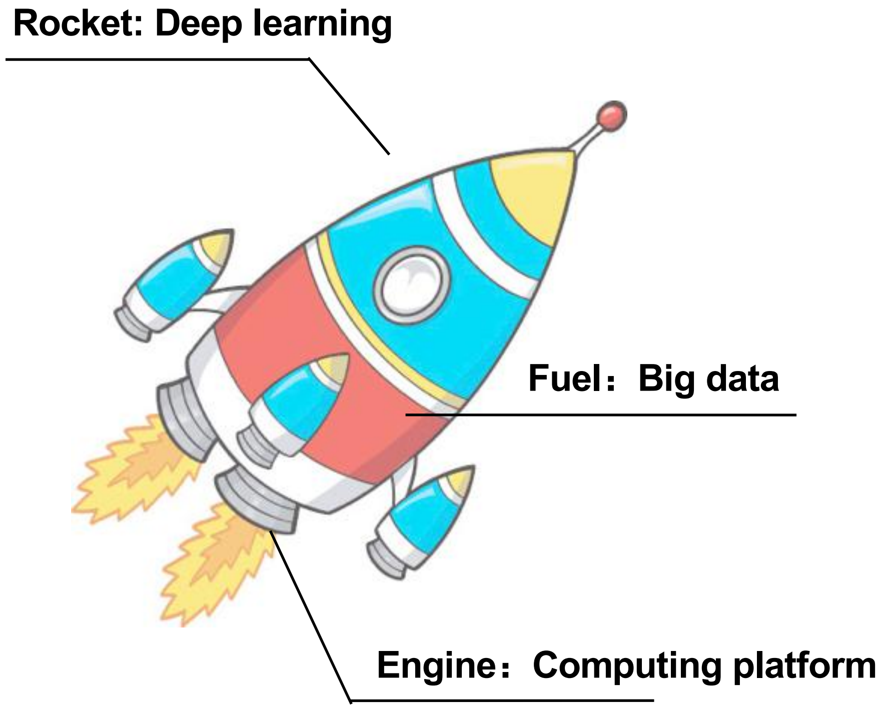
Neural Networks on Chip Design from the User Perspective

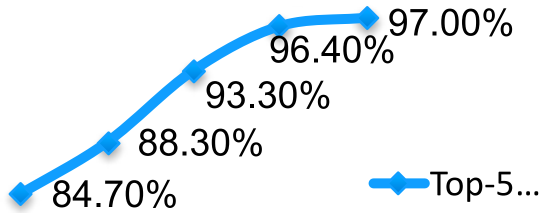
Yu Wang @ VLSID 2020
Prof. of E.E. Dept., Tsinghua University
Co-founder of DeePhi Tech (now part of Xilinx)
yu-wang@tsinghua.edu.cn
https://nicsefc.ee.tsinghua.edu.cn

Deep Learning for Everything

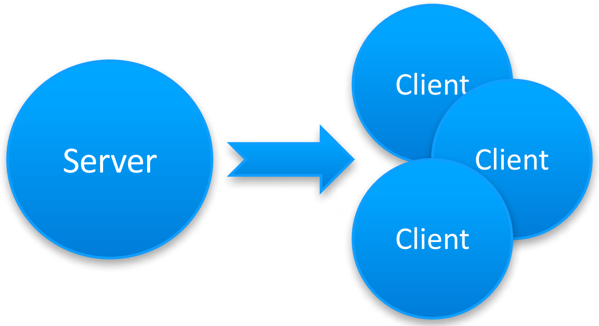


According to Prof. Andrew Ng

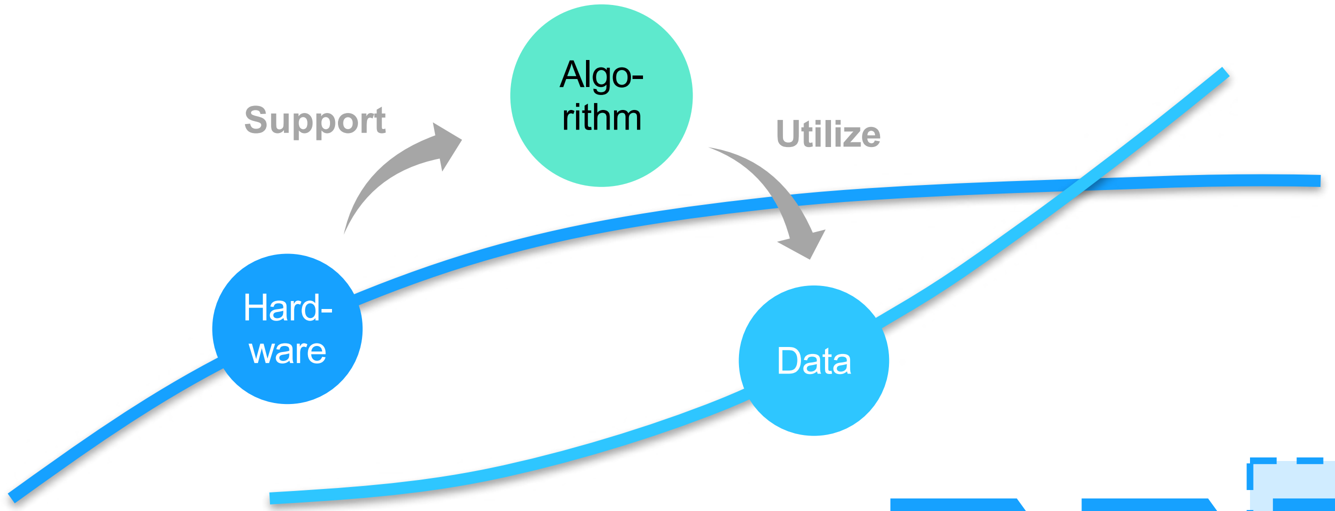
Training
– How accurate the model can de

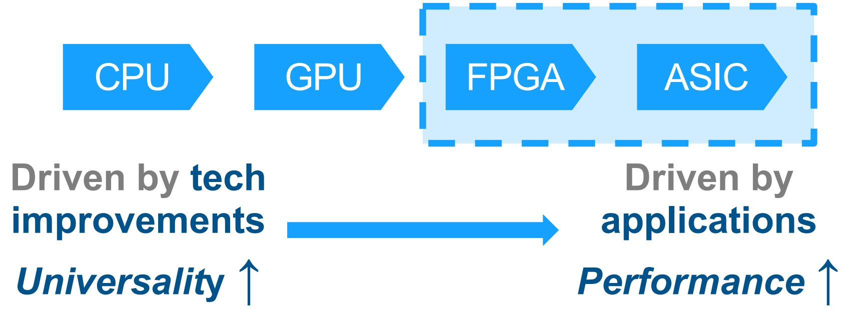


Inference
– **How many applications can use DL**



Current Hardware is Insufficient to Support Algorithm and Data





Driven by **tech improvements**

Universality ↑

Driven by **applications**

Performance ↑

The New Era is Waiting for the Next Rising Star

PC	**Game**	**Phone**	**AI**
100 billions markets	**100 billions markets**	**100 billions markets**	**Trillions markets**
CPU	GPU	Mobile CPU	**Next Big Thing**
intel	NVIDIA	ARM	

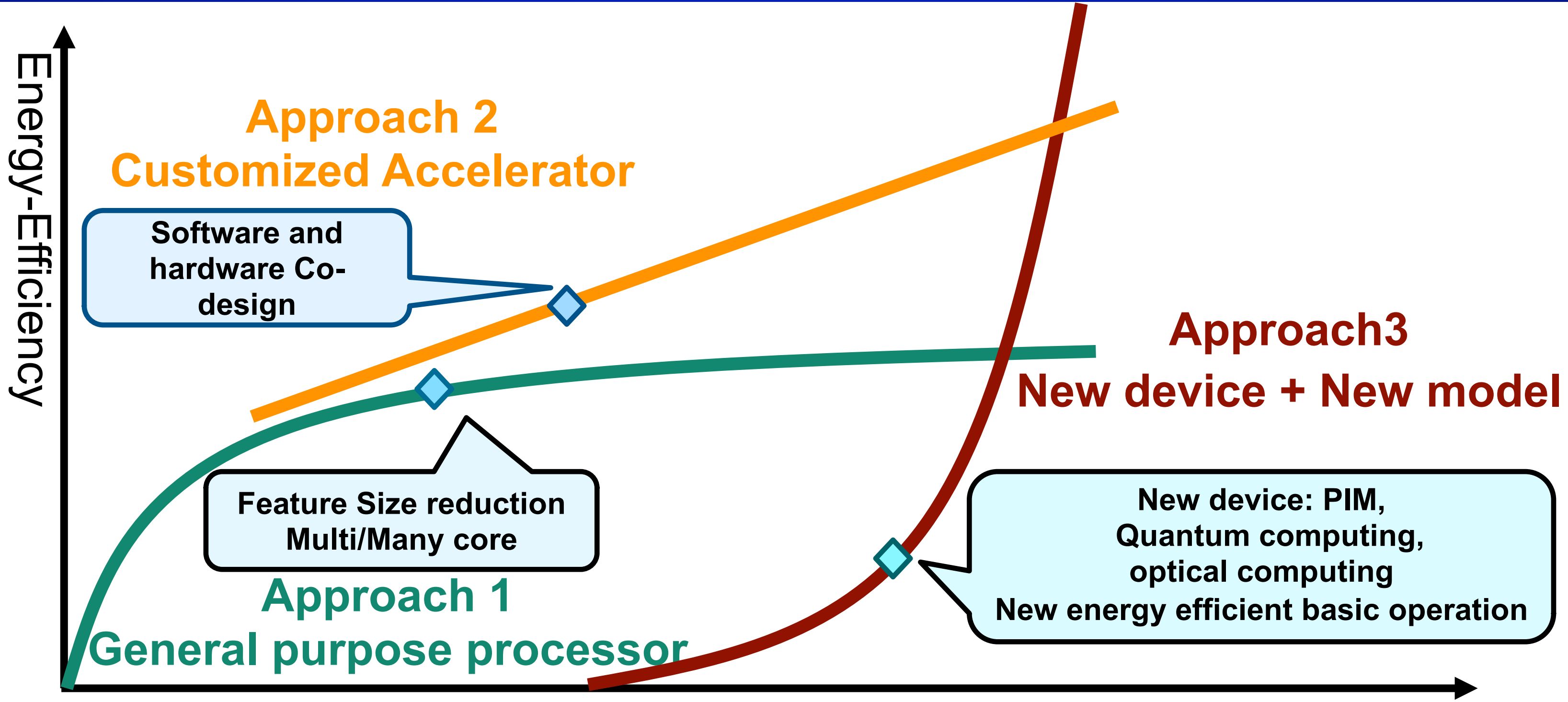
Back to 2012…

Are GPUs Enough?

Good, but not good enough….

Why? Power Consumption and Latency Are Crucial

Development of Energy-Efficient Computing Chips



Rapid development of technology brings problem of circuit power and reliability

Big data analysis proposed higher energy efficiency requirements on circuits and systems

Our Previous Work: Software-Hardware Co-design for Energy Efficient NN Inference System

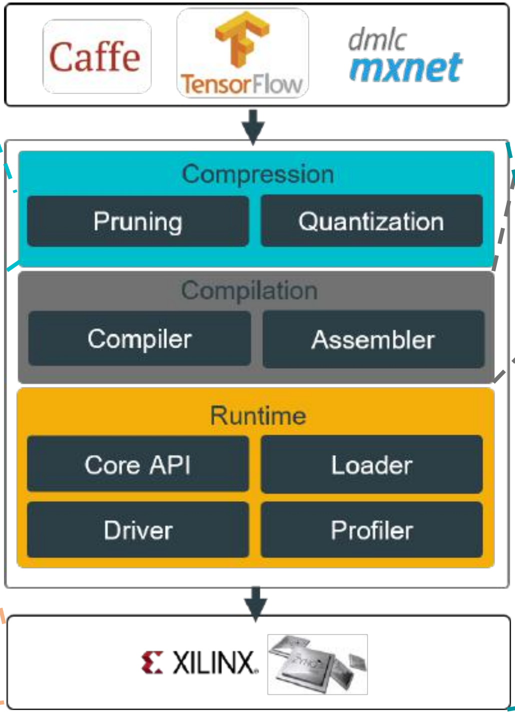
Peak performance ↑+ Utilization ↑ + Workload↓

NN Compression
[FPGA 16][FPGA 17]

Compress the network to 1/10 of the original size with pruning and quantization with negligible accuracy loss.

Hardware Architecture
[FPGA 16][FPGA 17][TCAD 18]

Focus on basic operations like 2D convolution and SpMV. FPGA based design achieves up to 10x better energy efficiency than GPU.



Compiler
[TCAD 19]

Optimize the mapping strategy of NN on the proposed hardware to maximize the runtime efficiency of the computation units.

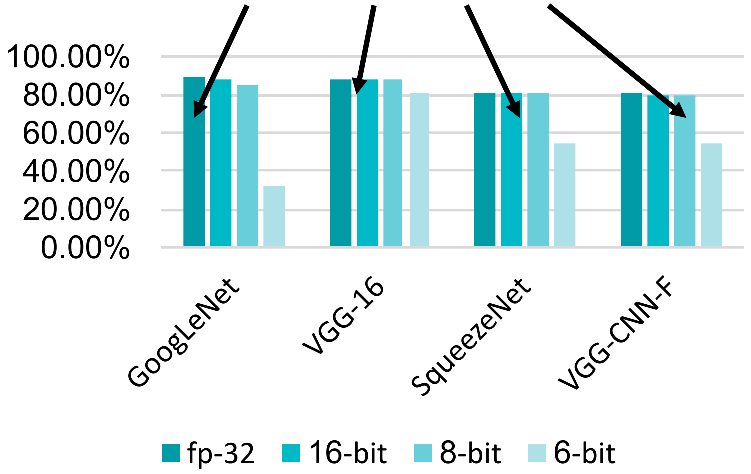
Design Framework
[IEEE MICRO 17][Hotchips 18]

Propose the idea of software-hardware co-design for NN inference system and implement the DNNDK framework.

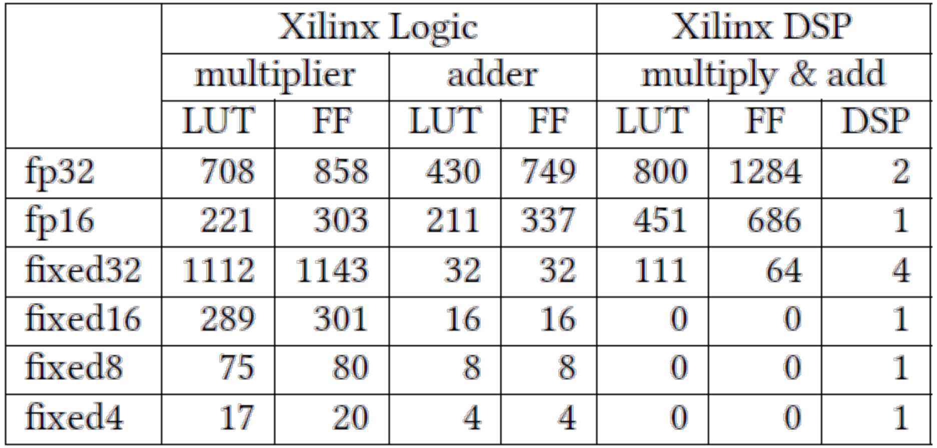
NN Compression: Quantization

Peak performance ↑+ Utilization ↑ + Workload↓

Negligible accuracy loss with
8-bit fixed-point weight & activation



Save FPGA resources



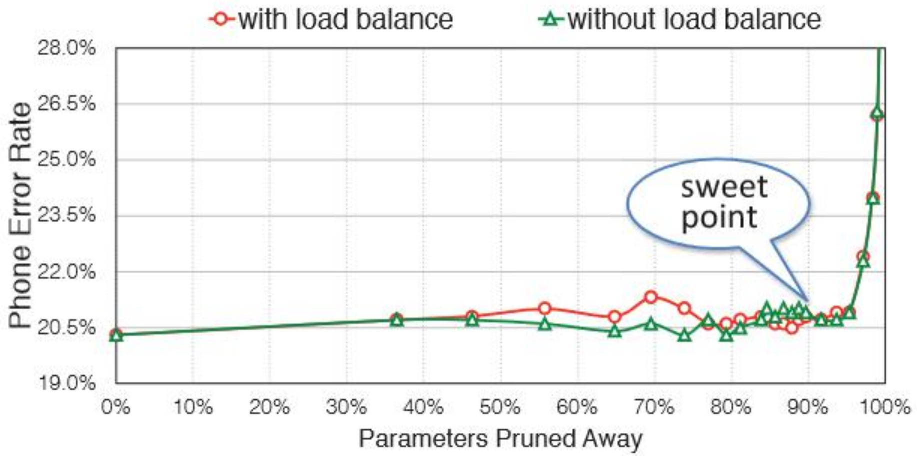
	Xilinx Logic				Xilinx DSP		
	multiplier		adder		multiply & add		
	LUT	FF	LUT	FF	LUT	FF	DSP
fp32	708	858	430	749	800	1284	2
fp16	221	303	211	337	451	686	1
fixed32	1112	1143	32	32	111	64	4
fixed16	289	301	16	16	0	0	1
fixed8	75	80	8	8	0	0	1
fixed4	17	20	4	4	0	0	1

[1] [IEEE MICRO 17] Kaiyuan Guo, Song Han, Song Yao, et al., Software–Hardware Codesign for Efficient Neural Network Acceleration , in IEEE Micro, vol.37, No.2, 2017, pp.18-25.
[2] [IEEE TCAD 18] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Jincheng Yu, Junbin Wang, Song Yao, Song Han, Yu Wang, Huazhong Yang, Angel-Eye: A Complete Design Flow for Mapping CNN onto Embedded FPGA , in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol.37, No.1, 2018, pp.35-47

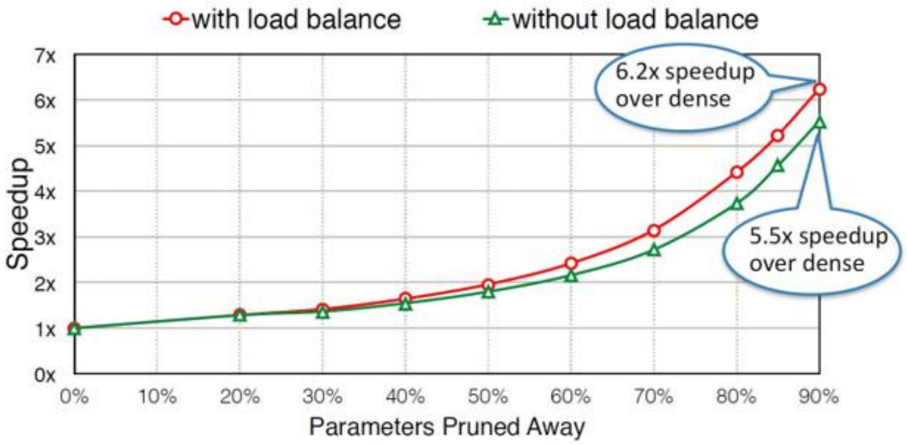
NN Compression: Pruning

Peak performance ↑+ Utilization ↑ + Workload↓

Negligible WER increase with 10% weights



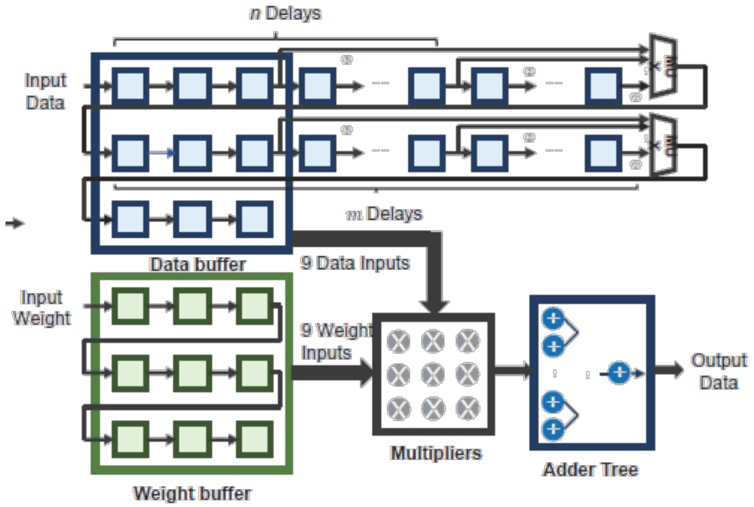
6x acceleration with customized hardware

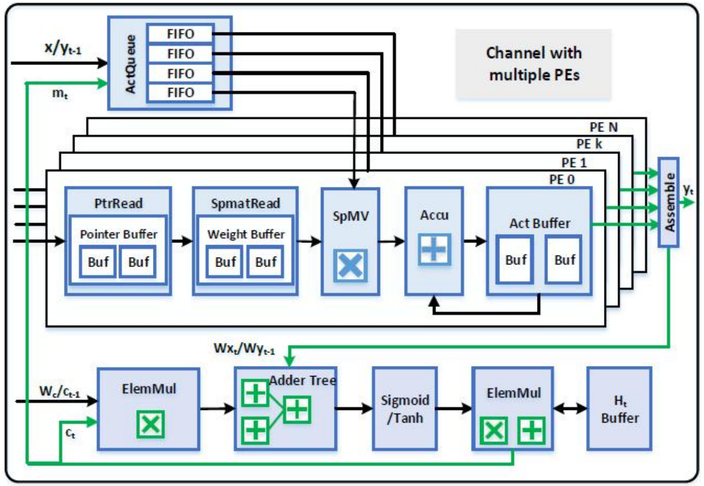


[1] [FPGA 17] Song Han, Junlong Kang, Huizi Mao, et al., ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA , in ACM International Symposium on FPGA, 2017, pp.75-84.

Hardware Architecture – Utilization ↑

- **Dedicate Datapath**: pipelined MAC array and nonlinear operations, SpMV
- **Memory Architecture**: Line buffer for convolution; scratchpad and pointer buffer for sparse memory access





[1] [FPGA 16] Jiantao Qiu, Jie Wang, Song Yao, et al., Going Deeper with Embedded FPGA Platform for Convolutional Neural Network , in ACM International Symposium on FPGA, 2016, pp.26-35.
[2] [FPGA 17] Song Han, Junlong Kang, Huizi Mao, et al., ESE: Efficient Speech Recognition Engine with Compressed LSTM on FPGA , in ACM International Symposium on FPGA, 2017, pp.75-84.

Compiler – Utilization ↑

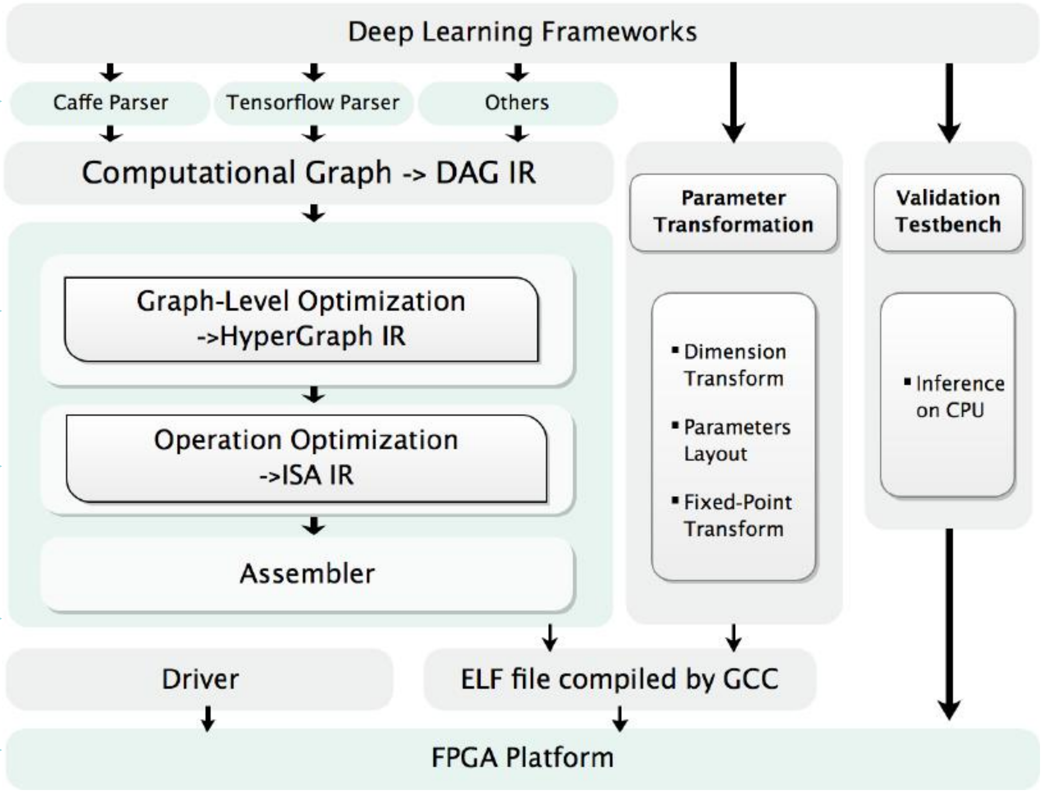
- Directed Acyclic Graph of operations
- Domain Specific Language

- Operation Fusion
- Constant Data Pre-Compute
- Parameter Transform

- Loop Optimization
- On-chip Data Layout

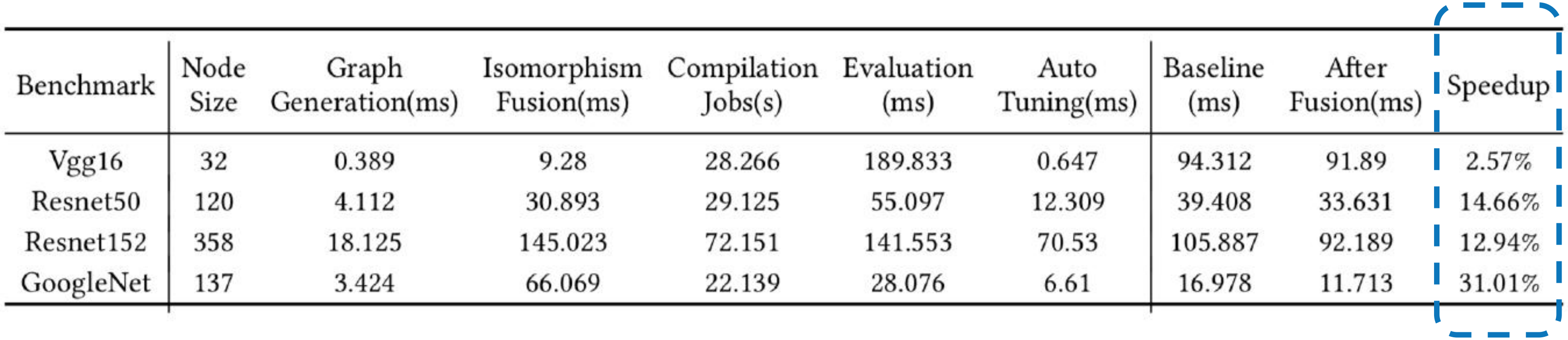
- Ahead-Of-Time
- Mapping to dedicated ISA

- Multiple FPGA platforms with ARM



[1] [TCAD 19] Yu Xing, Shuang Liang, Lingzhi Sui, et al., DNNVM : End-to-End Compiler Leveraging Heterogeneous Optimizations on FPGA-based CNN Accelerators , in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2019

Compiler

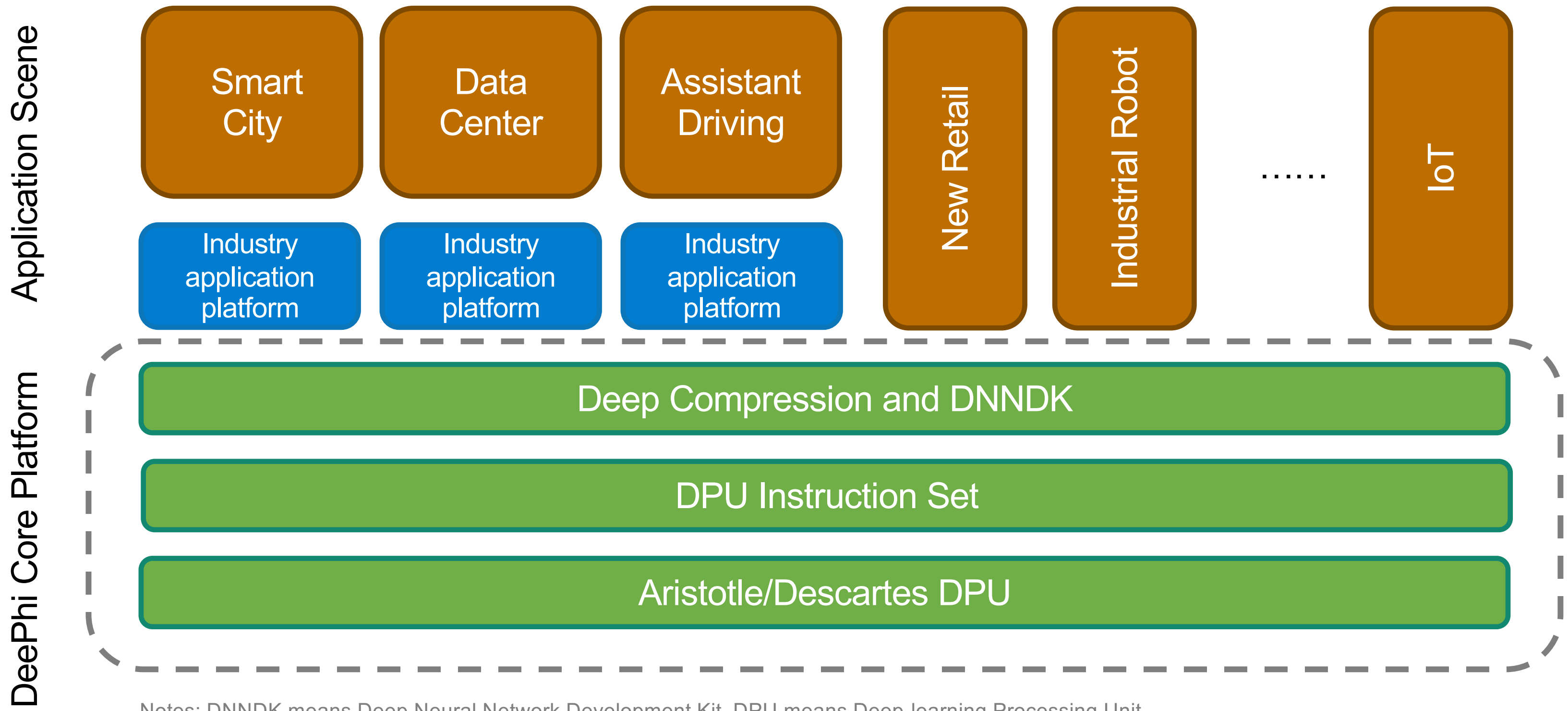


Benchmark	Node Size	Graph Generation(ms)	Isomorphism Fusion(ms)	Compilation Jobs(s)	Evaluation (ms)	Auto Tuning(ms)	Baseline (ms)	After Fusion(ms)	Speedup
Vgg16	32	0.389	9.28	28.266	189.833	0.647	94.312	91.89	2.57%
Resnet50	120	4.112	30.893	29.125	55.097	12.309	39.408	33.631	14.66%
Resnet152	358	18.125	145.023	72.151	141.553	70.53	105.887	92.189	12.94%
GoogleNet	137	3.424	66.069	22.139	28.076	6.61	16.978	11.713	31.01%

- Up to 30% speedup with optimized scheduling
- Low compilation time overhead: < 100s for each network

DeePhi Tech: From Academy to Industry

- Aims to provide efficient, convenient & economic DL platform for AI applications

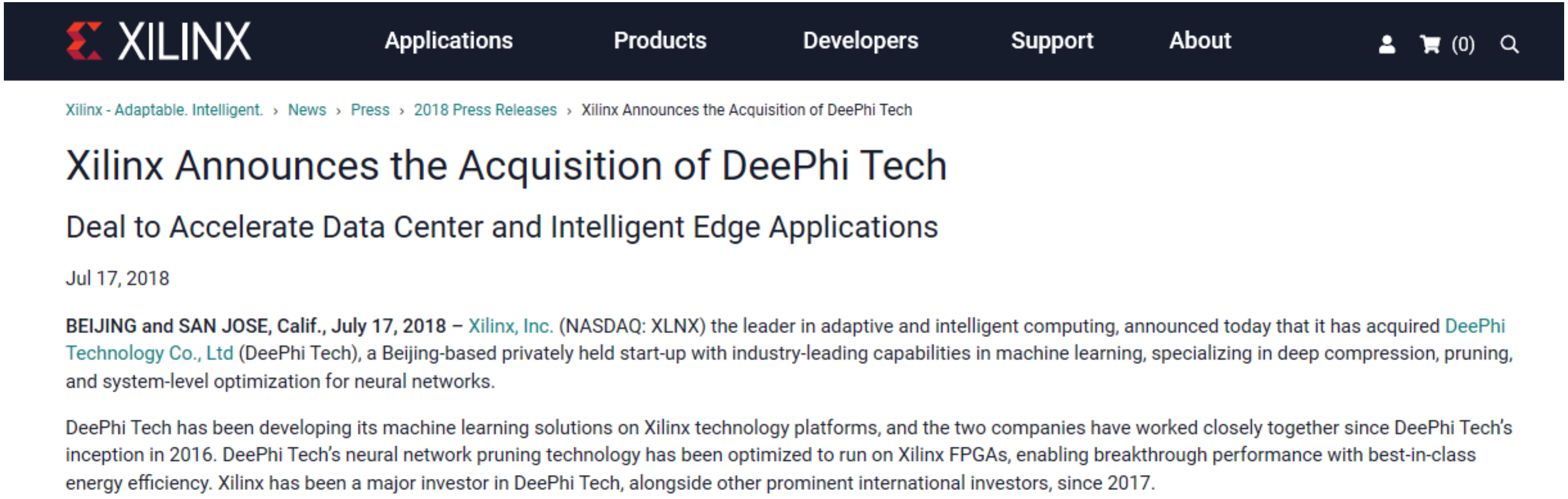


Notes: DNNDK means Deep Neural Network Development Kit, DPU means Deep-learning Processing Unit

DEEPHi
深 鉴 科 技
Discovering the Philosophy behind
Deep Learning Computing

Now
Part of

XILINX®



Xilinx - Adaptable. Intelligent. › News › Press › 2018 Press Releases › Xilinx Announces the Acquisition of DeePhi Tech

Xilinx Announces the Acquisition of DeePhi Tech

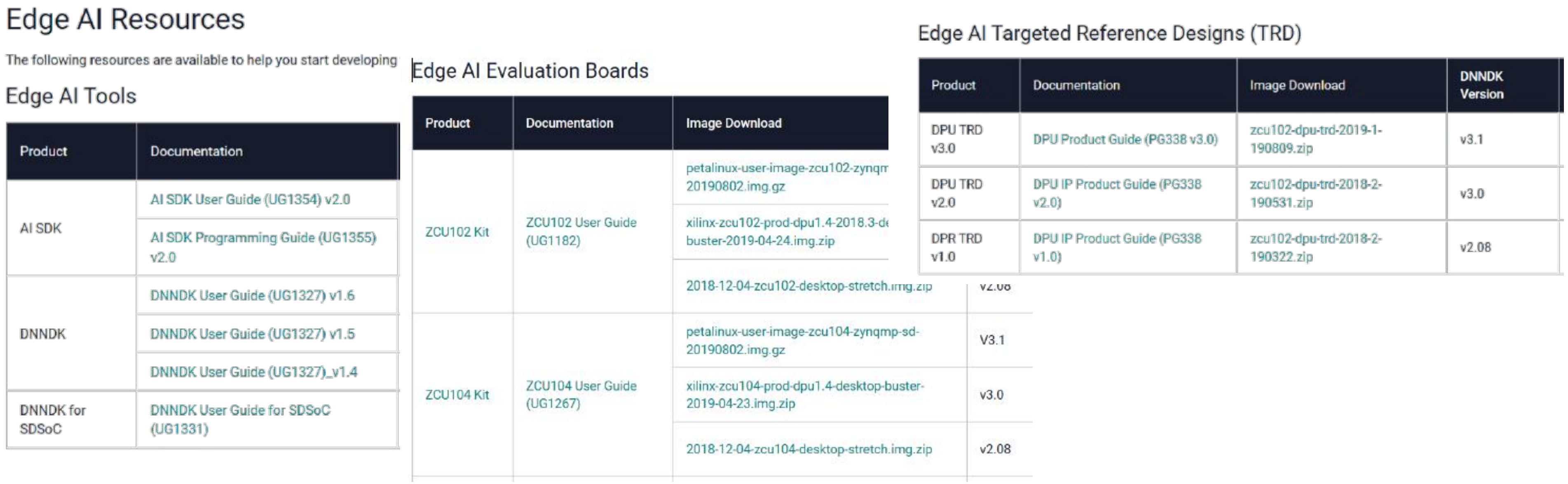
Deal to Accelerate Data Center and Intelligent Edge Applications

Jul 17, 2018

BEIJING and SAN JOSE, Calif., July 17, 2018 – Xilinx, Inc. (NASDAQ: XLNX) the leader in adaptive and intelligent computing, announced today that it has acquired DeePhi Technology Co., Ltd (DeePhi Tech), a Beijing-based privately held start-up with industry-leading capabilities in machine learning, specializing in deep compression, pruning, and system-level optimization for neural networks.

DeePhi Tech has been developing its machine learning solutions on Xilinx technology platforms, and the two companies have worked closely together since DeePhi Tech's inception in 2016. DeePhi Tech's neural network pruning technology has been optimized to run on Xilinx FPGAs, enabling breakthrough performance with best-in-class energy efficiency. Xilinx has been a major investor in DeePhi Tech, alongside other prominent international investors, since 2017.

DPU and DNNDK available now!



Edge AI Resources

The following resources are available to help you start developing

Edge AI Tools

Product	Documentation
AI SDK	AI SDK User Guide (UG1354) v2.0
	AI SDK Programming Guide (UG1355) v2.0
DNNDK	DNNDK User Guide (UG1327) v1.6
	DNNDK User Guide (UG1327) v1.5
	DNNDK User Guide (UG1327)_v1.4
DNNDK for SDSoC	DNNDK User Guide for SDSoC (UG1331)

Edge AI Evaluation Boards

Product	Documentation	Image Download	
ZCU102 Kit	ZCU102 User Guide (UG1182)	petalinux-user-image-zcu102-zynqm 20190802.img.gz	
		xilinx-zcu102-prod-dpu1.4-2018.3-de buster-2019-04-24.img.zip	
		2018-12-04-zcu102-desktop-stretch.img.zip	v2.08
ZCU104 Kit	ZCU104 User Guide (UG1267)	petalinux-user-image-zcu104-zynqmp-sd-20190802.img.gz	V3.1
		xilinx-zcu104-prod-dpu1.4-desktop-buster-2019-04-23.img.zip	v3.0
		2018-12-04-zcu104-desktop-stretch.img.zip	v2.08

Edge AI Targeted Reference Designs (TRD)

Product	Documentation	Image Download	DNNDK Version
DPU TRD v3.0	DPU Product Guide (PG338 v3.0)	zcu102-dpu-trd-2019-1-190809.zip	v3.1
DPU TRD v2.0	DPU IP Product Guide (PG338 v2.0)	zcu102-dpu-trd-2018-2-190531.zip	v3.0
DPR TRD v1.0	DPU IP Product Guide (PG338 v1.0)	zcu102-dpu-trd-2018-2-190322.zip	v2.08

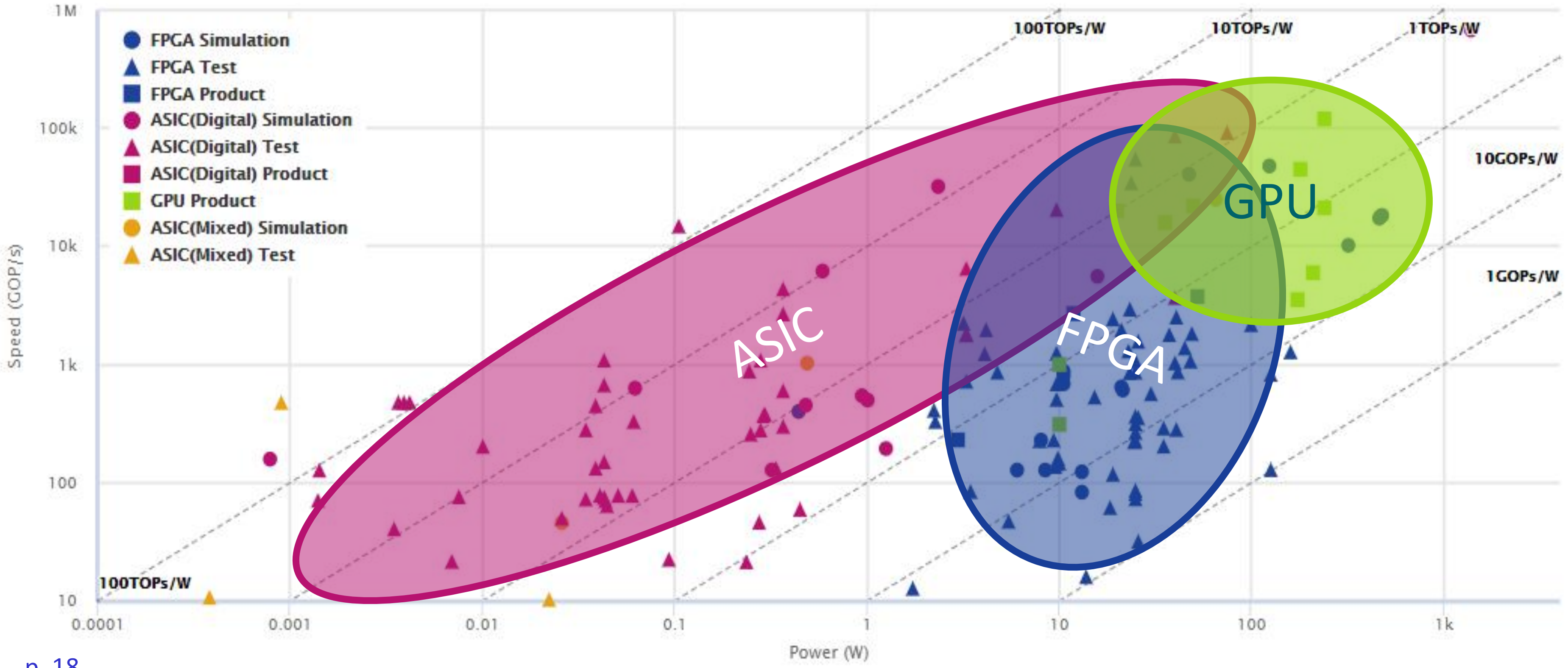
DNNDK (Tool) + Ready to use image (DPU on boards) + Targeted Reference Design

https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge

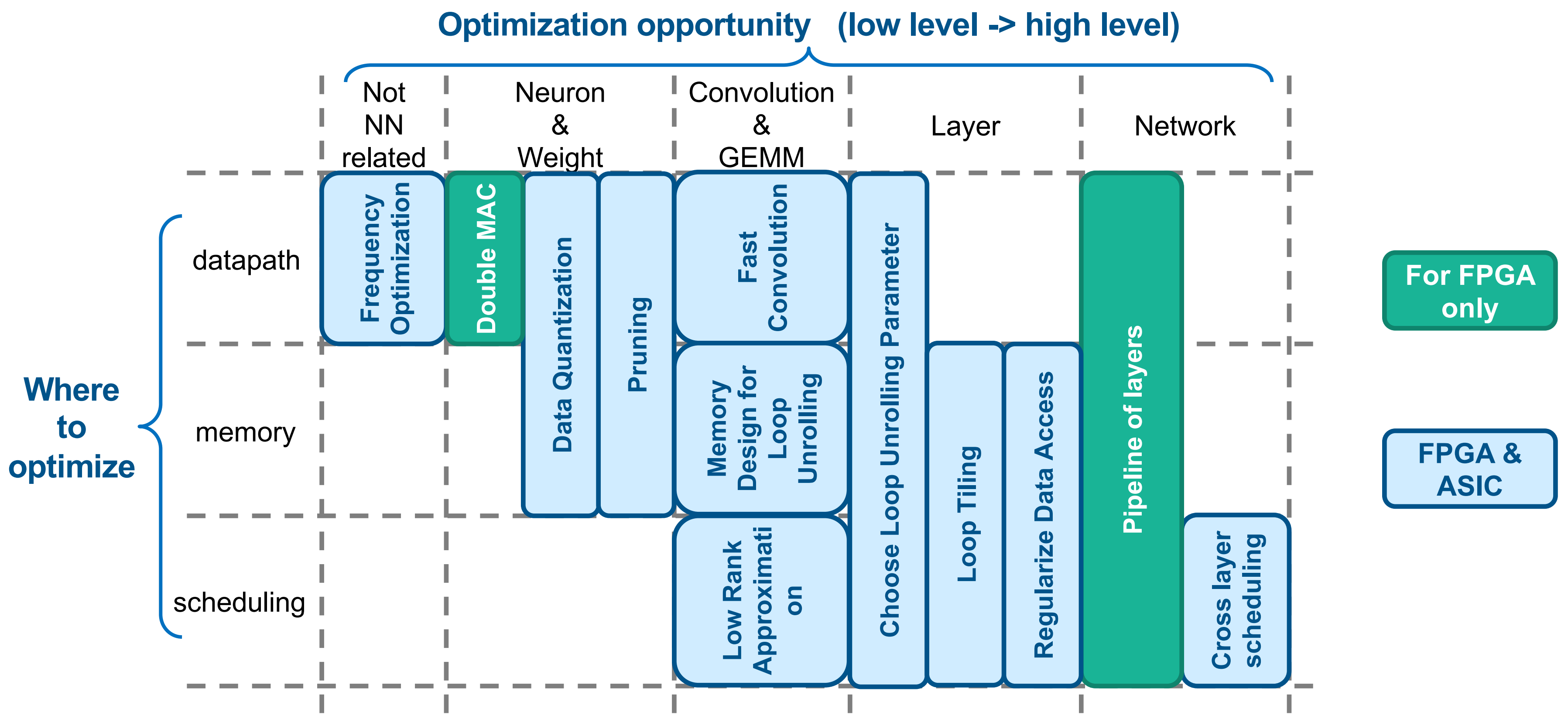
Taking one step back to see the **bigger picture**

Academic NN Accelerators (Performance vs Power)

https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

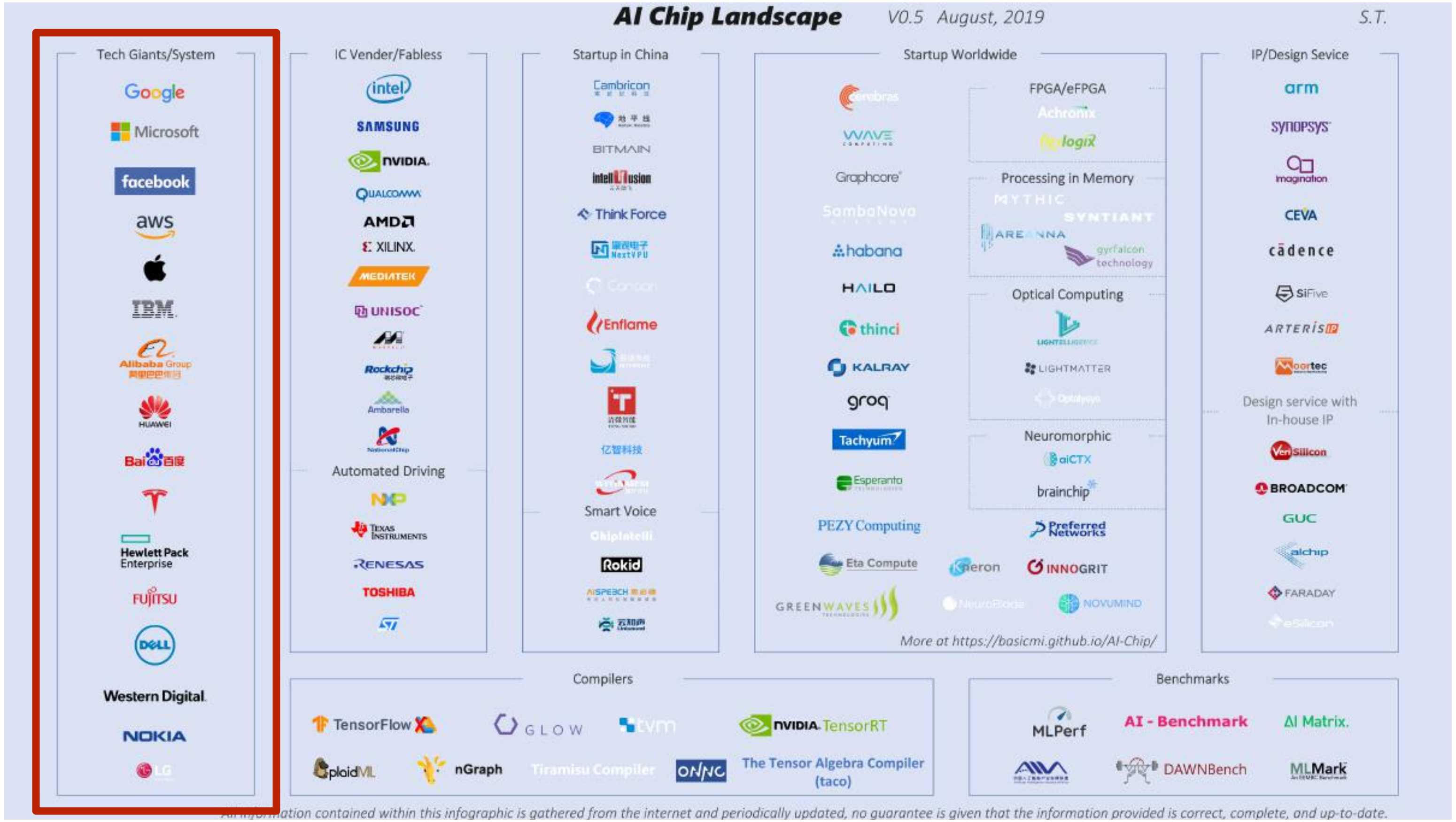


Survey on FPGA based Inference Accelerators



[1] [TRETS 19] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang and Huazhong Yang, A Survey of FPGA-Based Neural Network Inference Accelerator , in ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol.12, No.1, 2019.

Industry is Crowding into this Region


AI Chip Landscape V0.5 August, 2019 S.T.
Tech Giants/System
IC Vender/Fabless
Startup in China
Startup Worldwide
IP/Design Sevice
FPGA/eFPGA
Processing in Memory
Optical Computing
Neuromorphic
Automated Driving
Smart Voice
Design service with In-house IP
Compilers
Benchmarks
More at https://basicmi.github.io/AI-Chip/
All information contained within this infographic is gathered from the internet and periodically updated, no guarantee is given that the information provided is correct, complete, and up-to-date.

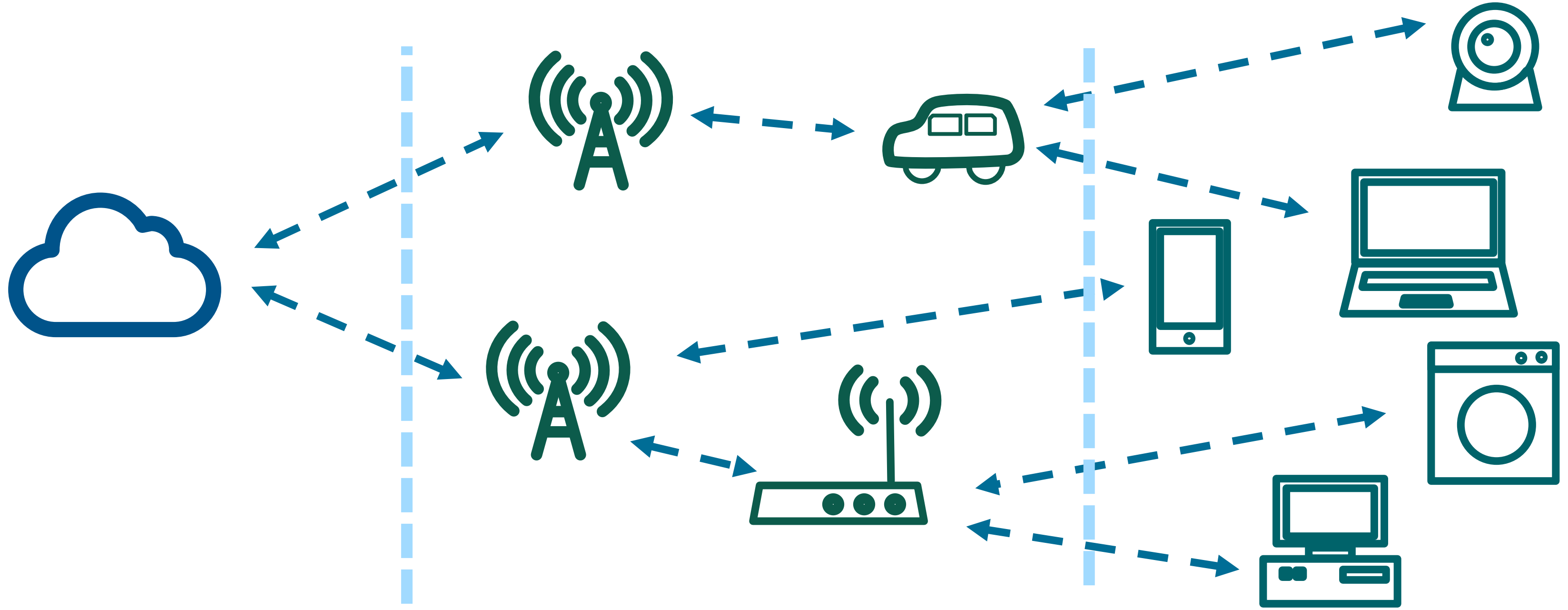
https://basicmi.github.io/AI-Chip/

Application Scenarios: Cloud, Edge, Terminal

Cloud
Server Cluster

Edge
Base station, Vehicle, Router

Terminal
Mobile Devices, Terminal of IOT



Application Scenarios: Cloud, Edge, Terminal

Data Management ⟺

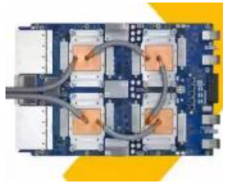
Action Display ⟺

Cloud
Computing + Storage

Edge
Computing + Communication

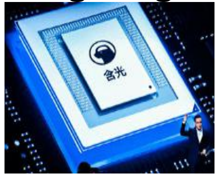
Terminal
Display + Interaction

Google TPU 3.0



Huawei Ascend

Alibaba Hanguang800



Xilinx DPU

Amba CV2

Rockchip

Intel SpringCrest

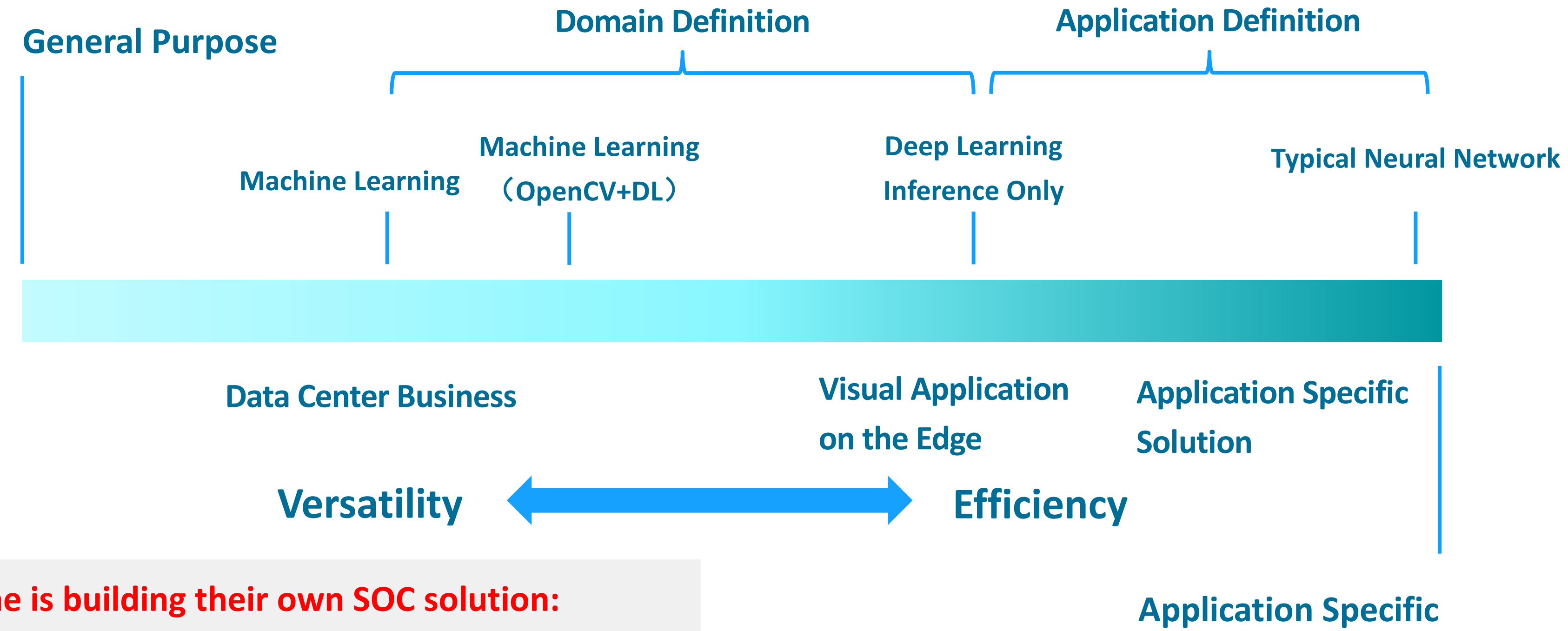
Xilinx ACAP Versal

Nvidia TX2

Mobileye EyeQ4

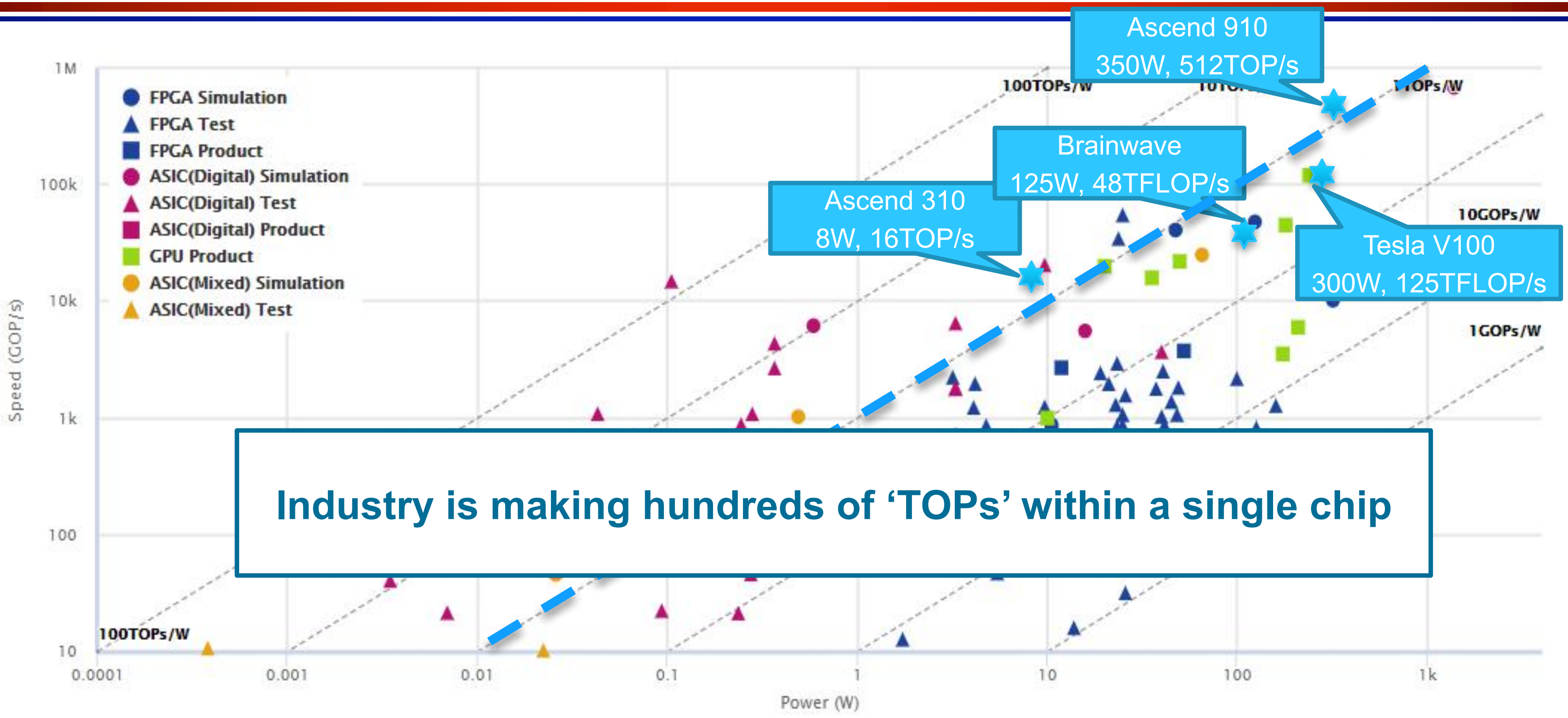
MOBILEYE An Intel Company

Choose Your Business Scope: AI is a Large Area



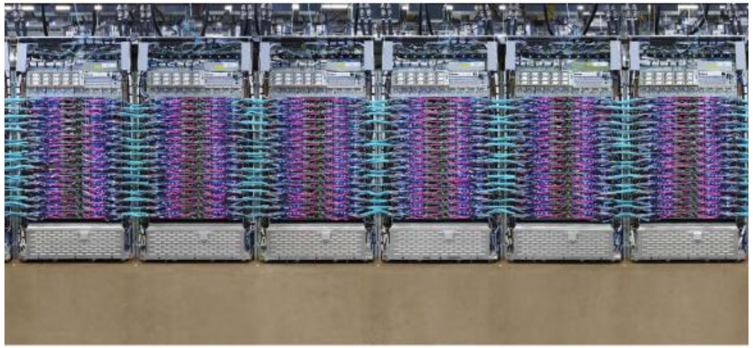
Everyone is building their own SOC solution:
Manufacturing + IP + Tools + Integration Technology

Growing of Computation Power

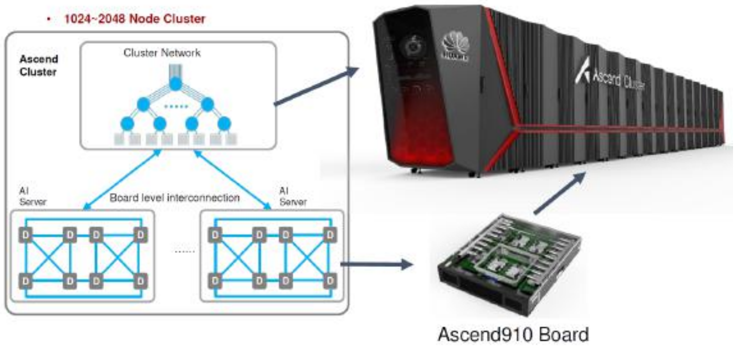


Industry is making hundreds of 'TOPs' within a single chip

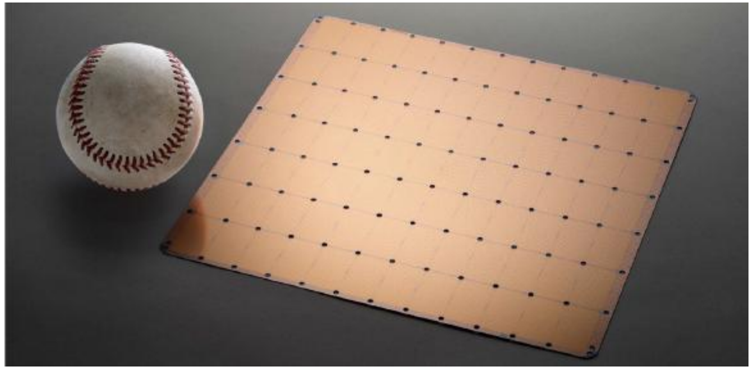
Even Larger?



TPU v3 pod: >100PFLOPS

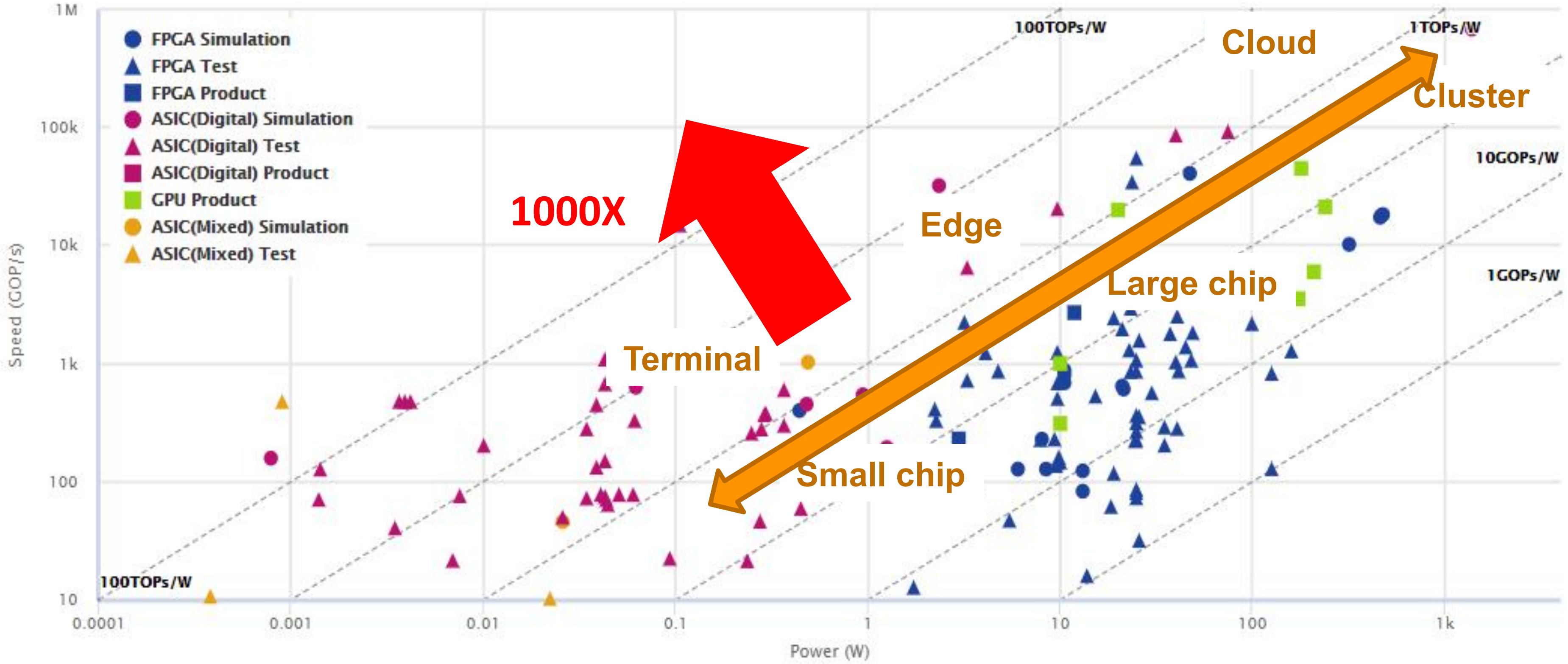


Ascend 910 Cluster: 512PFLOPS



Cerebras: Wafer-scale chip with 1.4T transistors

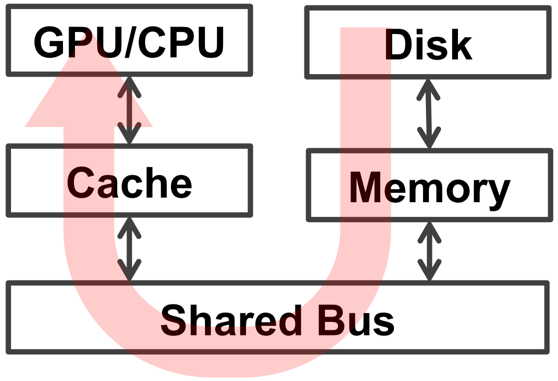
We Already Have Many Choices Now



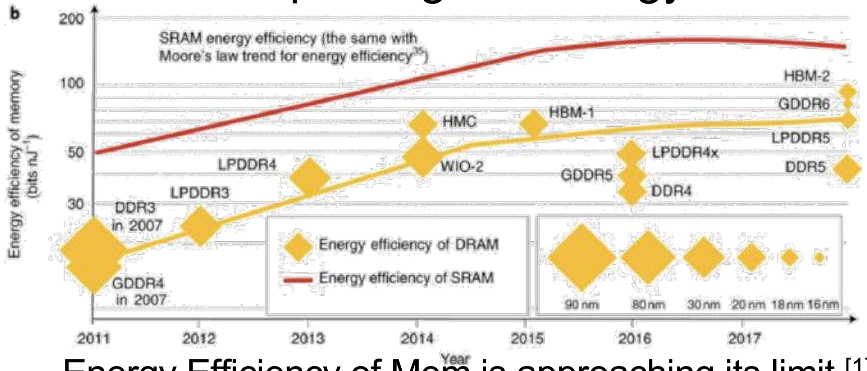
Bottleneck of Energy Efficiency Improvement

- ***Data movements*** become the main bottleneck for improving the energy efficiency

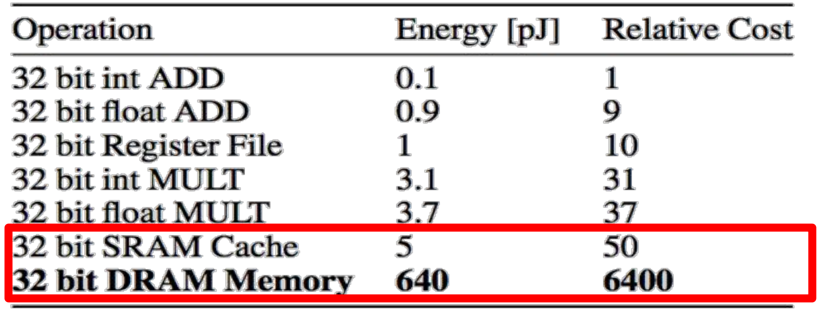
Large data movements in AI algorithms -> Aggravated Von Neumann bottleneck



Von Neumann Architecture



Energy Efficiency of Mem is approaching its limit [1]

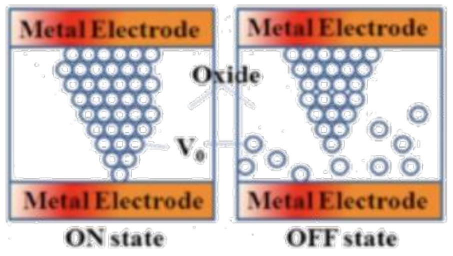


Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**

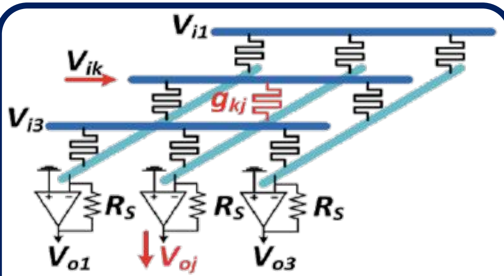
High Memory Access Energy Consumption [2]

[1] Xu, Xiaowei, et al. "Scaling for edge inference of deep neural networks." Nature Electronics 1.4 (2018): 216.
[2] Han, Song, et al. "Learning both weights and connections for efficient neural network." *Advances in NIPS*. 2015.

Non-Volatile Memory (NVM) - based CNN Accelerator



NVM Device

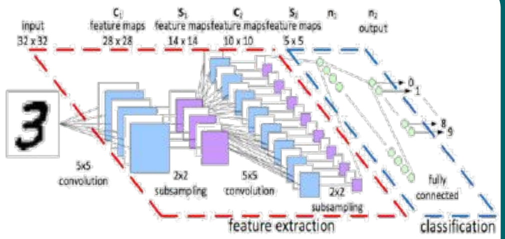


Crossbar Structure

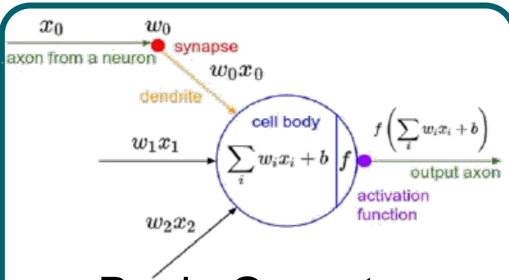
$$V_{oj} = \sum_k V_{ik} \cdot c_{kj}\, , c_{kj} = -\frac{g_{kj}}{g_s}$$

Kirchhoff Law

MVM-Form



CNN Algorithm



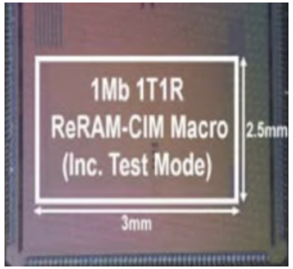
Basic Operator

$$f_o(x,y,z) = \sum_{i=0}^{K-1}\sum_{j=0}^{K-1}\sum_{k=1}^{C_{in}} f_i(x+i,y+j,k)k_z(i,j,k)$$

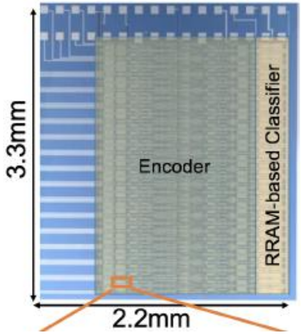
Mathematical Expression

NVM-based CNN Accelerator

Academic



- RRAM chip, NTHU, ISSCC19
- 256 rows x 512 columns
- CNN computing: 11.75~14.6ns
- Energy Efficiency:
  - **21.9~53.17 TOPS/W**



- CNFET+ RRAM chip, Stanford, ISSCC18
- 1952 CNFETs + 224 RRAM cells
- Pairwise classification of 21 European Languages: 98% accuracy

[1] Xue, Cheng-Xin, et al. "24.1 A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6 ns Parallel MAC Computing Time for CNN Based AI Edge Processors." 2019 IEEE ISSCC
[2] Wu, Tony F., et al. "Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: Hyperdimensional computing case study." 2018 IEEE ISSCC

Industry

MYTHIC

- Based on embedded NOR FLASH
- Highest energy efficiency: 4TOPS/W
- https://www.mythic-ai.com/technology/

WITINMEM 知存科技

- Based on NOR Flash

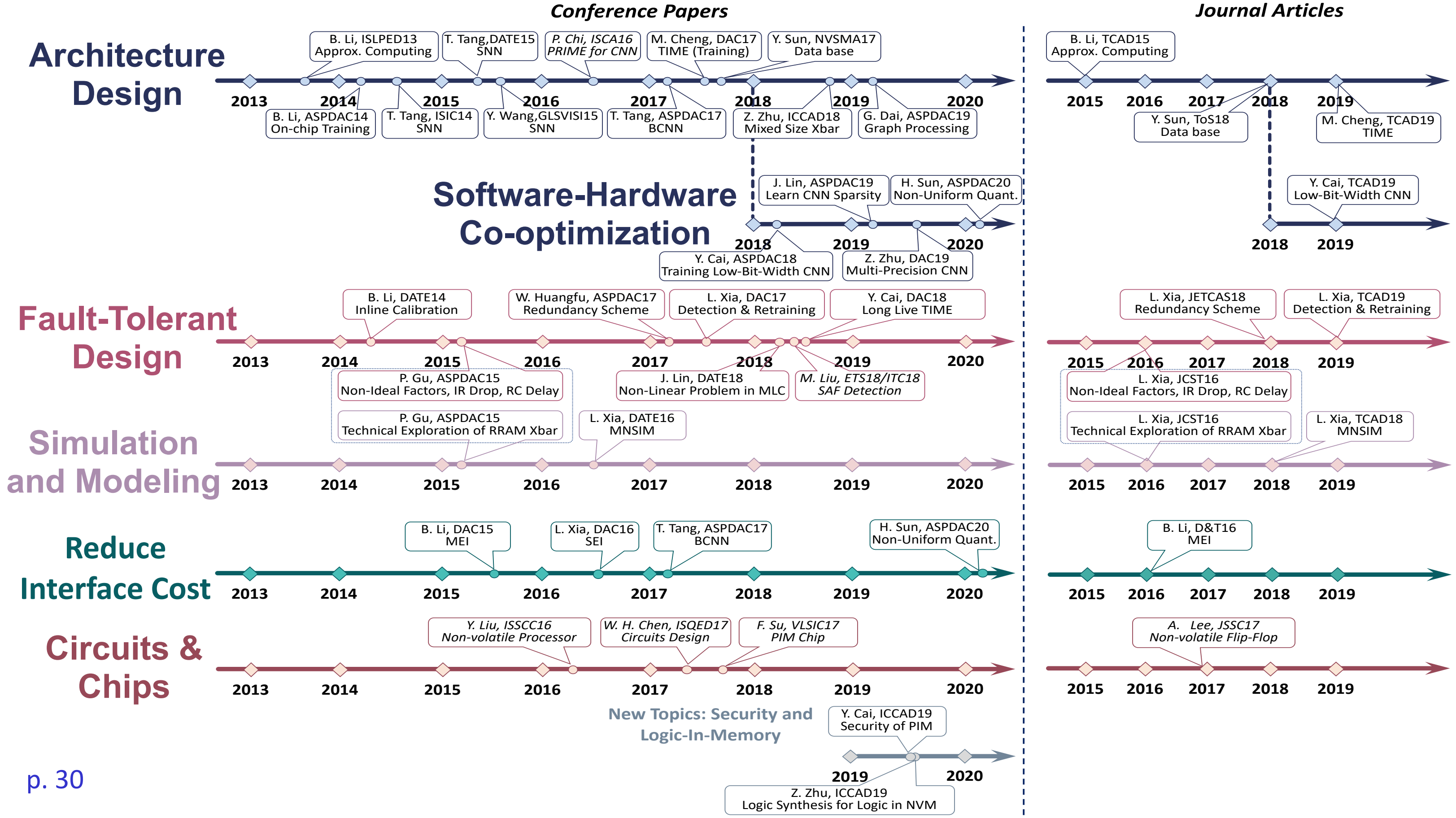
SYNTIANT

- Based on NOR Flash
- Built for voice applications

IBM®

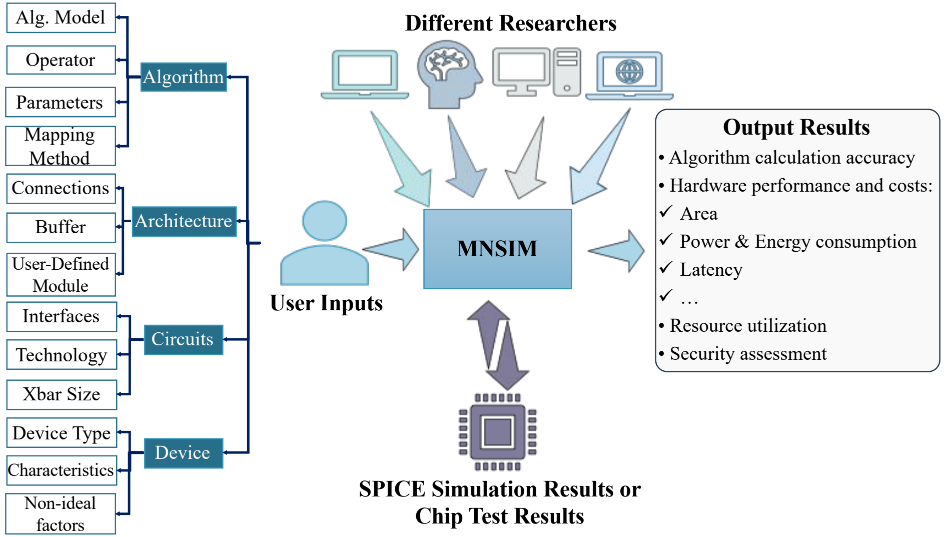
- Based on PCM
- 8-bit precision device (still in research stage)
- IEDM 2018

Research on RRAM-based PIM

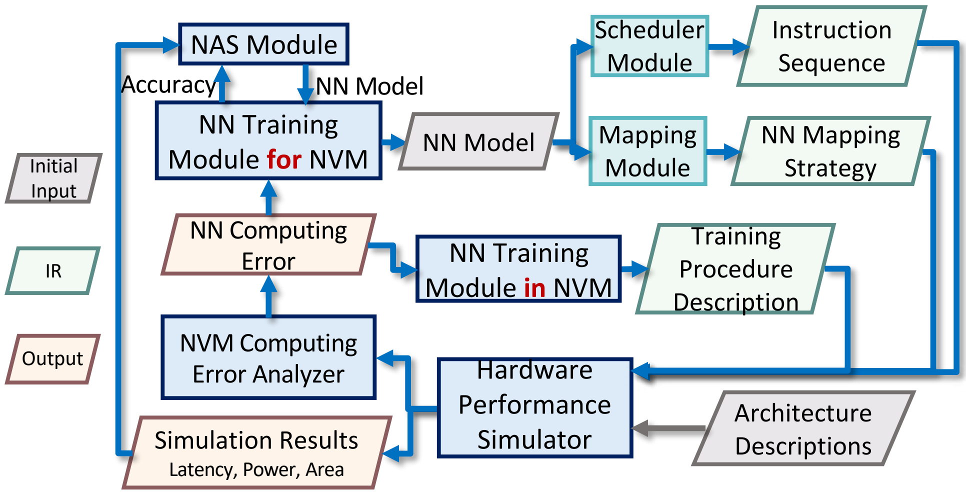


Conference Papers
Journal Articles
Architecture Design
B. Li, ISLPED13 Approx. Computing
T. Tang, DATE15 SNN
P. Chi, ISCA16 PRIME for CNN
M. Cheng, DAC17 TIME (Training)
Y. Sun, NVSMA17 Data base
B. Li, ASPDAC14 On-chip Training
T. Tang, ISIC14 SNN
Y. Wang, GLSVISI15 SNN
T. Tang, ASPDAC17 BCNN
Z. Zhu, ICCAD18 Mixed Size Xbar
G. Dai, ASPDAC19 Graph Processing
B. Li, TCAD15 Approx. Computing
Y. Sun, ToS18 Data base
M. Cheng, TCAD19 TIME
Software-Hardware Co-optimization
J. Lin, ASPDAC19 Learn CNN Sparsity
H. Sun, ASPDAC20 Non-Uniform Quant.
Y. Cai, ASPDAC18 Training Low-Bit-Width CNN
Z. Zhu, DAC19 Multi-Precision CNN
Y. Cai, TCAD19 Low-Bit-Width CNN
Fault-Tolerant Design
B. Li, DATE14 Inline Calibration
W. Huangfu, ASPDAC17 Redundancy Scheme
L. Xia, DAC17 Detection & Retraining
Y. Cai, DAC18 Long Live TIME
P. Gu, ASPDAC15 Non-Ideal Factors, IR Drop, RC Delay
J. Lin, DATE18 Non-Linear Problem in MLC
M. Liu, ETS18/ITC18 SAF Detection
L. Xia, JETCAS18 Redundancy Scheme
L. Xia, TCAD19 Detection & Retraining
L. Xia, JCST16 Non-Ideal Factors, IR Drop, RC Delay
Simulation and Modeling
P. Gu, ASPDAC15 Technical Exploration of RRAM Xbar
L. Xia, DATE16 MNSIM
L. Xia, JCST16 Technical Exploration of RRAM Xbar
L. Xia, TCAD18 MNSIM
Reduce Interface Cost
B. Li, DAC15 MEI
L. Xia, DAC16 SEI
T. Tang, ASPDAC17 BCNN
H. Sun, ASPDAC20 Non-Uniform Quant.
B. Li, D&T16 MEI
Circuits & Chips
Y. Liu, ISSCC16 Non-volatile Processor
W. H. Chen, ISQED17 Circuits Design
F. Su, VLSIC17 PIM Chip
A. Lee, JSSC17 Non-volatile Flip-Flop
New Topics: Security and Logic-In-Memory
Y. Cai, ICCAD19 Security of PIM
Z. Zhu, ICCAD19 Logic Synthesis for Logic in NVM
2013
2014
2015
2016
2017
2018
2019
2020

MNSIM: High-Speed Behavior-Level Model for Design Optimization



The overview of MNSIM



The entire modeling flow

The code is available in:
https://github.com/Zhu-Zhenhua/MNSIM_Python

Brief Summary

What we have or we can expect

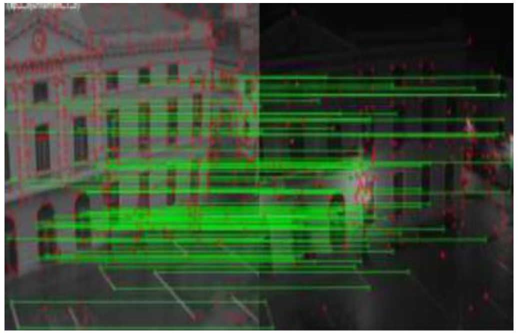
- AI platforms are covering everywhere.
- We already have many methods to make the chips powerful.

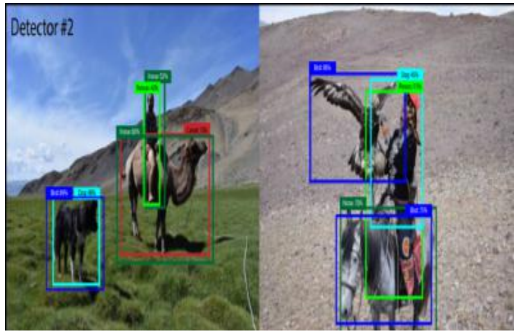
What's next: how they will influence the hardware and software design for NN on chip

- Are the chips ready for real applications?
- Can we use the large 'TOPs' to the extreme?
- Are the systems robust enough?
- Are the systems secure enough?

Case Study 1:
Using CNN accelerator in moving robots

CNN Greatly Benefits Basic Functions in Robotic Applications





Feature-Point Extraction

SuperPoint [1] :

The matching accuracy is **20%** higher than handcrafted method.

Place Recognition

GeM [2] :

The recall of image query is **10%** higher than handcrafted method.

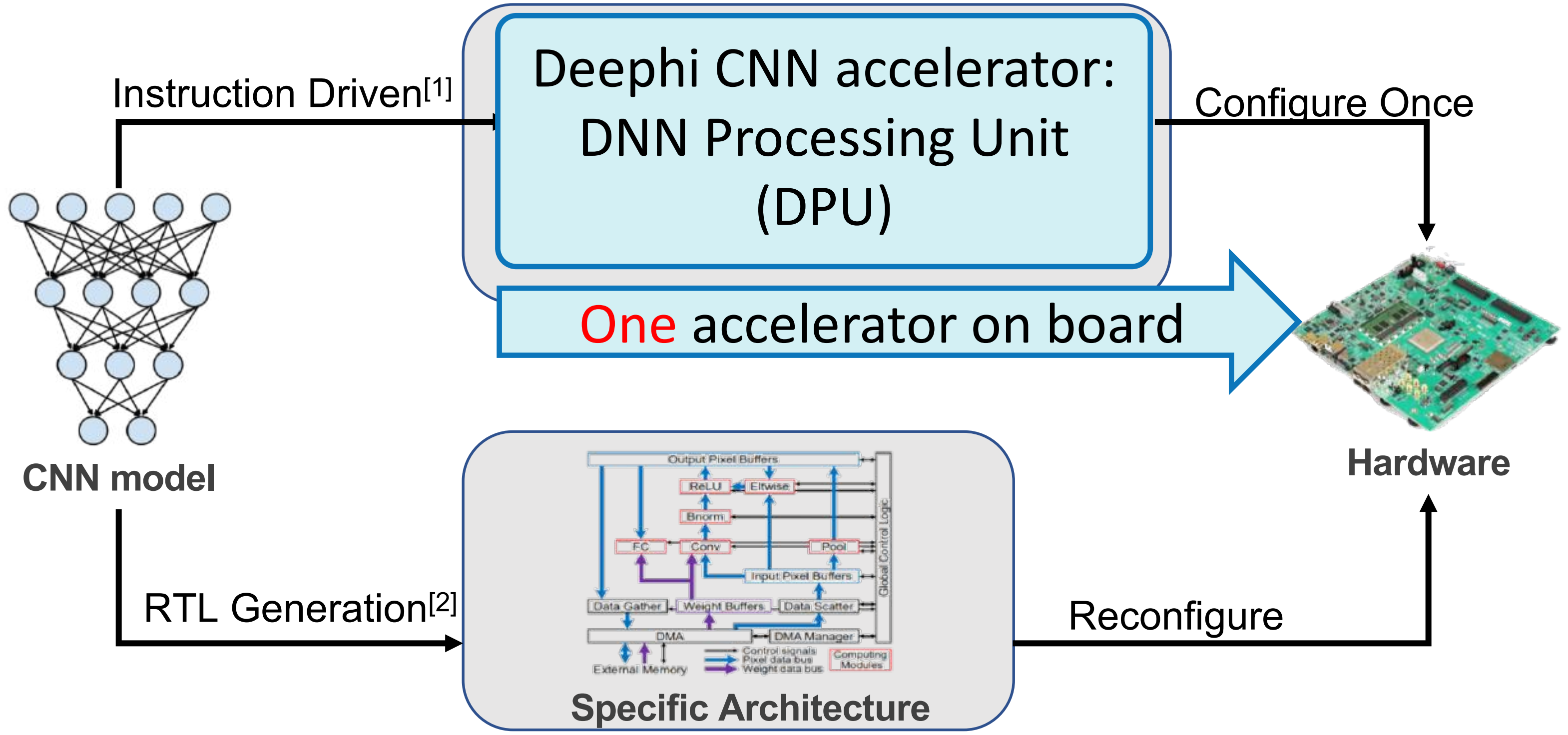
Object Detection

YOLOv3 [3] :

The mean average precision (mAP) on COCO is above **58%.**

[1] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. 2018. Superpoint: Self-supervised interest point detection and description. In CVPR Workshops. 224–236.
[2] Filip Radenović, Giorgos Tolias, and Ondřej Chum. 2018. Fine-tuning CNN image retrieval with no human annotation. IEEE transactions on pattern analysis and machine intelligence 41, 7 (2018), 1655–1668.
[3] Redmon, J., & Farhadi, A. (2018). Yolov3: an incremental improvement. arXiv. 1804.02767.

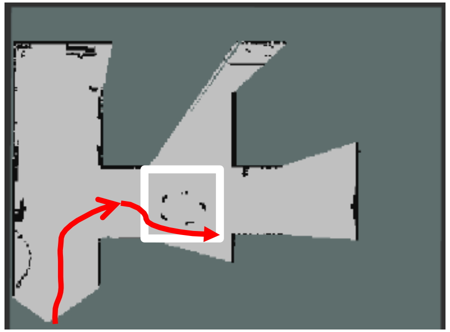
Accelerators Enables CNN on Moving Robots



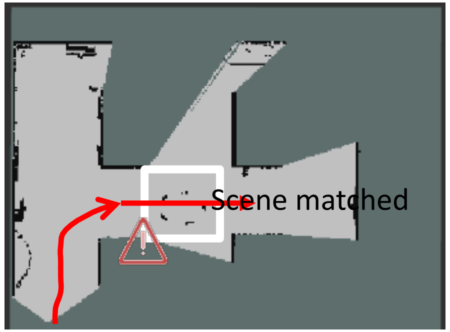
[1] [FPT 17] Jincheng Yu, Yiming Hu, Xuefei Ning, Jiantao Qiu, Kaiyuan Guo, Yu Wang, and Huazhong Yang. 2017. Instruction driven cross-layer CNN accelerator with winograd transformation on FPGA. In International Conference on Field Programmable Technology. 227–230
[2] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks. In Field Programmable Logic and Applications (FPL), 2017 27th International Conference on. IEEE, 1–8

Tasks with Hard Deadline

Visual Odometry (VO):
Basic for localization & obstacle detection

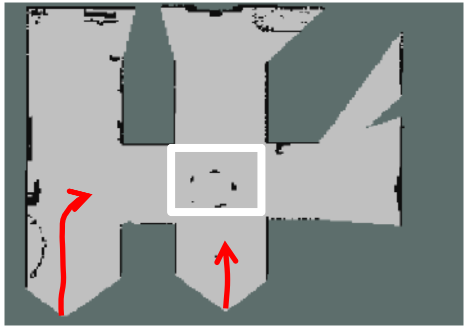


Obstacle detected on time : Circumvention



Obstacle detected not on time : Collision

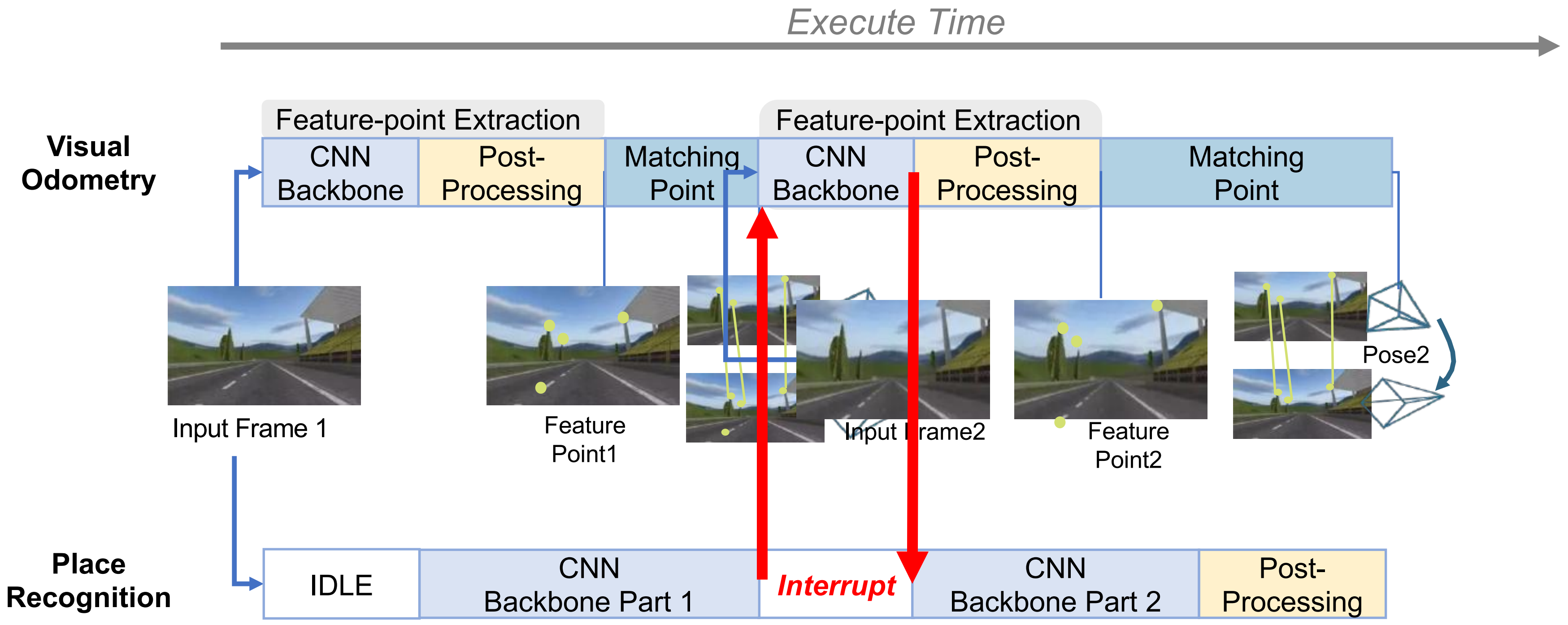
Place Recognition (PR):
For map merging & optimization



Map merging can be delayed

Some tasks need to be finished before deadline

Accelerator Interrupt for Hardware Conflicts



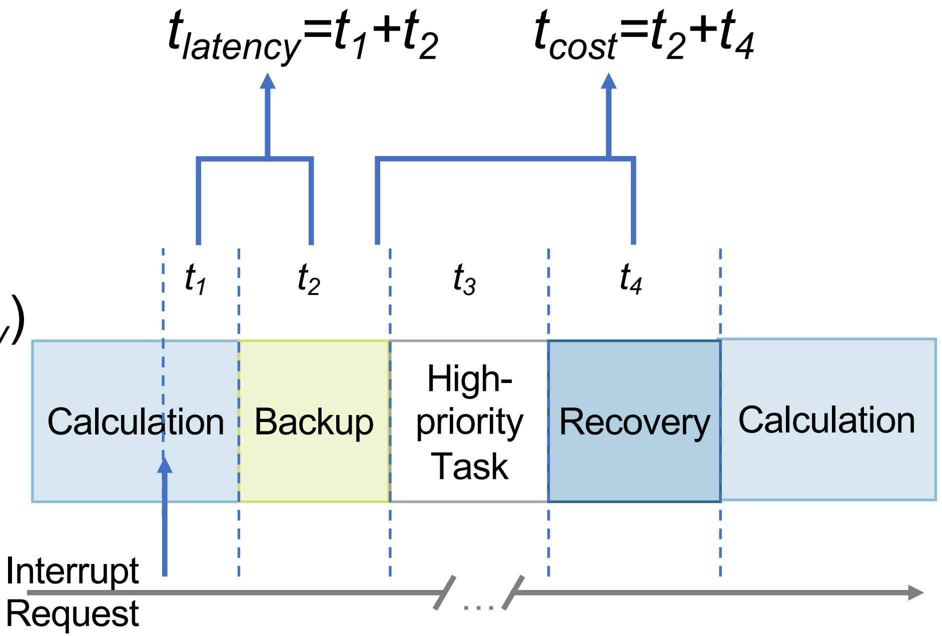
If we only have ONE DPU, how to share between two threads?

Interrupt Respond Latency & Extra Cost

- 4 Stages for Interrupt
  - To finish current instruction (*t1*)
  - To backup low-priority task (*t2*)
  - To execute high-priority task (*t3*)
  - To recovery low-priority task (*t4*)

- Interrupt Respond Latency ($t_{latency}$)
  - $t_{latency}=t_1+t_2$

- Extra Cost (t_{cost})
  - $t_{cost}=t_2+t_4$



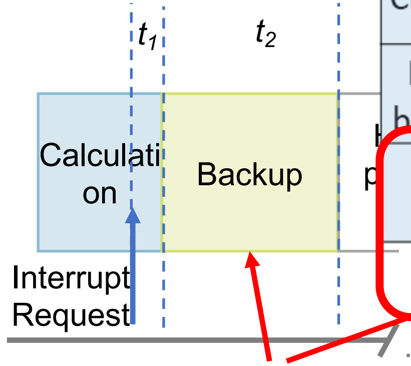
How to Interrupt?

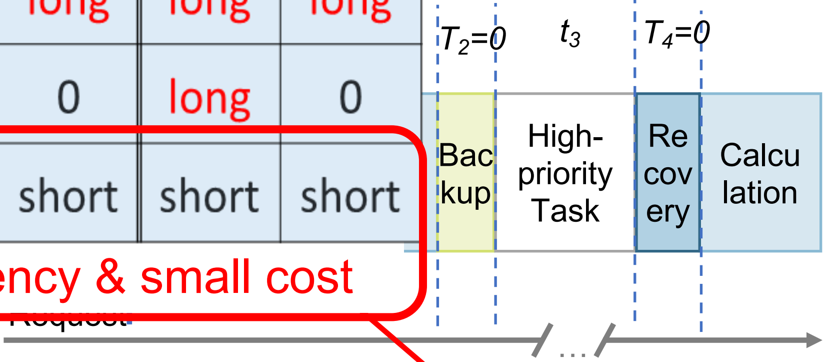
- Interrupt Intermediately (CPU-like)
  - Same handle
  - Short *t1*. Lon

- Interrupt After Each layer (Layer-by-layer)
  - (*t1*) & Recovery (*t2*)

$t_{latency}=t_1+t_2$				$t_{cost}=t_2+t_4$		
	t1	*t2*	*t3*	*t4*	$t_{latency}$	t_{cost}
CPU-like	short	long	-	long	long	long
Layer-by-layer	long	0	-	0	long	0
Ours	short	short	-	short	short	short

We want both short latency & small cost



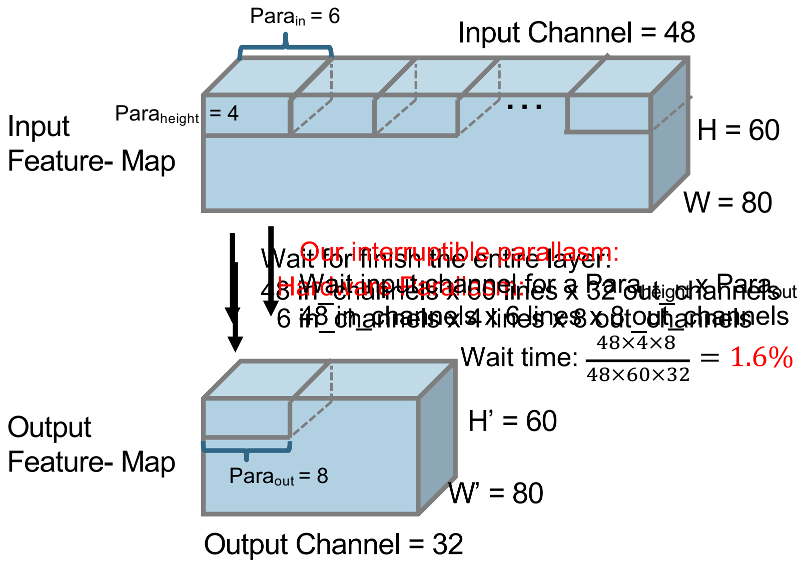


Backup/Recovery All on-chip "caches"

Wait for finishing the entire layer

Virtual-Instruction-Based Interrupt

- Interruptible at some positions with small extra cost
- Different handler for different interrupt position
  - Some Virtual-instructions are inserted into the instruction sequence.

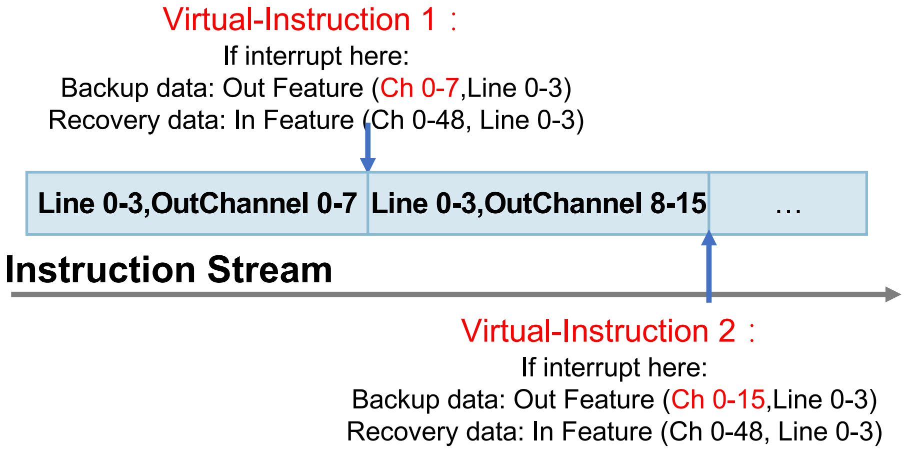


Wait for finish the entire layer:
48 in_channels x 60 lines x 32 out_channels

Hardware Parallelism:
6 in_channels x 4 lines x 8 out_channels

Our interruptible parallasm:
- Wait input channel for a $Para_{height}$ x $Para_{out}$:
48 in_channels x 4 lines x 8 out_channels

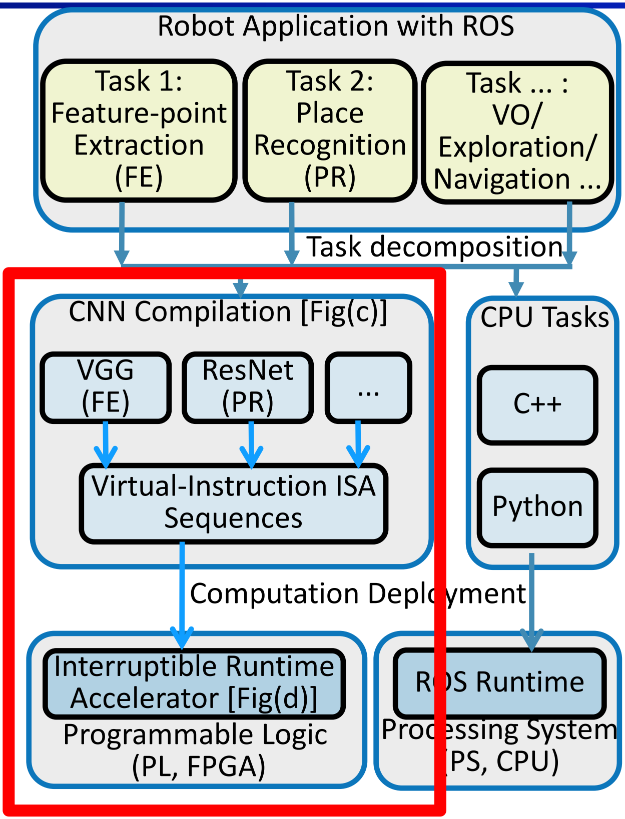
Wait time: $\frac{48\times4\times8}{48\times60\times32} = 1.6\%$



Virtual-Instruction 1 :
If interrupt here:
Backup data: Out Feature (Ch 0-7,Line 0-3)
Recovery data: In Feature (Ch 0-48, Line 0-3)

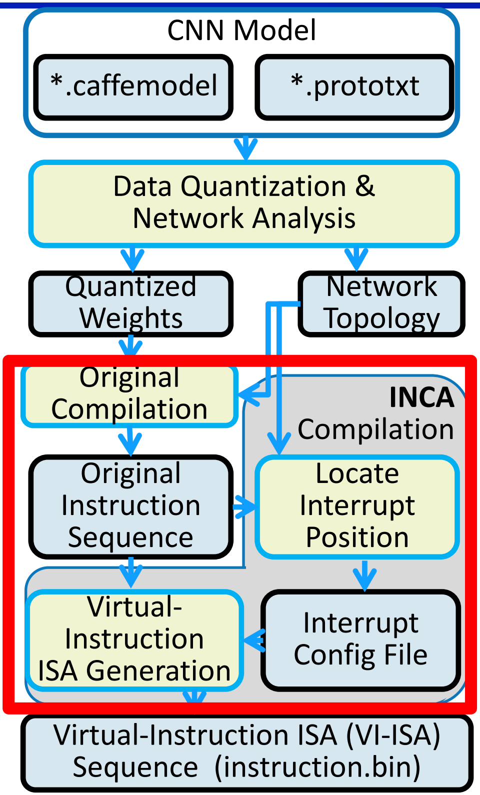
Virtual-Instruction 2 :
If interrupt here:
Backup data: Out Feature (Ch 0-15,Line 0-3)
Recovery data: In Feature (Ch 0-48, Line 0-3)

INCA: Mapping Robot Applications to FPGA



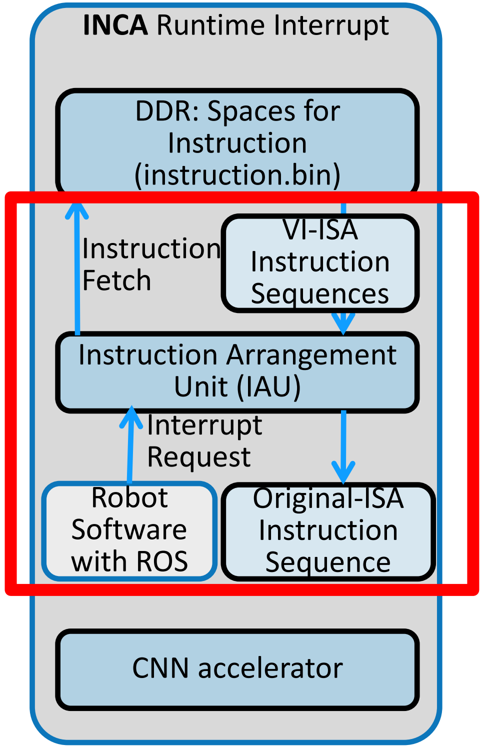
Robot Application with ROS
Task 1: Feature-point Extraction (FE)
Task 2: Place Recognition (PR)
Task ... : VO/ Exploration/ Navigation ...
Task decomposition
CNN Compilation [Fig(c)]
VGG (FE)
ResNet (PR)
...
Virtual-Instruction ISA Sequences
CPU Tasks
C++
Python
Computation Deployment
Interruptible Runtime Accelerator [Fig(d)]
Programmable Logic (PL, FPGA)
ROS Runtime
Processing System (PS, CPU)

(a) INCA framework from robot application to hardware platform.



CNN Model
*.caffemodel
*.prototxt
Data Quantization & Network Analysis
Quantized Weights
Network Topology
Original Compilation
INCA Compilation
Original Instruction Sequence
Locate Interrupt Position
Virtual-Instruction ISA Generation
Interrupt Config File
Virtual-Instruction ISA (VI-ISA) Sequence (instruction.bin)

(b) At compilation, INCA adds virtual instructions to produce VI-ISA instruction sequences.

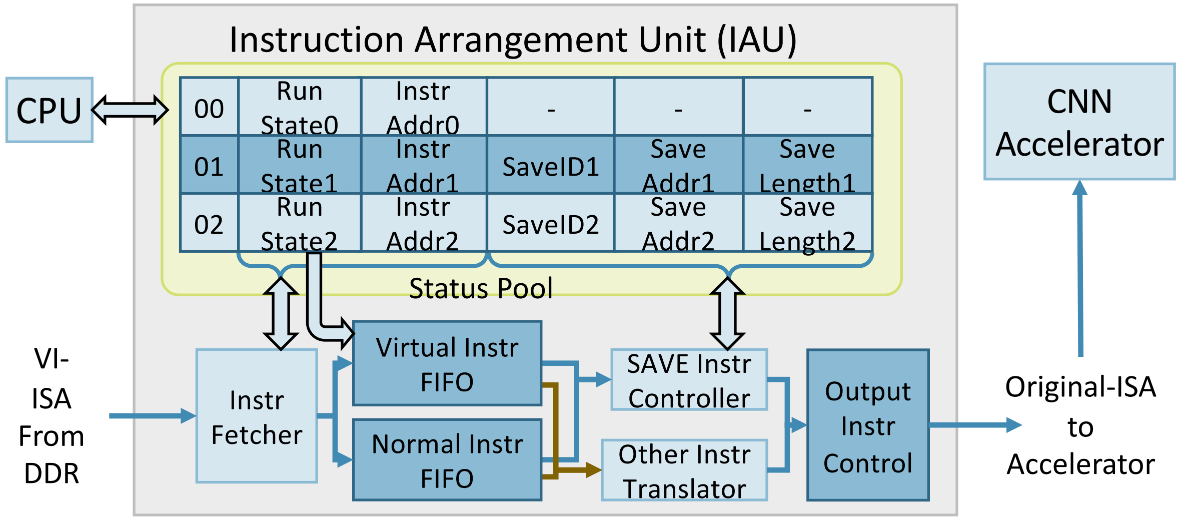


INCA Runtime Interrupt
DDR: Spaces for Instruction (instruction.bin)
Instruction Fetch
VI-ISA Instruction Sequences
Instruction Arrangement Unit (IAU)
Interrupt Request
Robot Software with ROS
Original-ISA Instruction Sequence
CNN accelerator

(c) At runtime, INCA translates the VI-ISA to original ISA executed on the accelerator.

Instruction Arrangement Unit (IAU)

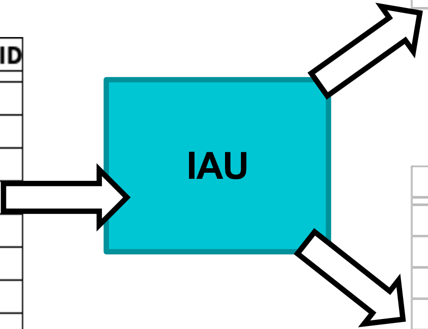
- IAU
  - Monitor interrupt status
  - Fetch instructions for different tasks
  - Translate VI-ISA to original ISA



Example of VI-ISA

	Type	Address1	Address2	Address3	Workload	Virtual	SaveID
1	LOAD_D	0x0000	0x4000	-	0x20	2'b00	1
2	LOAD_W	0x1000	0x5000	-	0x1	2'b00	1
3	CALC_F	0x4000	0x6000	0x5000	0x20	2'b00	1
4	SAVE	0x2000	0x6000	-	0x20	2'b01	1
5	LOAD_D	0x0000	0x4000	-	0x20	2'b10	1
6	LOAD_W	0x1001	0x5000	-	0x1	2'b00	1
7	CALC_F	0x4000	0x6020	0x5000	0x20	2'b00	1
8	SAVE	0x2000	0x6000	-	0x40	2'b00	1

(a) Input instruction sequence



	Type	Address1	Address2	Address3	Workload
1	LOAD_D	0x0000	0x4000	-	0x20
2	LOAD_W	0x1000	0x5000	-	0x1
3	CALC_F	0x4000	0x6000	0x5000	0x20
4	LOAD_W	0x1001	0x5000	-	0x1
5	CALC_F	0x4000	0x6020	0x5000	0x20
6	SAVE	0x2000	0x6000	-	0x40

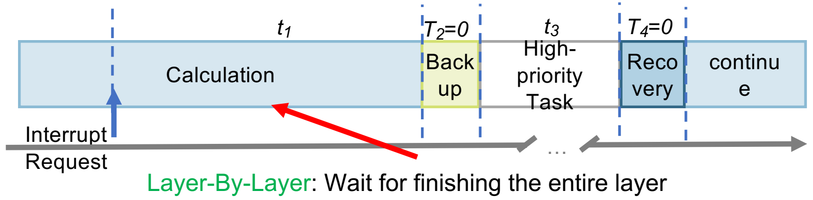
(b) Executed sequence when no interrupt.
Virtual Instructions are deleted.

	Type	Address1	Address2	Address3	Workload
1	LOAD_W	0x1000	0x5000	-	0x1
2	LOAD_D	0x0000	0x4000	-	0x20
3	CALC_F	0x4000	0x6000	0x5000	0x20
4	SAVE	0x2000	0x6000	-	0x20
	HIGH-PRIORITY CNN				
5	LOAD_D	0x0000	0x4000	-	0x20
6	LOAD_W	0x1001	0x5000	-	0x1
7	CALC_F	0x4000	0x6020	0x5000	0x20
8	SAVE	0x2020	0x6020	-	0x20

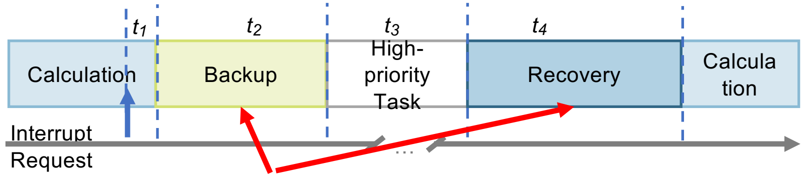
(c) Executed sequence when interrupt occurs.
Virtual Instructions (**Blue**) are executed.
Normal SAVE (**Red**) is modified.

Results for Interrupt (T_{cost} and $T_{latency}$)

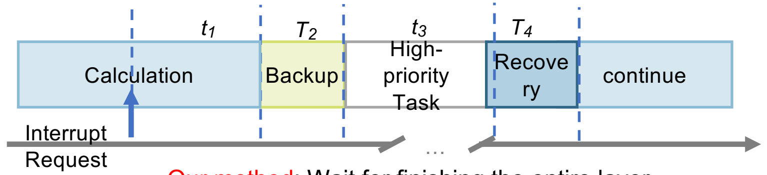
- Low-priority interruptible CNN backbone: Example on ResNet-101



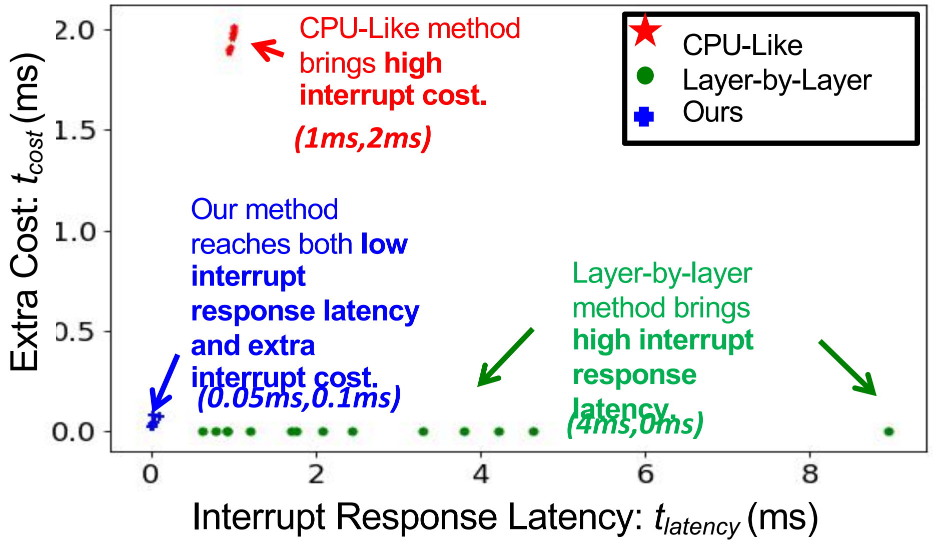
Layer-By-Layer: Wait for finishing the entire layer



CPU-Like (Interrupt Immediately):Backup/Recovery All on-chip caches

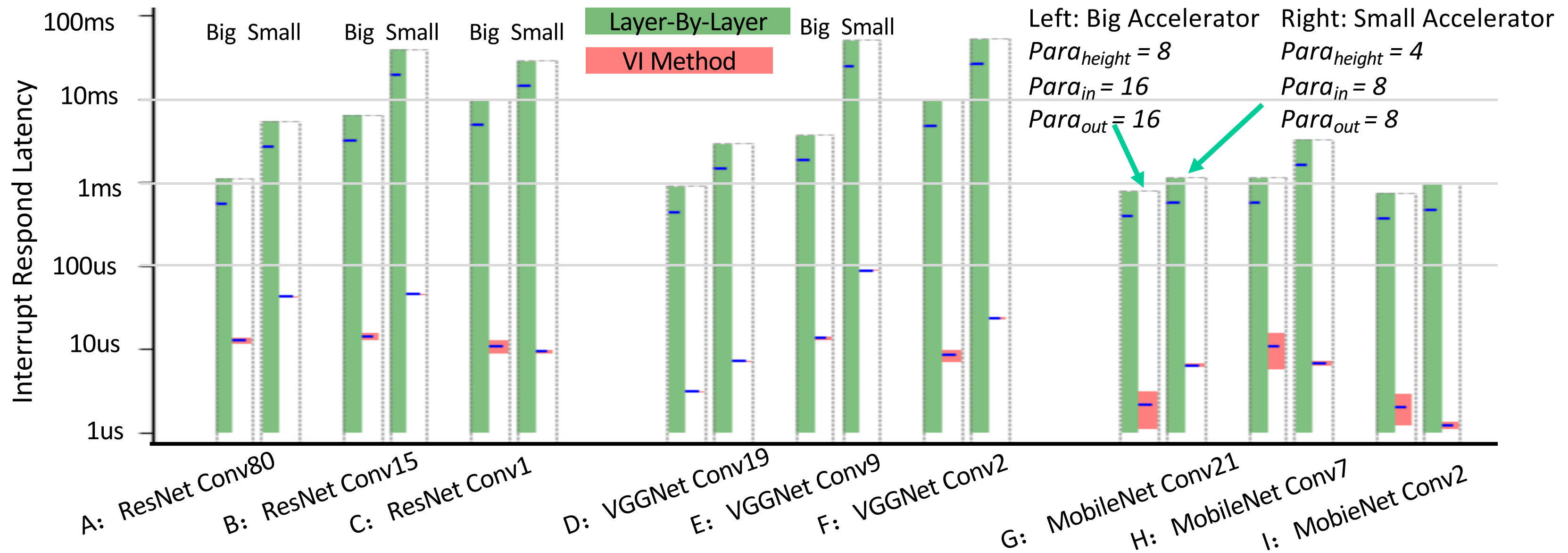


Our method: Wait for finishing the entire layer



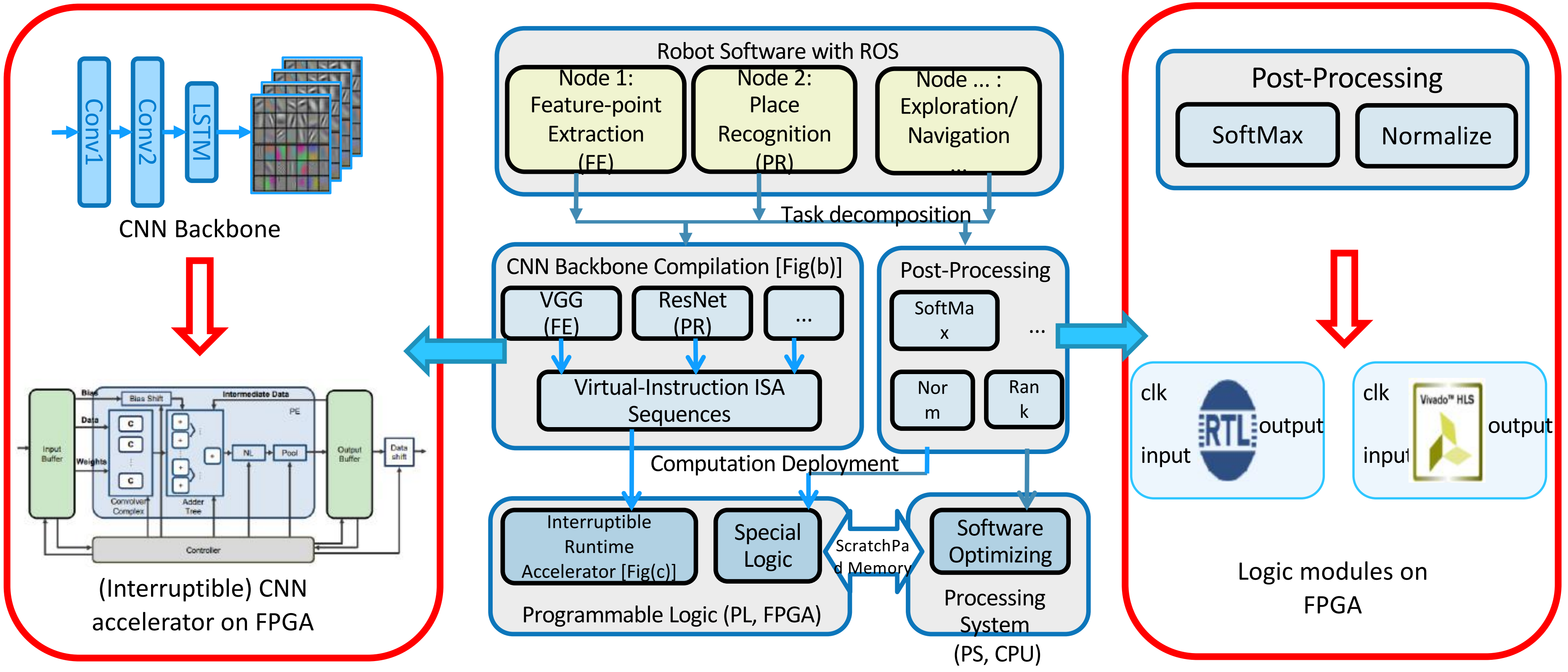
Latency Comparison ($T_{latency}$)

- Test on different CNN networks: ~100us latency



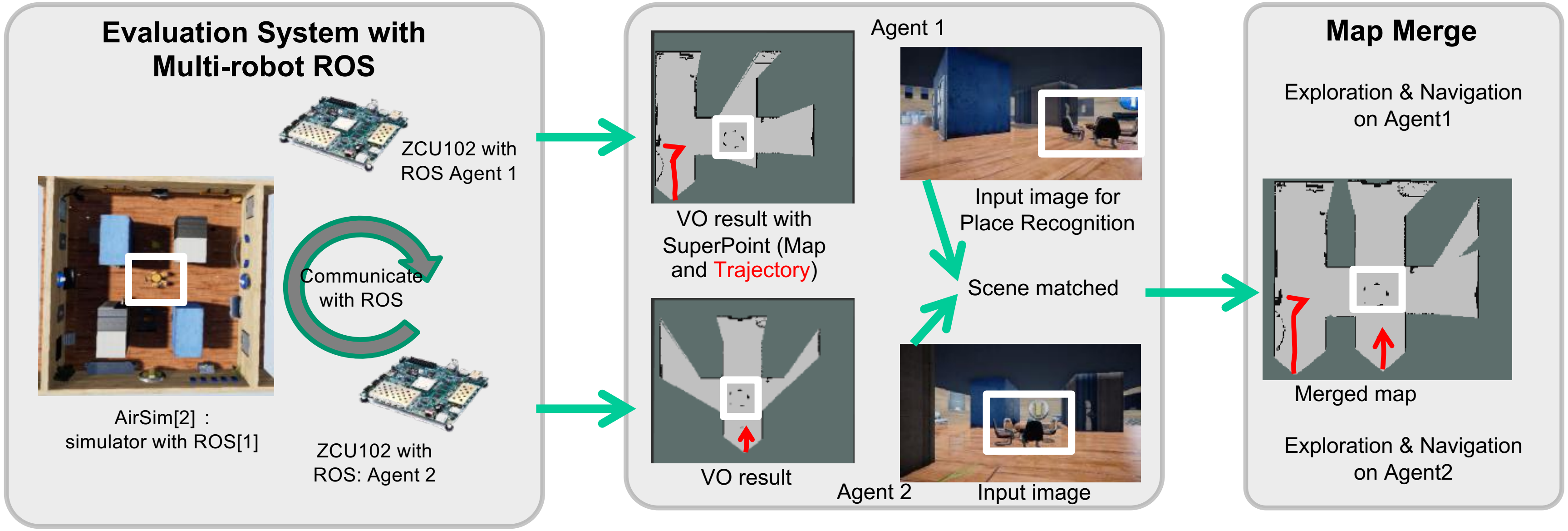
Layer	A	B	C	D	E	F	G	H	I
Network	ResNet101			VGGNet			MobileNet		
W x H	41 x 31	160 x 120	640 x 480	7 x 7	56 x 56	224 x 224	14 x 14	56 x 56	112 x 112
Chin x Chout	1024 x 256	128 x 128	3 x 64	512 x 512	256 x 256	64 x 128	512 x 512	256 x 256	32 x 32
kernel	1	3	7	3	3	3	1	1	3

Map Robot Applications onto FPGA



Multi-robot Exploration/SLAM with Hardware-In-Loop Simulation

- Multi-robot Exploration with ROS and Interruptible DPU on Xilinx Board
  - Feature-point extraction 20fps, place recognition (loop-closure detection) 2 fps.

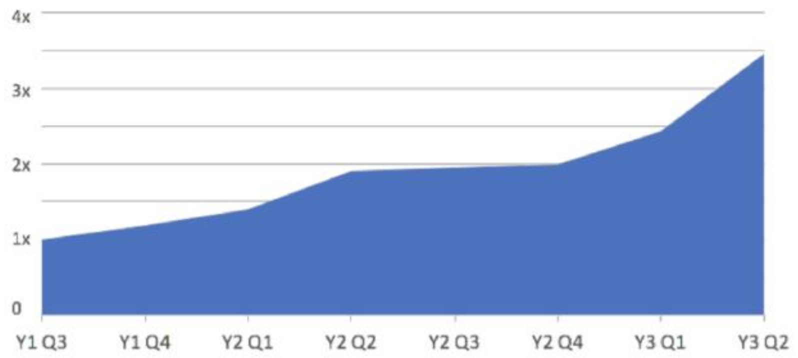

Evaluation System with Multi-robot ROS
ZCU102 with ROS Agent 1
Communicate with ROS
AirSim[2] : simulator with ROS[1]
ZCU102 with ROS: Agent 2
Agent 1
VO result with SuperPoint (Map and Trajectory)
Input image for Place Recognition
Scene matched
VO result
Agent 2
Input image
Map Merge
Exploration & Navigation on Agent1
Merged map
Exploration & Navigation on Agent2

[1] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In ICRA workshop, Vol. 3. Kobe, Japan, 5.
[2] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2018. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and service robotics. Springer, 621–635.

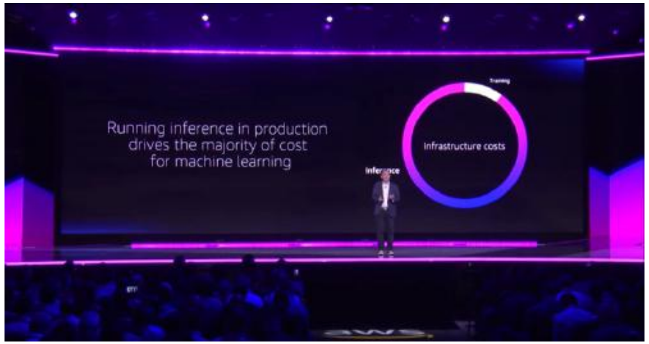
Case Study 2:
Can we use the large ‘TOPs’ to the extreme?
-- Sharing on the Cloud

DNN Inference Tasks in the Cloud

- Deep neural network (DNN) inference tasks take up the majority of the total deep learning workloads in data centers.



DL inference tasks is doubling each year in Facebook data center [1]



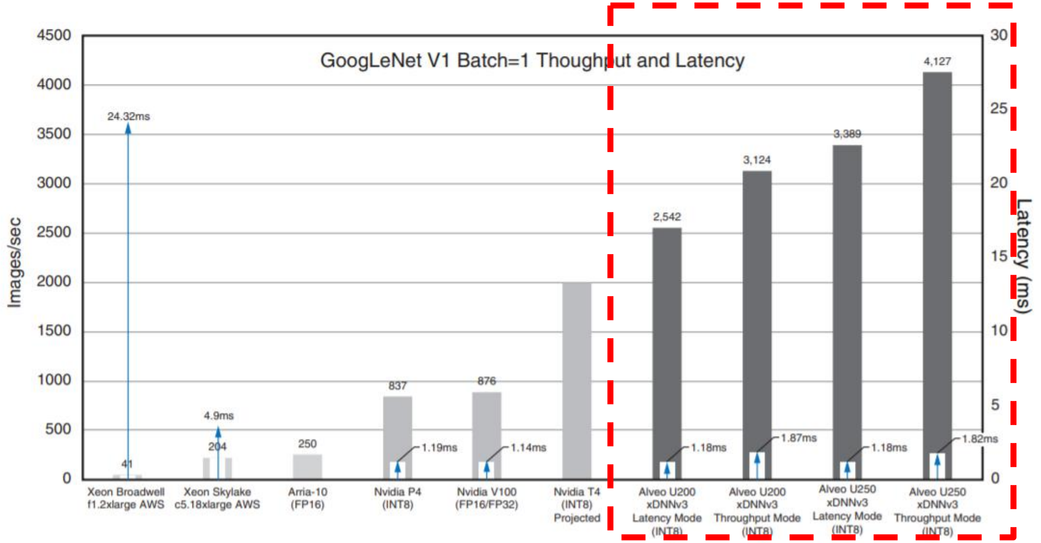
Running inference in production drives the majority of cost for machine learning
Training
Infrastructure costs
Inference

Inference tasks make up nearly 90% of the total deep learning tasks in Amazon data center [2]

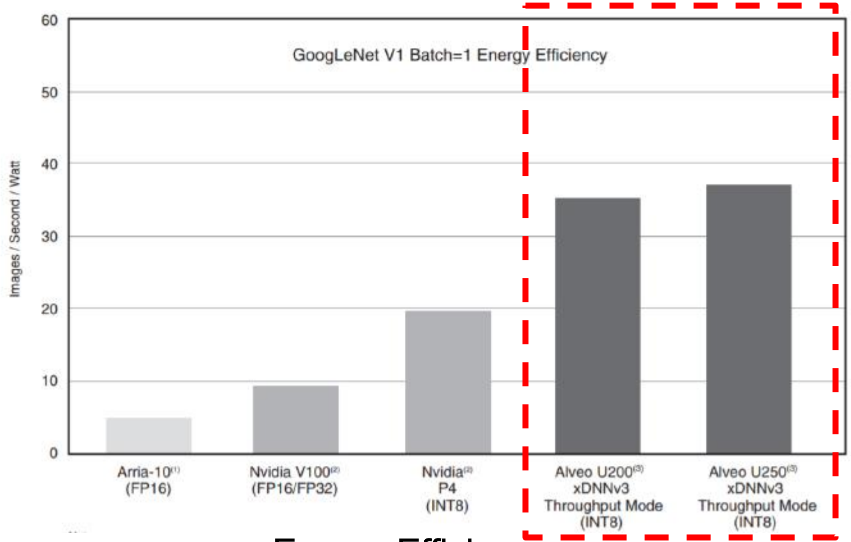
[1] Jongsoo Park et al. 2018. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. arXiv.
[2] Andy Jassy. 2018. Amazon AWS ReInvent Keynote. https://www.youtube.com/watch?v=ZOIkOnW640A.

FPGAs for DNN Inference

- FPGAs are promising acceleration platforms for DNN inference in the cloud[1].



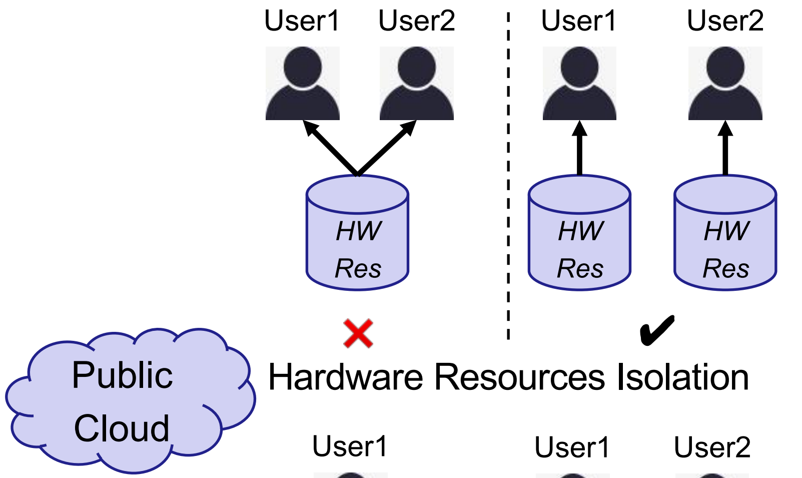
Throughput and Latency



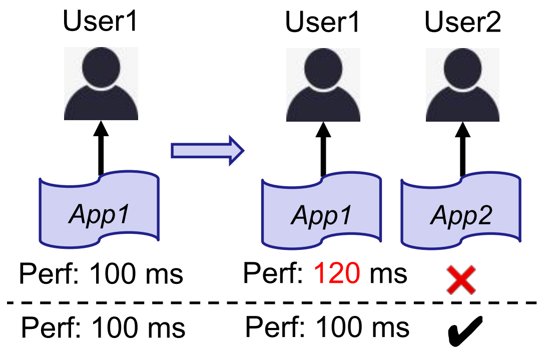
Energy Efficiency

[1] Xilinx. 2018. White Paper: Accelerating DNNs with Xilinx Alveo Accelerator Cards

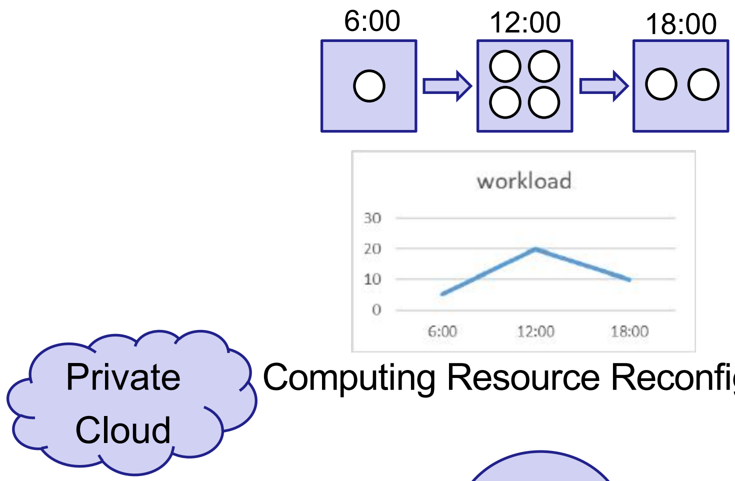
Key Ideas of Virtualization



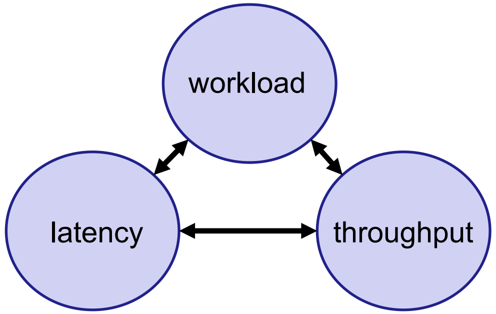
Hardware Resources Isolation



Computing Performance Isolation

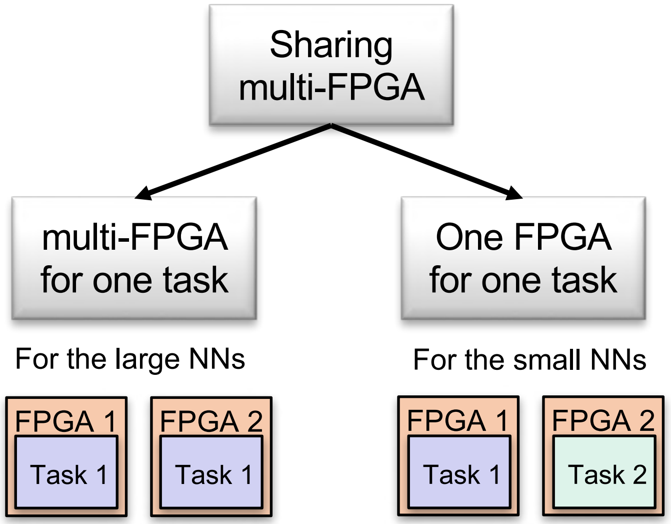


Computing Resource Reconfiguration

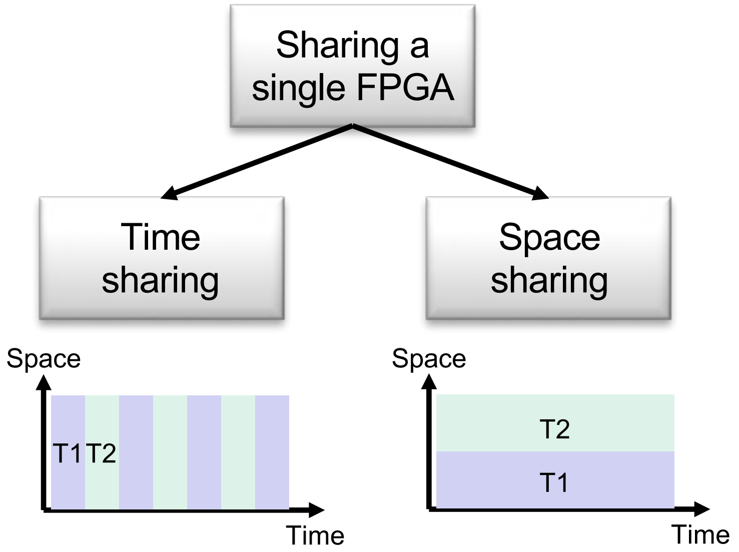


Maximize System Performance

FPGA Sharing in the Cloud

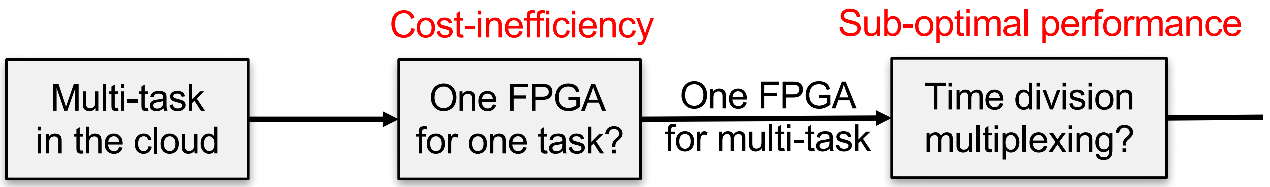


- Good physical and performance isolation
- High reconfiguration overhead
- Cost-inefficient



- Time sharing:
  - Hard to ensure physical isolation
  - Sub-optimal performance
- Space sharing:
  - Easy to ensure isolation
  - High reconfiguration overhead

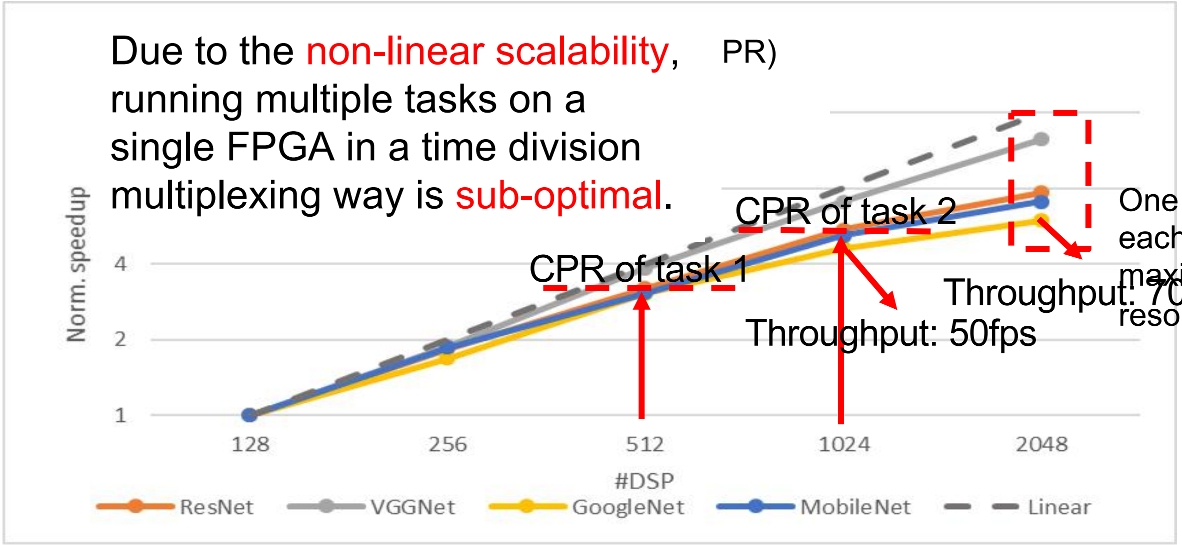
How to Support Multiple Tasks in the Cloud?



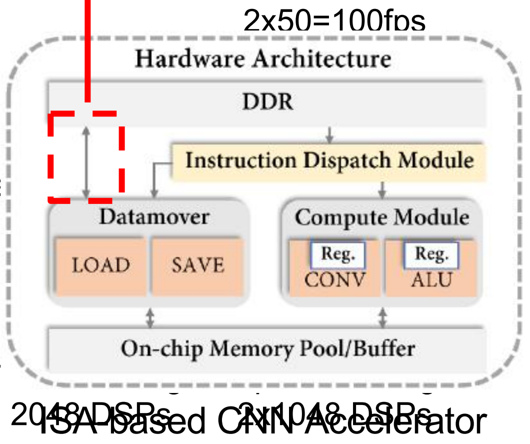
Design goal

The non-linear scalability is due to the limited DDR memory bandwidth on FPGA (2048 DSPs, 12.8 GB/s)

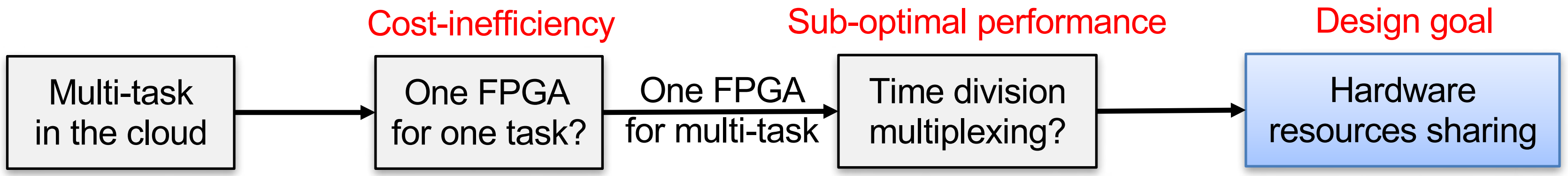
Due to the non-linear scalability, running multiple tasks on a single FPGA in a time division multiplexing way is sub-optimal.

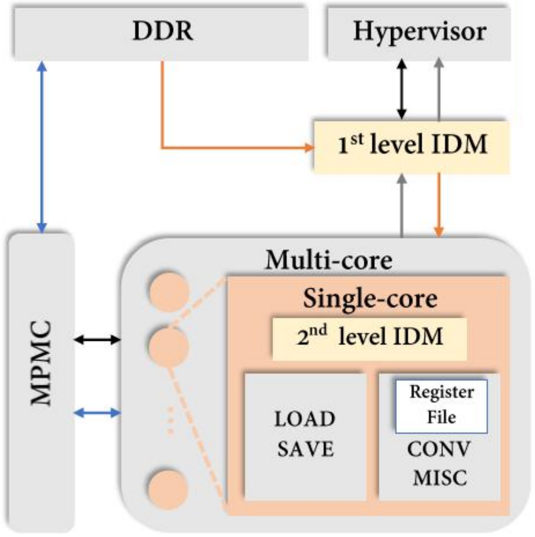


Normalized performance with different hardware resources in the single-task and static-workload scenario.



How to Support Multiple Tasks in the Cloud?

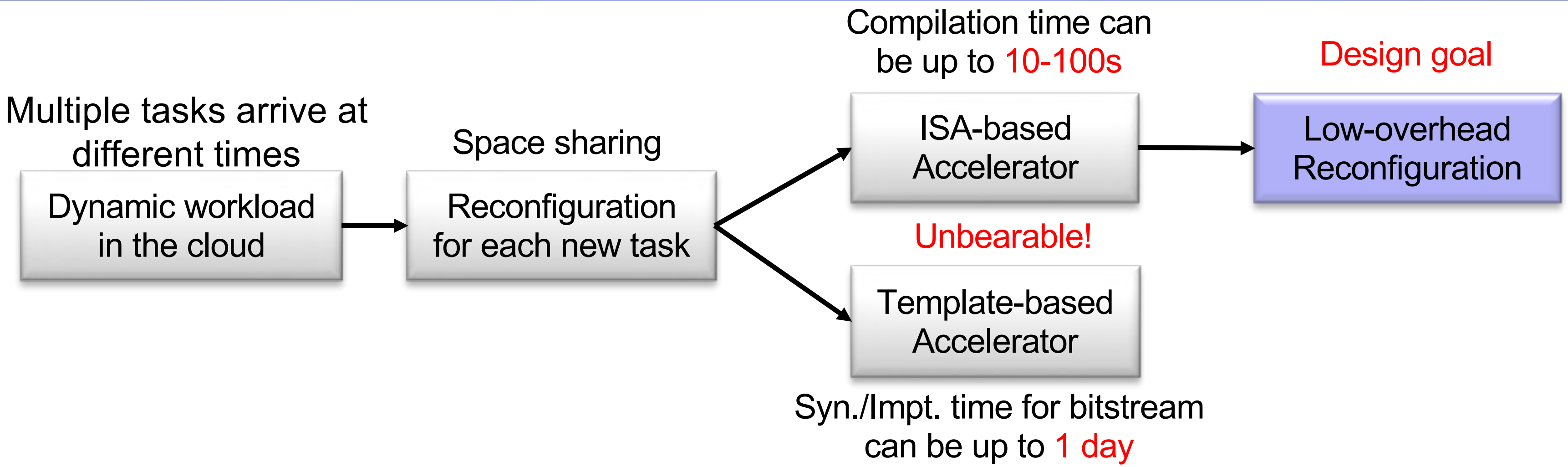


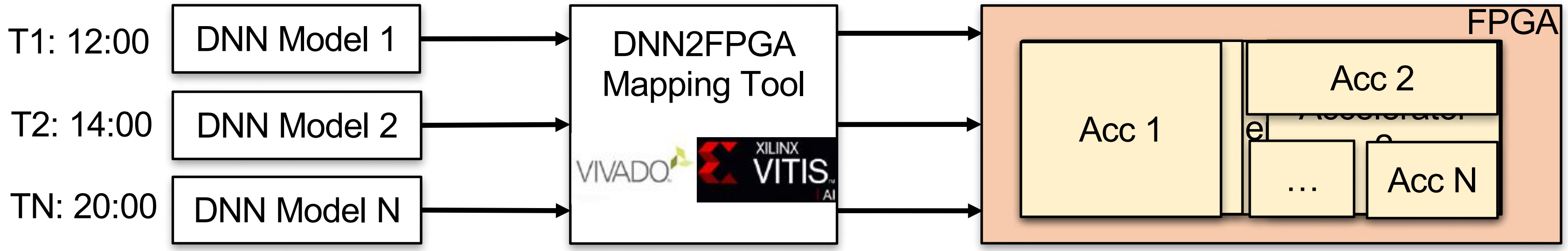


Our solution: Multi-core CNN Accelerator

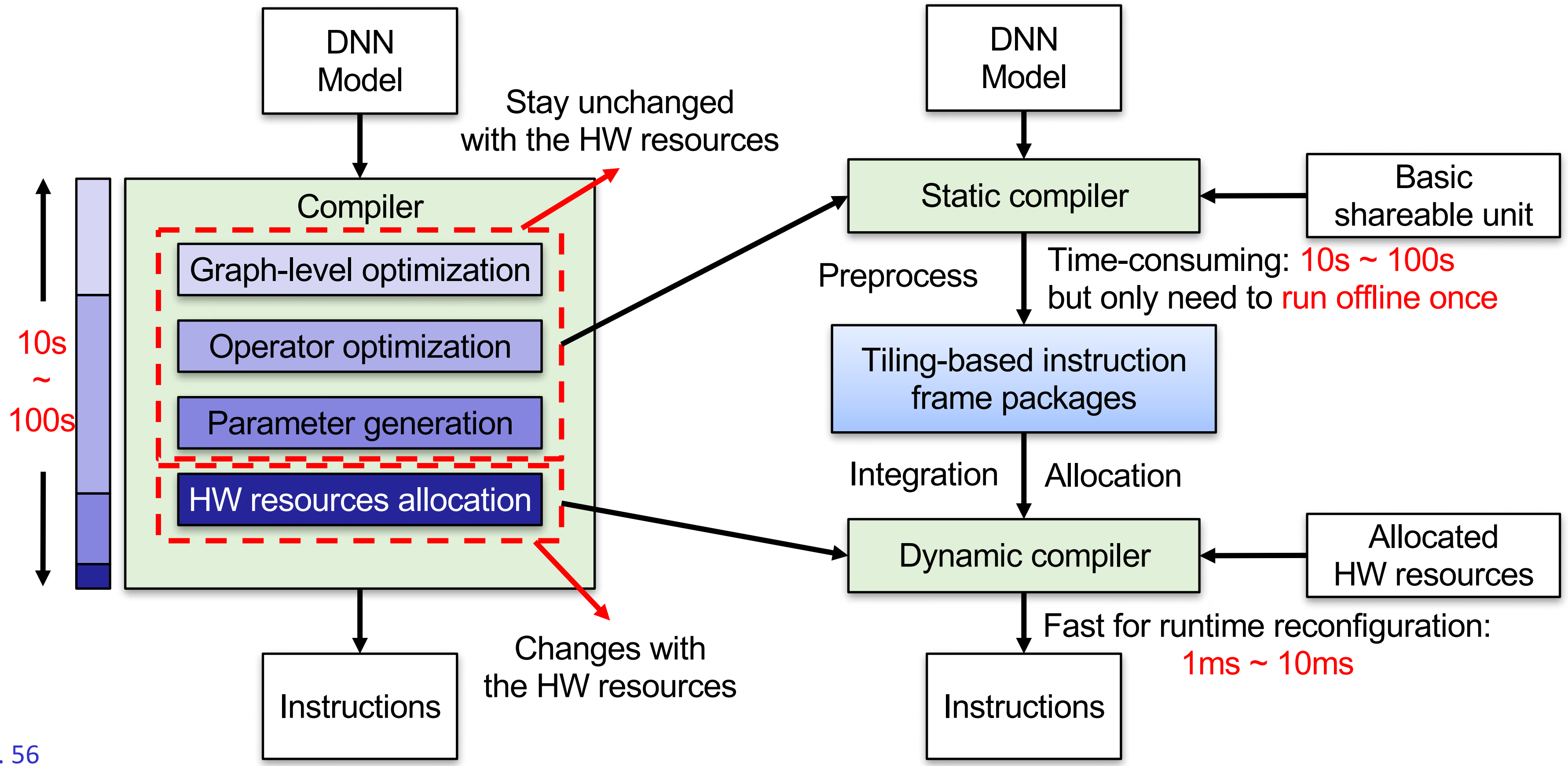
- Good isolation: each core is an unique hardware resource
- Dynamic reconfiguration: allocate the computing cores based on the CNN workload
- Optimized for both latency and throughput
- Key problem: high reconfiguration overhead?

How to Support Dynamic Workload in the Cloud?


Multiple tasks arrive at different times
Dynamic workload in the cloud
Space sharing
Reconfiguration for each new task
Compilation time can be up to 10-100s
ISA-based Accelerator
Unbearable!
Template-based Accelerator
Syn./Impt. time for bitstream can be up to 1 day
Design goal
Low-overhead Reconfiguration

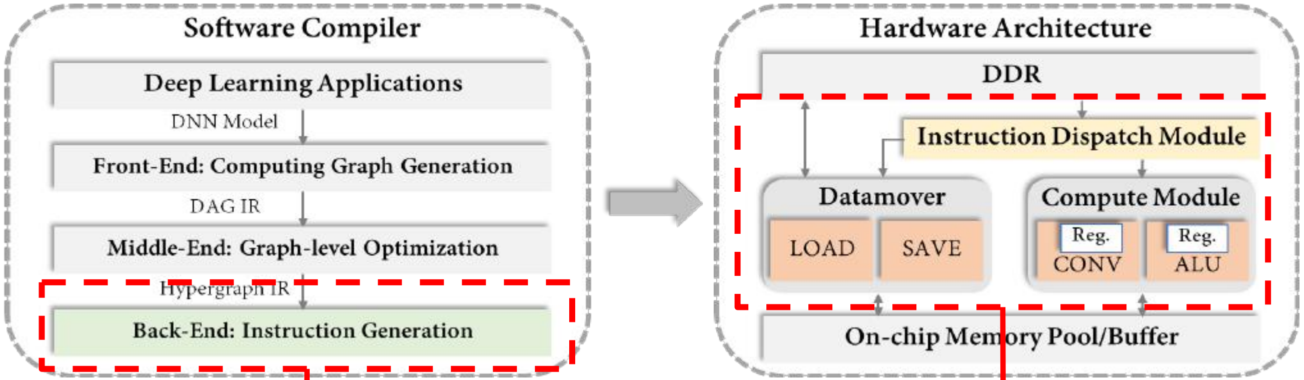


Low-overhead Reconfiguration of ISA-based Accelerator



Design Techniques

Baseline design of ISA-based CNN accelerator


Software Compiler
Deep Learning Applications
DNN Model
Front-End: Computing Graph Generation
DAG IR
Middle-End: Graph-level Optimization
Hypergraph IR
Back-End: Instruction Generation
Hardware Architecture
DDR
Instruction Dispatch Module
Datamover
LOAD
SAVE
Compute Module
Reg.
CONV
Reg.
ALU
On-chip Memory Pool/Buffer

Design goal for software:
Low-overhead reconfiguration

Design goal for software:
Multi-Task concurrent execution

Two-stage static-dynamic compilation ↔ Hardware resources pool

Design Techniques for Virtualization

Instruction Support:
A bridge between the static and dynamic compiler.

Instruction Support:
Original IDM: Module-level scheduler
New IDM: Task-level scheduler

Tiling-based instruction frame package ↔ Two-level Instruction Dispatch Module

Experiments

- Experiment Setup
  - Hardware platform: Xilinx VU9P FPGA (Vivado 2018.2) + Intel Xeon E5-2643 CPU
  - Deep learning model (Caffe): VGG-16, ResNet-18, MobileNet-v1
  - **Simulation of applications in the cloud: Poisson distribution [1]**
  - CNN accelerator: Parallelism is 8192 ops/cycle (2048 DSPs, each for 2xINT8 MAC), running at 200MHz.
  - Static implementation: 1) single large core (1 x 8192). 2) eight small cores (8 x 1024)
  - Virtualized implementation: Multi-core virtualized design (16 x 512).
- Hardware resources utilization
  - The virtualized multi-core design utilize 3.25% more BRAMs, 3.08% more LUTs than the static multi-core design.
  - Noted that the resources of URAM are not utilized in this experiment.

Implementation	LUT	FF	BRAM	DSP
Static single core	22.01%	11.06%	35.23%	29.94%
Static multi-core	54.51%	13.3%	86.34%	29.94%
Virtualized multi-core	57.59%	30.59%	89.59%	29.94%

[1] Hamzeh Khazaei et al. 2012. Performance of cloud centers with high degree of virtualization under batch task arrivals. IEEE Transactions on Parallel and Distributed Systems.

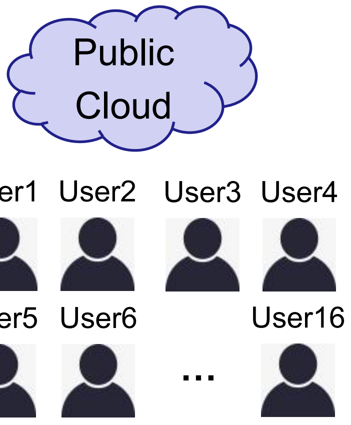
Experiments

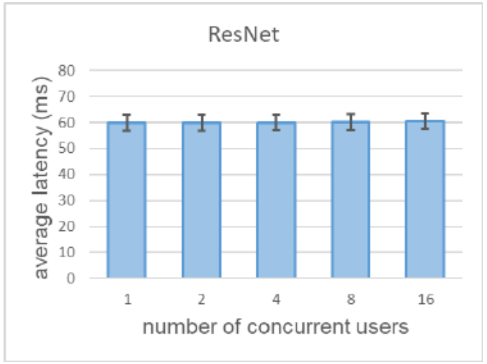
Evaluation on the ***isolation characteristics*** of the **Public Cloud**:

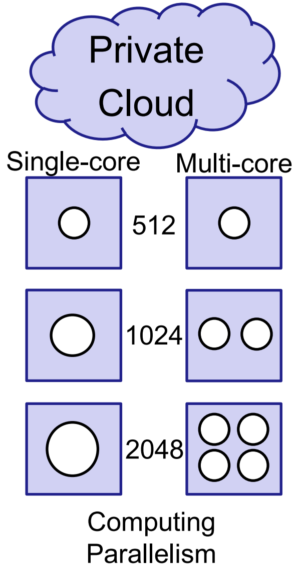
- Effective isolation: the average performance difference is <1 ms.

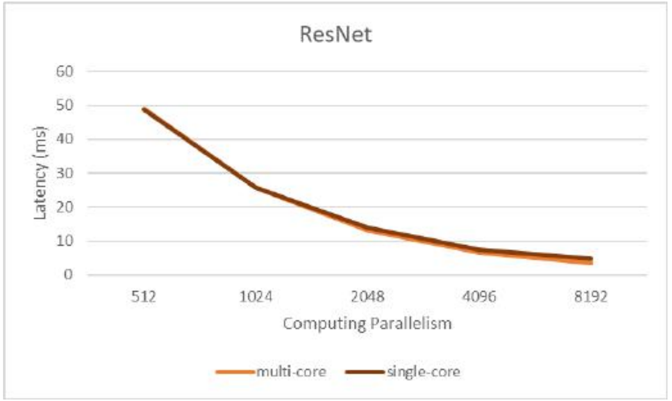
Evaluation on the ***single-task performance (latency)*** of the **Private Cloud**:

- The average performance loss of multi-core is less than 3%.









Experiments

Evaluation on the reconfiguration overhead:

- Complete re-generation of instructions can be 12-200s, which is unbearable.
- Time sharing method has no context-switch cost, but it is cost-efficient and sub-optimal.
- Our space sharing method introduces ~1ms context-switch overhead, but achieves better performance on throughput and negligible performance loss on latency.

10.6-35.2% extra overhead of static compilation, which only needs to run offline once.

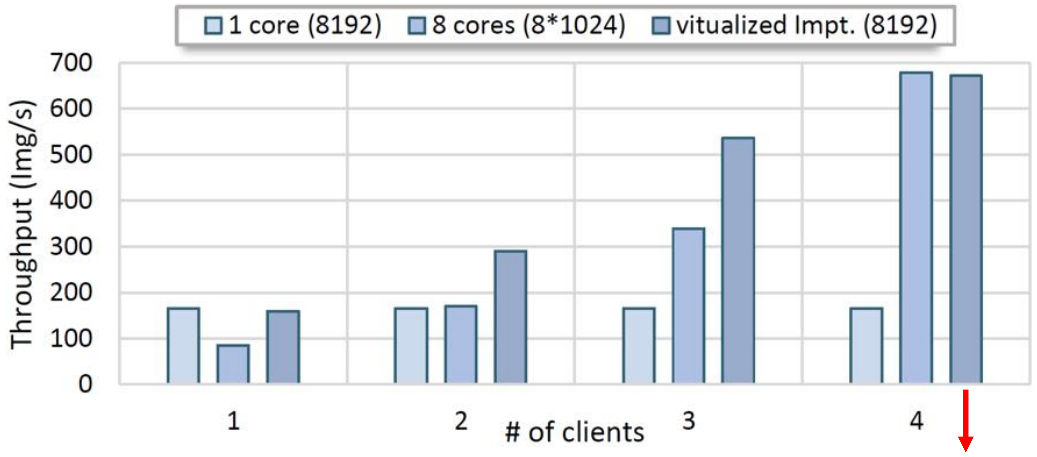
CNN Model	Original Compilation	Static Compilation	Dynamic Compilation	Transfer Time	Context-switch Cost	Computation Time	
						Min(8192)	Max(512)
VGG-16	201930	223457	1.75	0.26	2.04	21.2	219.1
ResNet-18	25336	34257	1.55	0.09	1.68	8.4	37.7
MobileNet-v1	12074	15767	0.81	0.09	0.94	6.0	15.2

~1ms context-switch cost is acceptable for the computation time of 6.0-219.2ms during runtime.

Experiments

Evaluation on the performance of *throughput* on the *multi-task* scenario:

- Static single core(8192): low throughput for multiple tasks due to only one core.
- Static multi-core(8x1024): the worst throughput when there is only one task.
- Virtualized multi-core (16x512): the best throughput for different workloads.



1.88x ~ 2.20x
higher throughput

Future Work

- Implementation on the cloud FPGA (Alibaba Cloud)
- Further improvement on the dynamic compilation cost
  - Down to 0.1ms, 1% of the inference time
- Further improvement on the hardware architecture.

Case Study 3: Reliable NN on FPGA?

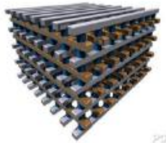
When NN Chips Meet Fault Tolerant Problem

Environment Effect

- Outer Space Radiation
- Temperature Variation
- Ultra Low Power Supply

Inherent Property

- Device Aging



- Immature New Device Tech.

Artificial

- Hacker Attack

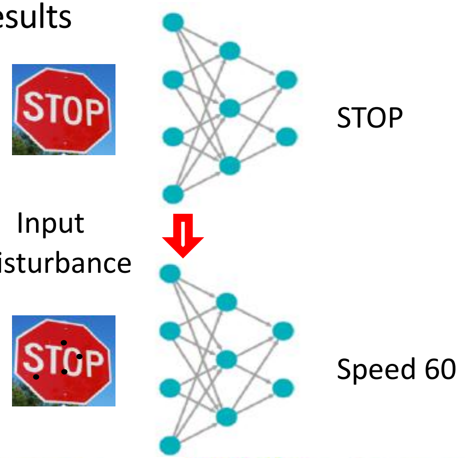
Environment Effect, Inherent Property ⇩ **Reliability Problem**

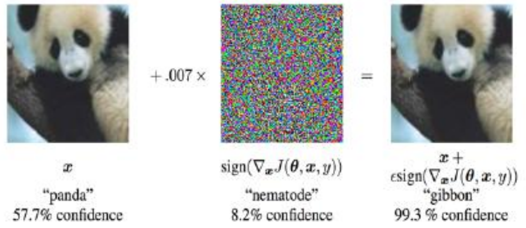
Artificial ⇩ **Security Problem**

Reliability problems of NN system

Algorithm:

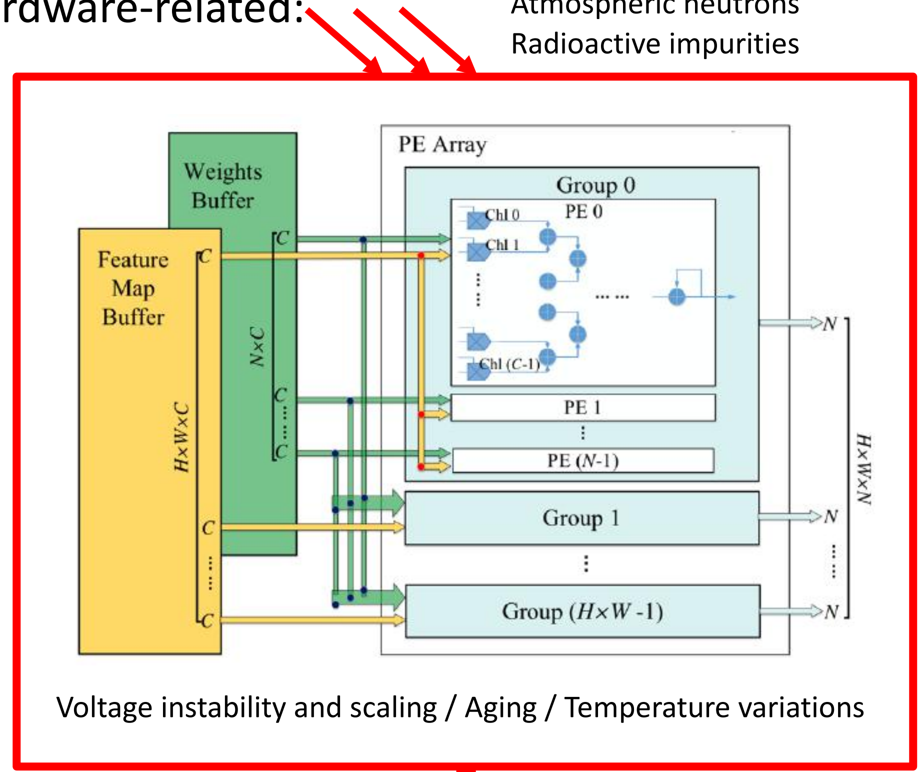
Adversarial Robustness, Neural networks are susceptible to subtle input disturbances and output incorrect results





Hardware-related:

Atmospheric neutrons
Radioactive impurities



Voltage instability and scaling / Aging / Temperature variations

Computational Faults

Mitigations for computational faults

Algorithm

Many algorithms (e.g. NN, of course) have intrinsic resilience.
Advantage: further improve algorithmic resilience -> more cost-effective solutions
Disadvantage: cannot tolerate faults in control flow

Software/ Mapping

Static:
Mapping/routing in synthesis
Dynamic:
Redundant execution
Run-time error detection + recovering or remapping

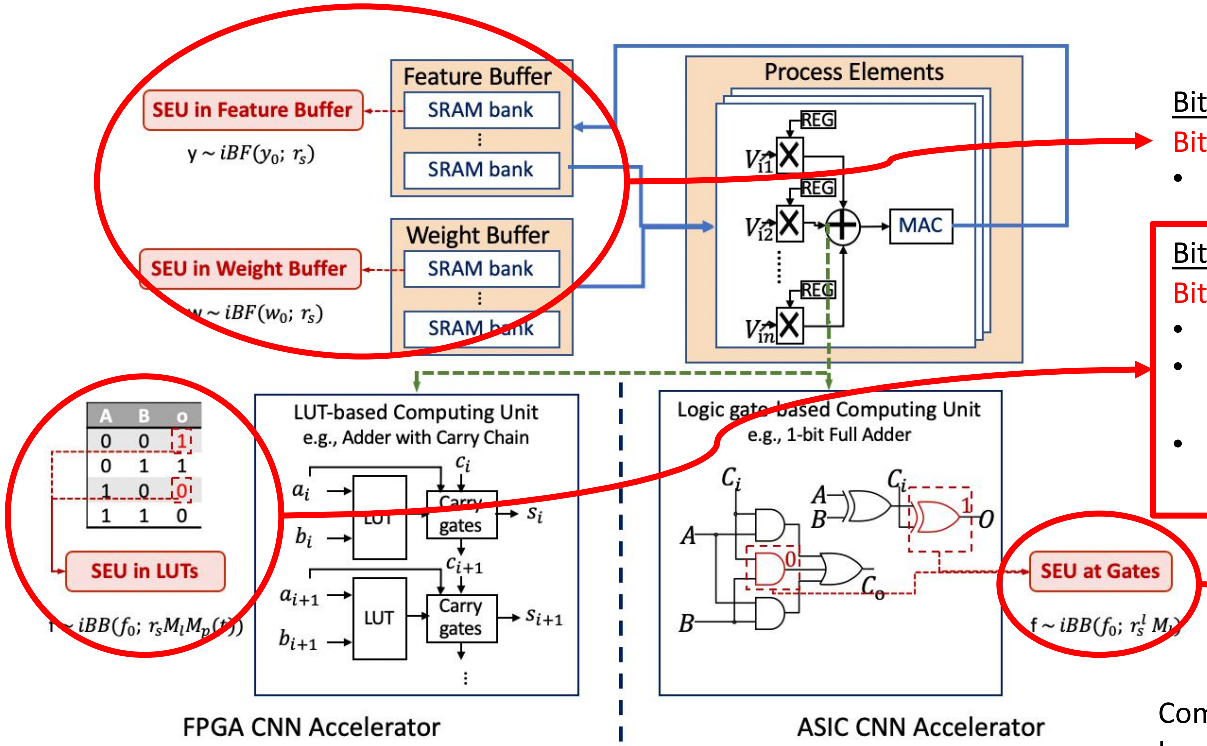
Hardware

Redundancy: e.g. TMR
Parity: e.g. ECC
Hardened circuit: e.g. DICE

Advantage: effective
Disadvantage: costly

Most studies on fault-tolerance are exploiting the NN's algorithmic resilience.
Few studies are improving the NN's algorithmic resilience.

Analysis of Platforms: FPGA and ASIC



Memory elements

Bit-flips in SRAM buffers:

Bit-flips on weights/features. E.g. 0010 -> 0110

- Transient

Bit-flips in LUT SRAM:

Bit-Biases added (LUT-based adders)

- Persistent
- Multiple adders are involved -> larger error rate per feature location
- PE reusing -> one fault would result in errors in multiple feature locations

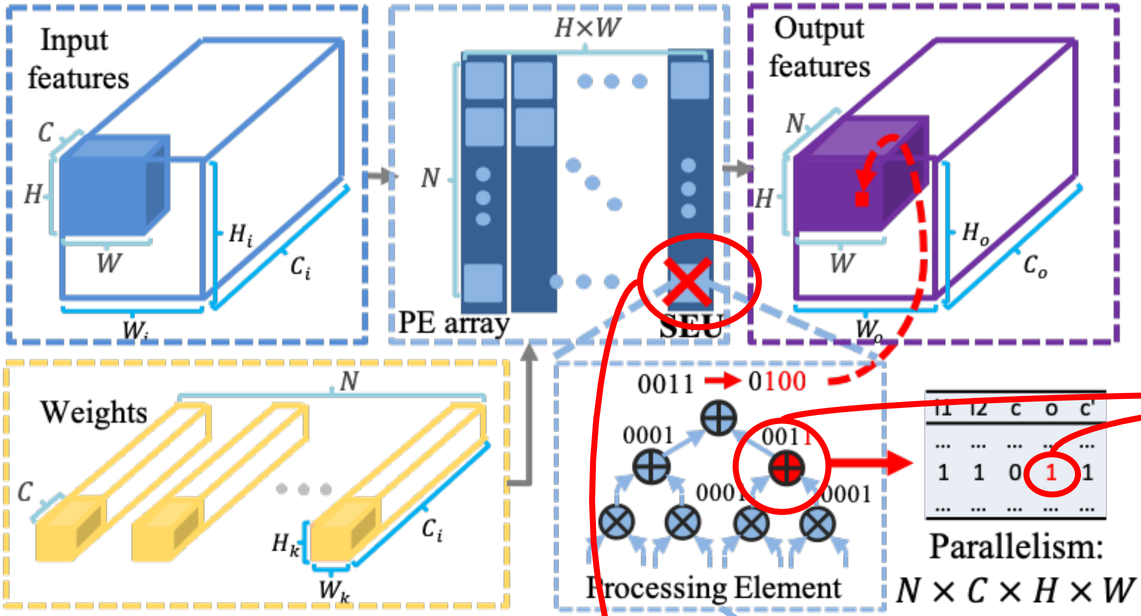
Logic elements

Combinational and sequential logic elements have lower soft error rates

[1] Wenshuo Li, Xuefei Ning, Guangjun Ge, Xiaoming Chen, Yu Wang, Huazhong Yang. FTT-NAS: Discovering Fault-Tolerant Neural Architecture. In 25th Asia and South Pacific Design Automation Conference (ASP-DAC 2020).

(LUTs) Feature Fault Model: MAC-i.i.d random Bit-Bias

MiBB feature fault model



Random Bit-Bias Faults: Error added as biases

$$y = g(W \circledast x + b + \theta \cdot 2^{\alpha - l} \cdot (-1)^{\beta})$$

$$\text{s.t.} \quad \theta \sim \text{Bernoulli}(p)^{C_o \times H \times W}$$

$$\alpha \sim U\{0, \cdots, Q-1\}^{C_o \times H \times W}$$

$$\beta \sim U\{0,1\}^{C_o \times H \times W}$$

$$p = p_m \times ck^2$$

p_m is the prob that an adder encounter error

α refers to the fault bit

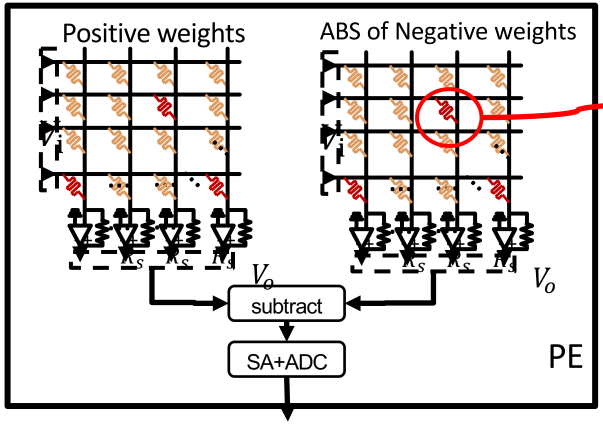
β refers to the sign of biases

$ck^2 - 1$ additions spatially expanded onto adders

θ_i refers to whether a fault occurred in one feature value

（RRAM）Weight Fault Model: arbitrary-distributed SAF

RRAM with multi-bit cells



Q-bit adSAF weight fault model

Weight representation range: $[-R^w, R^w] = [-2^{-l}(2^{Q+1}-1), 2^{-l}(2^{Q+1}-1)]$

stuck at 0 with p_0
stuck at Maximum R^w with p_1

Computation of one layer:

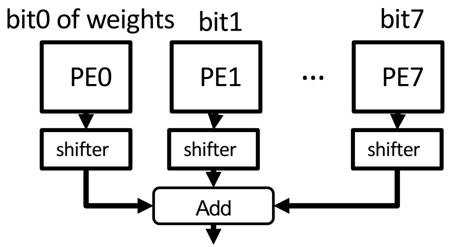
$$y = g(W' \circledast x + b)$$

$$\text{s.t.} \quad W' = (1-\theta)\cdot W + \theta \cdot e$$

θ refers to the weight fault position mask
m refers to the fault type mask, SAF0/SAF1
e refers to the fault target value, $0/\pm R^w$

$$\begin{cases} \theta \sim \text{Bernoulli}(p_0+p_1)^{C_o \times c \times k \times k} \\ m \sim \text{Bernoulli}(\frac{p_1}{p_0+p_1})^{C_o \times c \times k \times k} \\ e = R^w \,\text{sgn}(W)\cdot m \end{cases}$$

RRAM with single-bit cells (e.g. 8-bit weights)



1-bit adSAF weight fault model

memristor stuck at 1/0: a bit of a weight stuck at 0/1

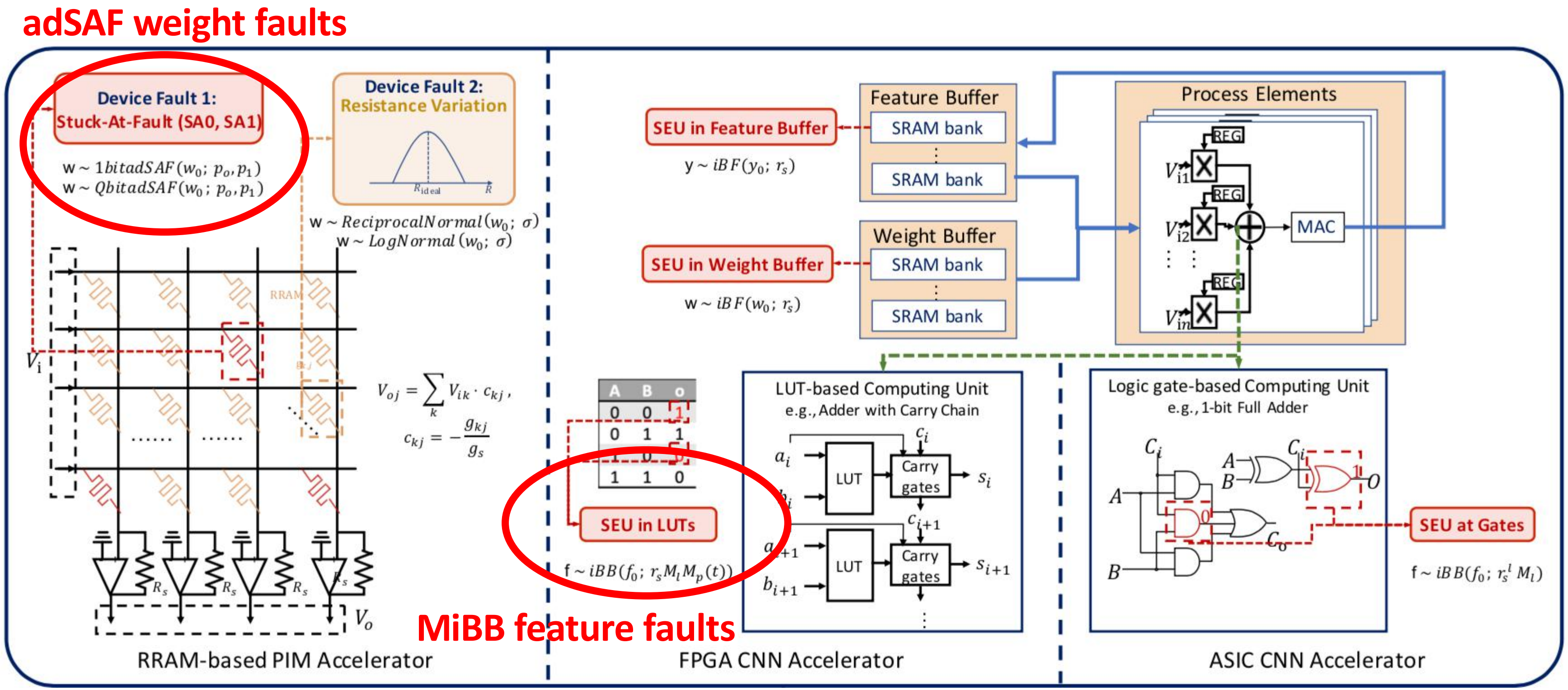
Change of one weight:

$$w' = \text{sgn}(w)2^{-l}(((\neg\theta) \wedge 2^l|w|) \vee (\theta \wedge e))$$

θ refers to the bit fault position mask
e refers to the target value at each bits, 0/1

$$\begin{cases} \theta = \sum_{q=1}^{Q} \theta_q 2^{q-1} \\ \theta_q \overset{iid}{\sim} \text{Bernoulli}(p_0+p_1), \quad q = 1,\cdots,Q \\ e = \sum_{q=1}^{Q} m_q 2^{q-1} \\ m_q \overset{iid}{\sim} \text{Bernoulli}(\frac{p_1}{p_0+p_1}), \quad q = 1,\cdots,Q \end{cases}$$

Summary of Computational Faults



Influence of Neural Architectures on Reliability

Preliminary experiments show that different archs have different resilience capabilities.

Feature faults: Random Bit-Bias
Comparison with different p_m $(0, 10^{-5}, 10^{-4})$

Model	Acc($0/10^{-5}/10^{-4}$)	#Params	#FLOPS
ResNet-20	94.7/63.4/10.0	11.2M	1110M
VGG-16†	93.1/21.4/10.0	14.7M	626M
MobileNet-V2	92.3/10.0/10.0	2.3M	182M

†: For simplicity, we only keep one fully-connected layer of VGG-16.

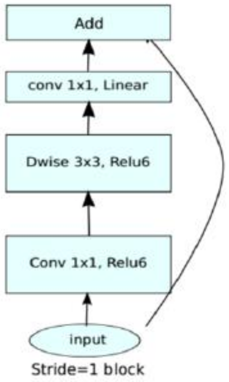
Weight faults: 8-bit SAF
Comparison with different SAF ratios (0, 4%, 8%)

Model	Acc(0/4%/8%)	#Params	#FLOPS
ResNet-20	94.7/64.8/17.8	11.2M	1110M
VGG-16	93.1/45.7/14.3	14.7M	626M
MobileNet-V2	92.3/26.2/11.7	2.3M	182M

Operation types and connections are possible to affect the reliability of models. For example:

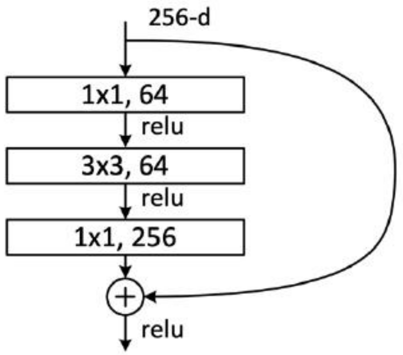
MobileNet Block

×

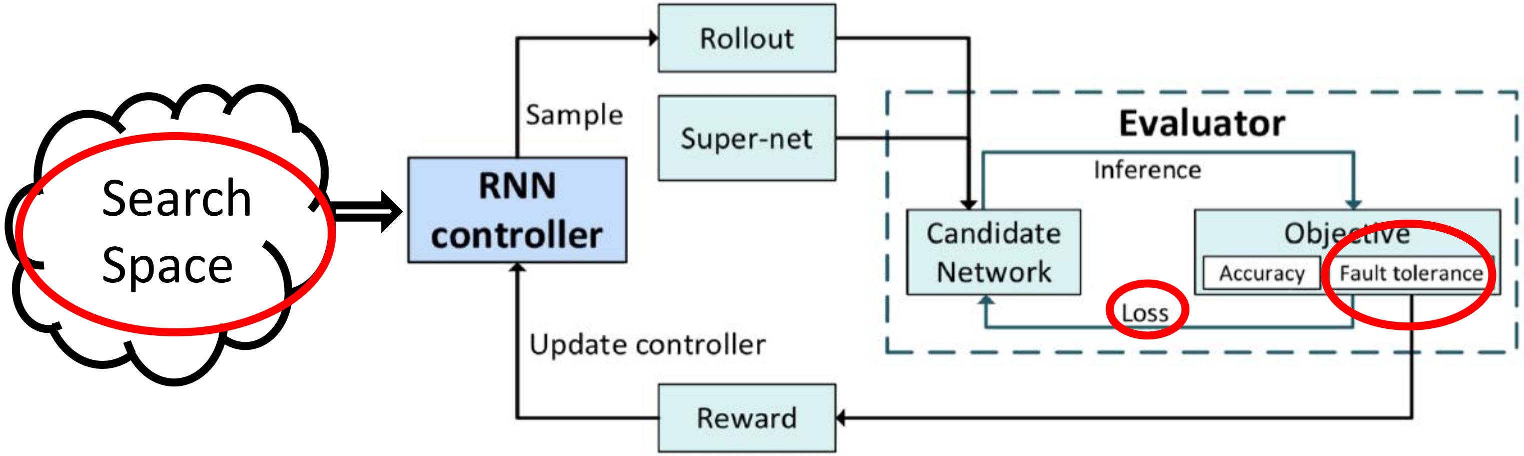


ResNet Block

√



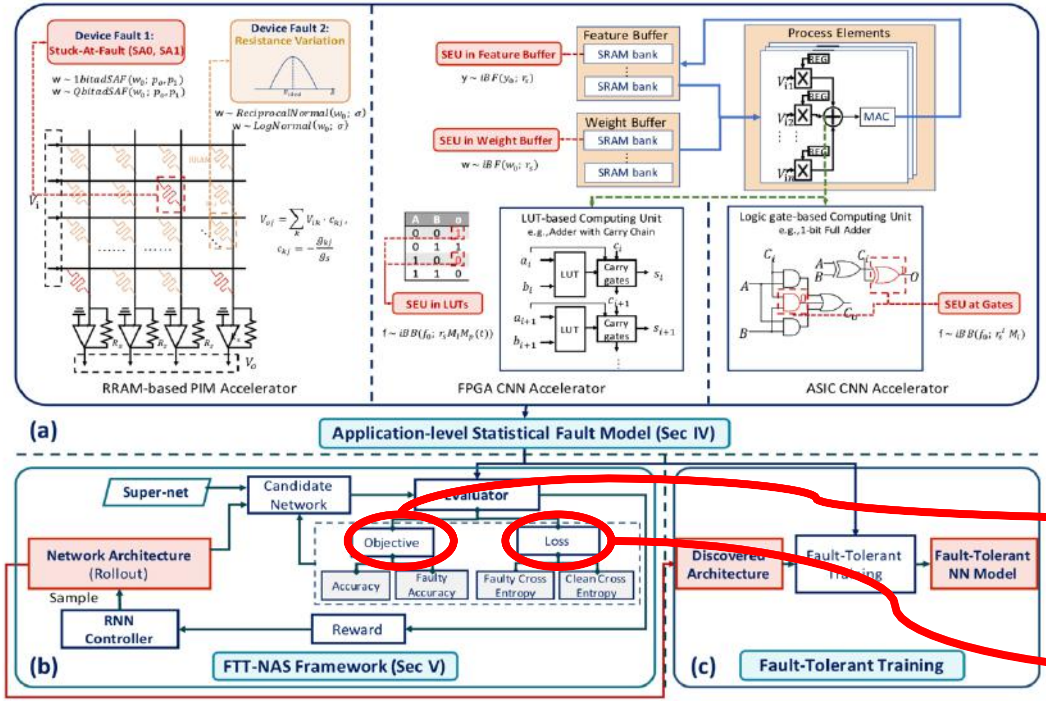
NAS Problem formalization



$$\min_{\alpha\in\mathcal{A}} \quad E_{x_v\sim D_v}[E_{f\sim F}[R(x_v,\mathrm{Net}(\alpha,w^*(\alpha)),f)]]$$
$$\text{s.t. } w^*(\alpha)=\mathrm{argmin}_w E_{x_t\sim D_t}[E_{f\sim F}[L(x_t,\mathrm{Net}(\alpha,w),f)]]$$

F:	Distribution characterized by the fault model
D_t:	Training dataset
D_v:	Valid dataset
A :	Architecture space
α:	An architecture rollout sampled from the architecture space
w:	network weights
$\mathrm{Net}(\alpha, w)$:	Candidate Network
$w^*(\alpha)$:	optimal network weights of α (get on the training dataset)
L:	Loss
R:	Reward

The Overall Workflow



a) Analyze and formalize the statistical fault models
  -> fault models (guide fault injection)

b) Run neural architecture search
  -> architecture

c) Fault tolerant training
  -> fault-tolerant model

reward

$$R = (1 - \alpha_r) * \mathrm{acc}_c + \alpha_r * \mathrm{acc}_f$$

loss

$$L = (1 - \alpha_l) * \mathrm{CE}_c + \alpha_l * \mathrm{CE}_f$$

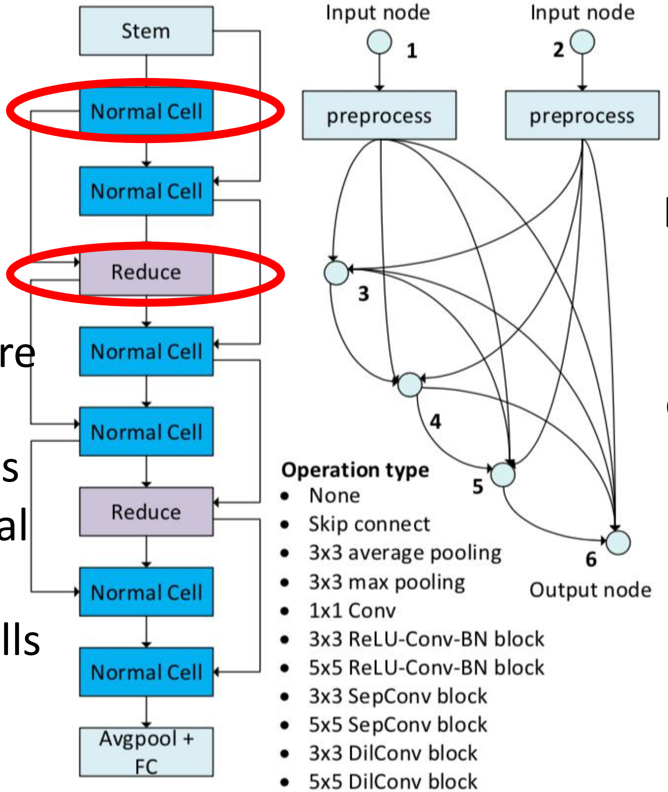
FTT-NAS: $\alpha_l > 0, \alpha_r > 0$. search with both fault-tolerant reward and fault-tolerant loss

FT-NAS (a degraded version): $\alpha_l = 0, \alpha_r > 0$ search with fault-tolerant reward and without fault-tolerant loss

Search Space Design

Artificial Macro Architecture

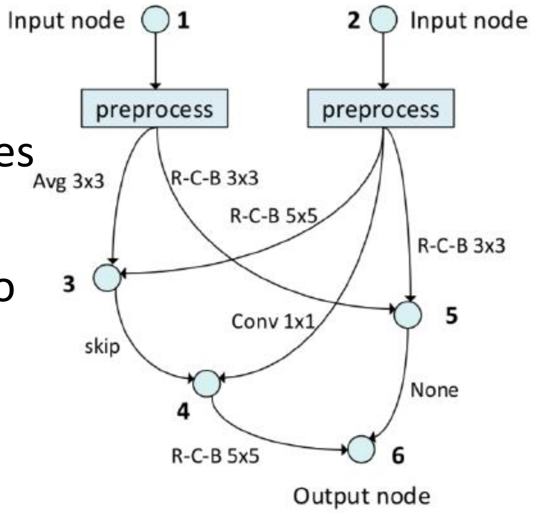
Divide Cells into Normal Cells and Reduce Cells



Operation type
- None
- Skip connect
- 3x3 average pooling
- 3x3 max pooling
- 1x1 Conv
- 3x3 ReLU-Conv-BN block
- 5x5 ReLU-Conv-BN block
- 3x3 SepConv block
- 5x5 SepConv block
- 3x3 DilConv block
- 5x5 DilConv block

Each cell has four nodes

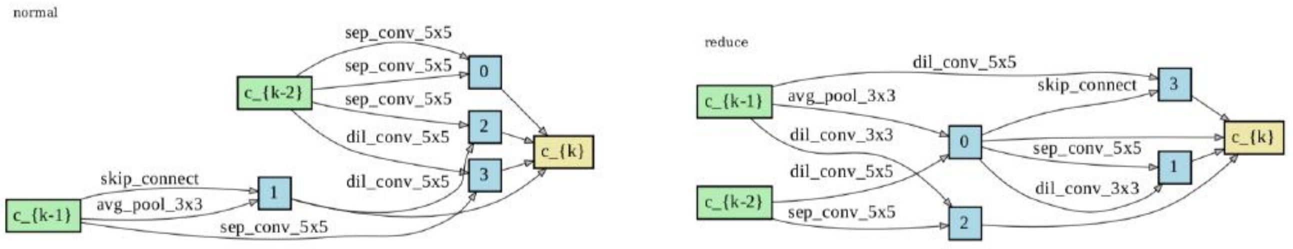
Every node selects two operations from all its predecessor nodes



Results: MiBB feature faults

Primitive Preference: Separable/dilation convolutions > normal convolutions

Since fewer MACs -> smaller equivalent feature error rate



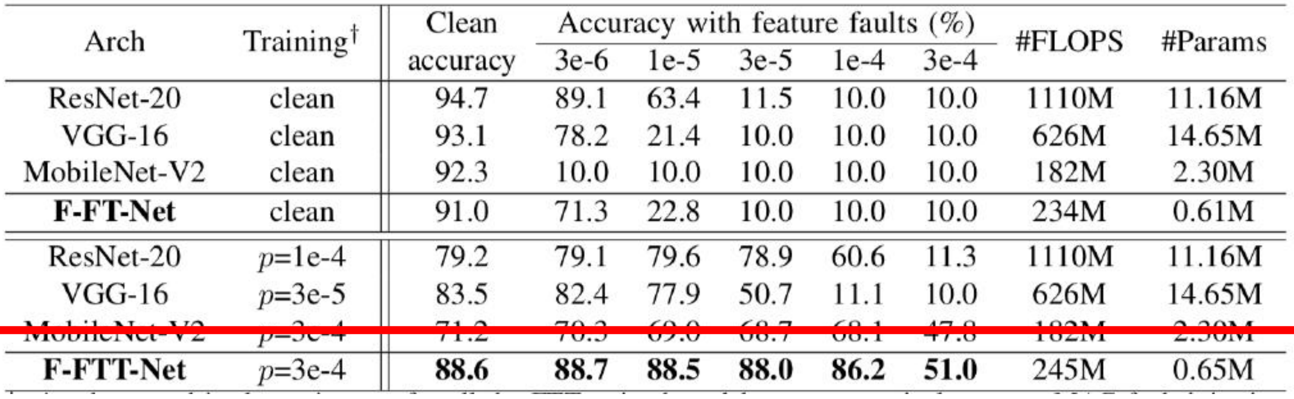
(a) (b)

Fig. 7: The discovered cell architectures under the MiBB feature fault model. (a) Normal cell. (b) Reduction cell.

Performance of stacks of 5 primitives (trained with p_m=1e-4)

Primitive	Fault acc (p_m=1e-4)
SepConv3x3	60.0%
SepConv5x5	65.1%
DilConv3x3	50.0%
DilConv5x5	56.3%

TABLE IV: Comparison of different architectures under the MiBB feature fault model

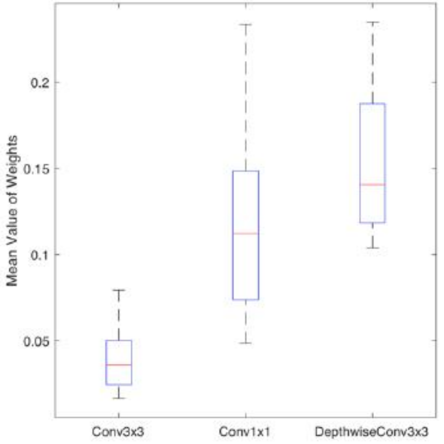


Arch	Training[†]	Clean accuracy	Accuracy with feature faults (%) 3e-6	1e-5	3e-5	1e-4	3e-4	#FLOPS	#Params
ResNet-20	clean	94.7	89.1	63.4	11.5	10.0	10.0	1110M	11.16M
VGG-16	clean	93.1	78.2	21.4	10.0	10.0	10.0	626M	14.65M
MobileNet-V2	clean	92.3	10.0	10.0	10.0	10.0	10.0	182M	2.30M
F-FT-Net	clean	91.0	71.3	22.8	10.0	10.0	10.0	234M	0.61M
ResNet-20	p=1e-4	79.2	79.1	79.6	78.9	60.6	11.3	1110M	11.16M
VGG-16	p=3e-5	83.5	82.4	77.9	50.7	11.1	10.0	626M	14.65M
MobileNet-V2	p=3e-4	71.2	70.3	69.0	68.7	68.1	47.8	182M	2.30M
F-FTT-Net	p=3e-4	**88.6**	**88.7**	**88.5**	**88.0**	**86.2**	**51.0**	245M	0.65M

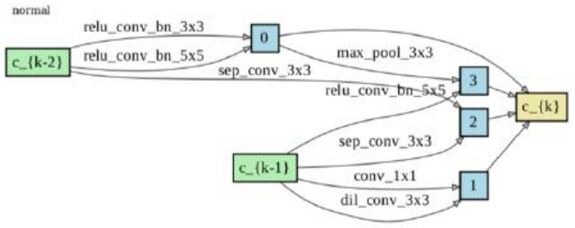
Results: adSAF weight faults

Primitive Preference:
Normal convolutions > separable/dilation convolutions

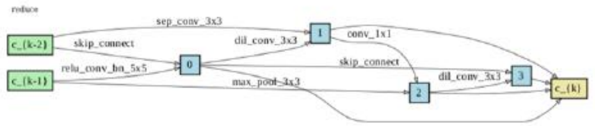
Inspection of weight distribution of different operations



Hypothesis:
Larger weight magnitudes -> Larger deviation caused by SAFs

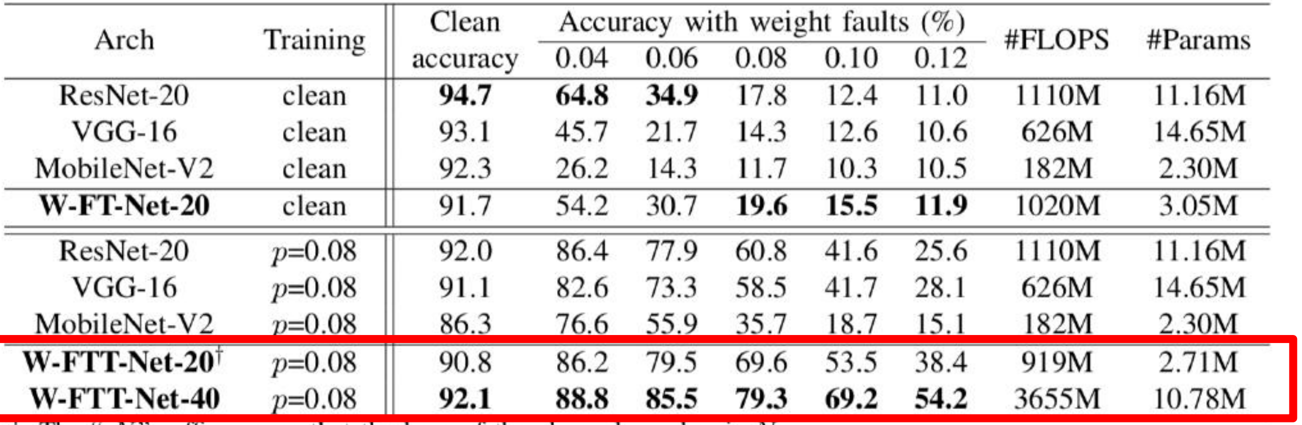


(a)



(b)

TABLE V: Comparison of different architectures under the adSAF weight fault model



Arch	Training	Clean accuracy	Accuracy with weight faults (%) 0.04	0.06	0.08	0.10	0.12	#FLOPS	#Params
ResNet-20	clean	**94.7**	**64.8**	**34.9**	17.8	12.4	11.0	1110M	11.16M
VGG-16	clean	93.1	45.7	21.7	14.3	12.6	10.6	626M	14.65M
MobileNet-V2	clean	92.3	26.2	14.3	11.7	10.3	10.5	182M	2.30M
W-FT-Net-20	clean	91.7	54.2	30.7	**19.6**	**15.5**	**11.9**	1020M	3.05M
ResNet-20	p=0.08	92.0	86.4	77.9	60.8	41.6	25.6	1110M	11.16M
VGG-16	p=0.08	91.1	82.6	73.3	58.5	41.7	28.1	626M	14.65M
MobileNet-V2	p=0.08	86.3	76.6	55.9	35.7	18.7	15.1	182M	2.30M
W-FTT-Net-20†	p=0.08	90.8	86.2	79.5	69.6	53.5	38.4	919M	2.71M
W-FTT-Net-40	p=0.08	**92.1**	**88.8**	**85.5**	**79.3**	**69.2**	**54.2**	3655M	10.78M

†: The "-N" suffix means that the base of the channel number is N.

Results: adSAF weight faults

NN model trained under 8bit-adSAF model, can also tolerate 1bit-adSAF faults, i.i.d random Bit-Flip faults

SAF in Multi-Bit RRAM

SAF in Single-Bit RRAM

SEU in Weight buffer

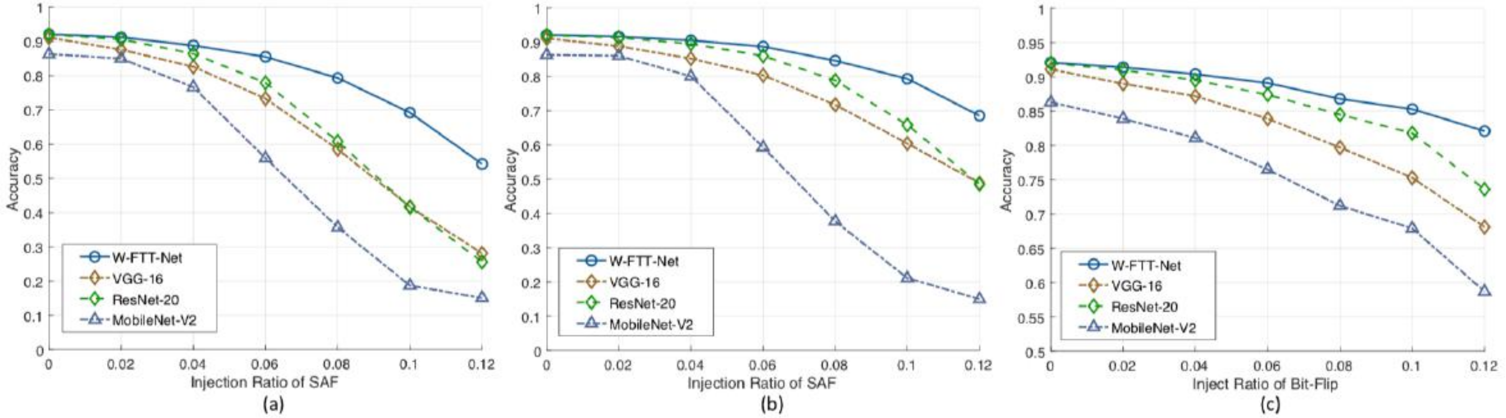


Fig. 9: Accuracy curve under different weight fault models. (a)W-FTT-Net under 8bit-adSAF model (b)W-FTT-Net under 1bit-adSAF model. (c)W-FTT-Net under iBF model

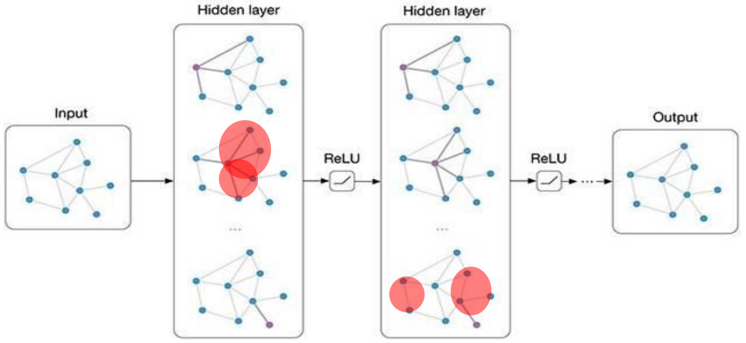
Future Work

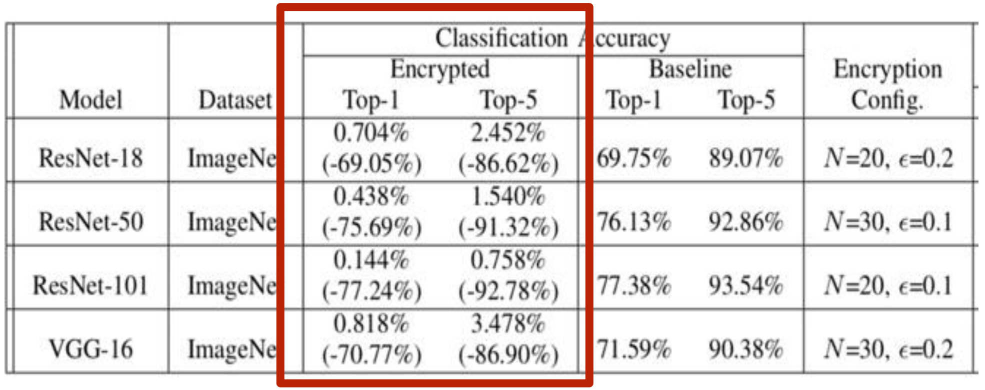
- Fault models
  - Now the modeling corresponds to **template-based FPGA accelerators**, for **instruction-based FPGA accelerators**, the fault model would be different since other factors should be considered:
    - Unrolling in kernel's spatial/channel dimensions leads to temporal reusing of hardware components, thus would leads to different fault propagation behavior for one feature value
    - Unrolling in OFM (output feature map) spatial/channel dimensions leads to repetitive error pattern in the OFM

Case Study 4: Secure NN on NVM/FPGA?

Analysis for NN Fault Problems

What's the general characteristics for various NN





Model	Dataset	Classification Accuracy				Encryption Config.
		Encrypted		Baseline		
		Top-1	Top-5	Top-1	Top-5	
ResNet-18	ImageNe	0.704% (-69.05%)	2.452% (-86.62%)	69.75%	89.07%	N=20, ϵ=0.2
ResNet-50	ImageNe	0.438% (-75.69%)	1.540% (-91.32%)	76.13%	92.86%	N=30, ϵ=0.1
ResNet-101	ImageNe	0.144% (-77.24%)	0.758% (-92.78%)	77.38%	93.54%	N=20, ϵ=0.1
VGG-16	ImageNe	0.818% (-70.77%)	3.478% (-86.90%)	71.59%	90.38%	N=30, ϵ=0.2

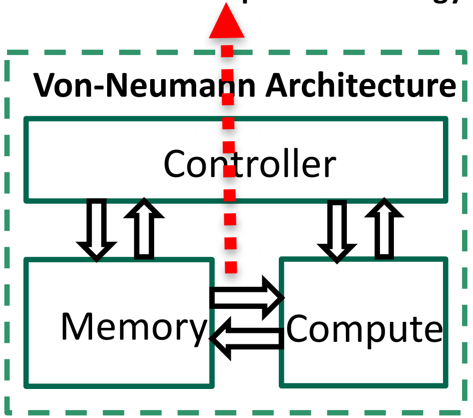
If we disturb N=20~30 picked parameters (< 0.1%) each layer, NN will failed its function.

KEY POINT: We observed that, most NN networks have a fraction of nodes (key point) which influence the performance a lot.

The Confidentiality Protection of NVM-based NN Systems

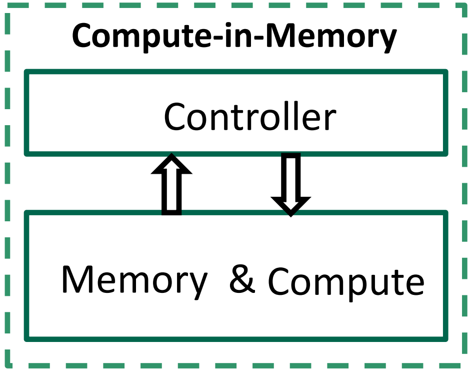
Memory Wall

Bottleneck of speed and energy



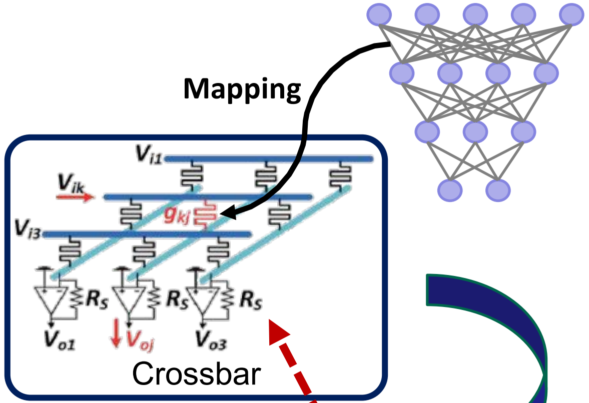
Compute-in-Memory

Merge the mem. & compute



Non-volatility

Weights will persist in memory



Key Point Encryption

?

138M

VGG-16
#Weights

01001…10010010100…

Encryption

11011…10010101001…

Adversary

Heavy Weights

Cause large encryption overhead

Motivating Encryption

To keep the data secure

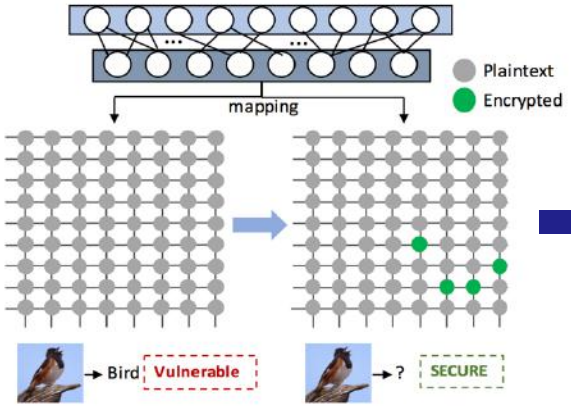
Confidentiality Attacks

Deployed NN models may be leaked

How We Distinguish the Key Points of NN Weights

Encryption Goal

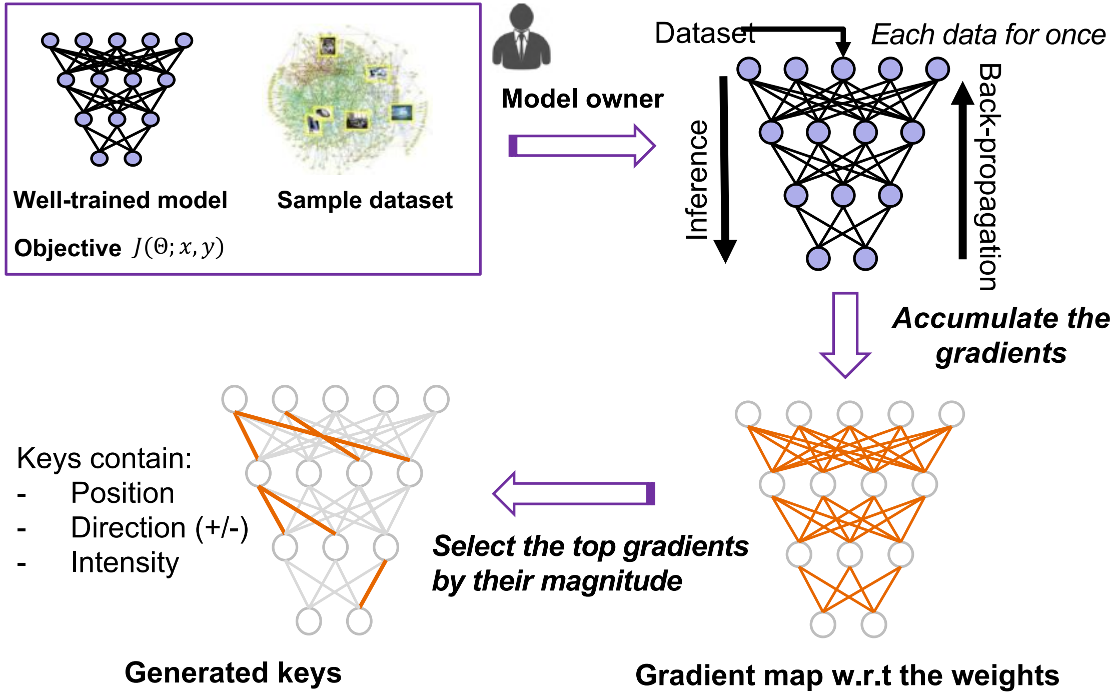
Encrypting key weights to disable the NN



$$\nabla_{\Theta} f = \lim_{\varepsilon \to 0} \frac{f(\Theta + \varepsilon) - f(\Theta)}{\varepsilon}$$

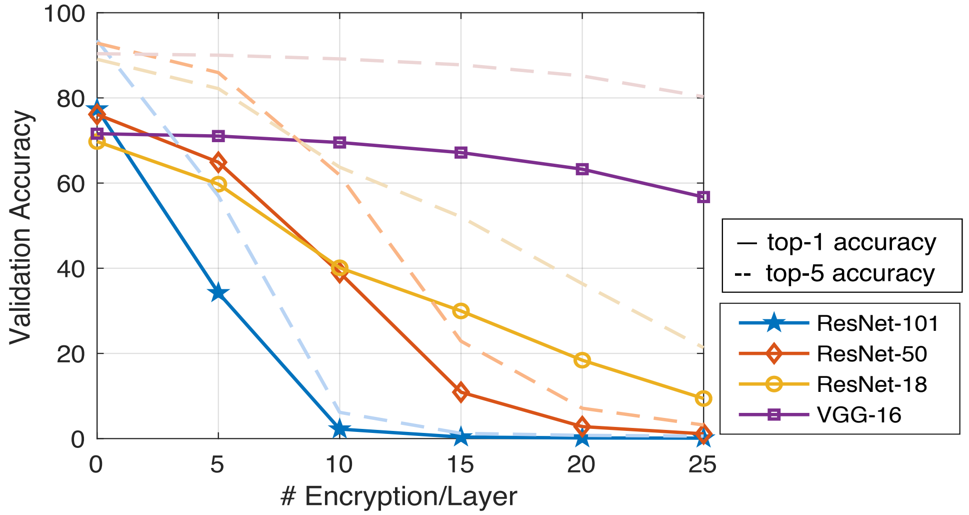
Gradient shows the speed of objective rising

Our Proposal **Sparse Fast Gradient Encryption (SFGE)**

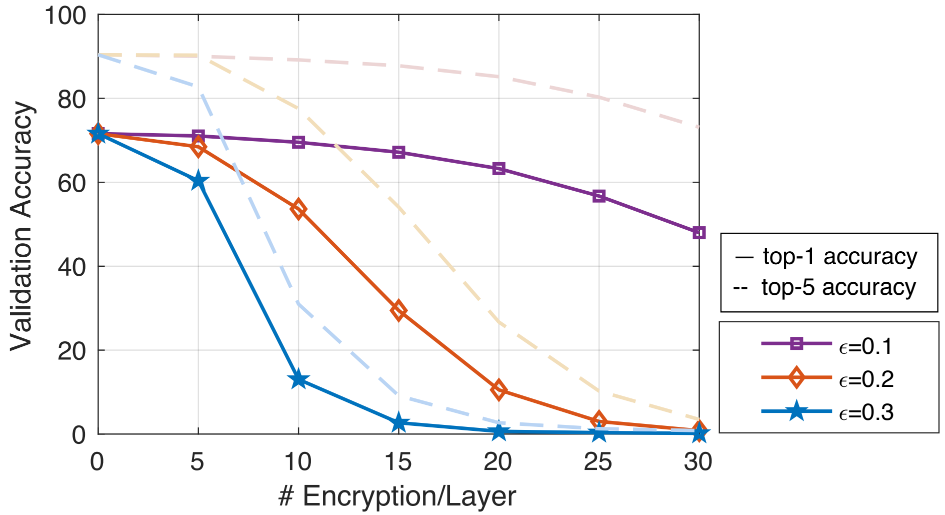


[1] [ICCAD 19] Yi Cai, Xiaoming Chen, Lu Tian, Yu Wang, Huazhong Yang, Enabling Secure in-Memory Neural Network Computing by Sparse Fast Gradient Encryption, in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019.

Experimental Results: the Effectiveness and Efficiency



Accuracy versus # **encryption *N***



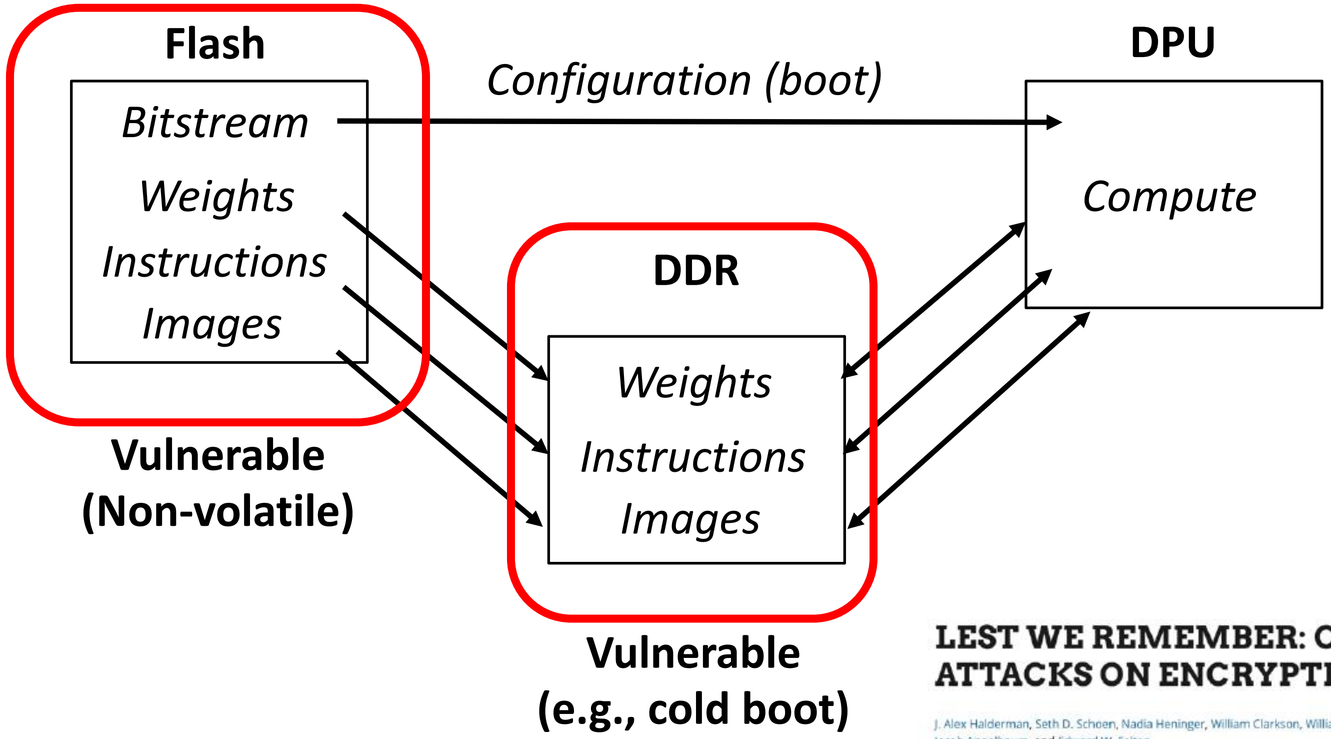
VGG-16 accuracy versus # encryption *N* under **different intensity ϵ**

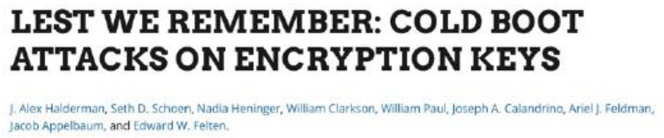
Only Encrypting 20~30 Weights per Layer in the Neural Network can Strictly Protect the model Confidentiality

[1] [ICCAD 19] Yi Cai, Xiaoming Chen, Lu Tian, Yu Wang, Huazhong Yang, Enabling Secure in-Memory Neural Network Computing by Sparse Fast Gradient Encryption, in IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019.

Does FPGA design secure enough?

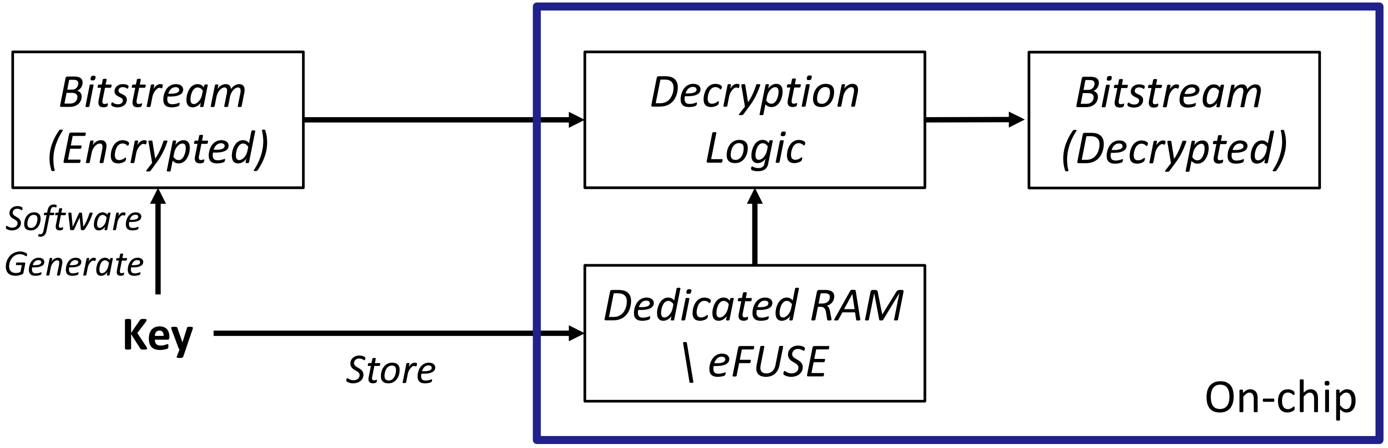
A typical workflow of FPGA neural network accelerator




LEST WE REMEMBER: COLD BOOT ATTACKS ON ENCRYPTION KEYS

J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten.

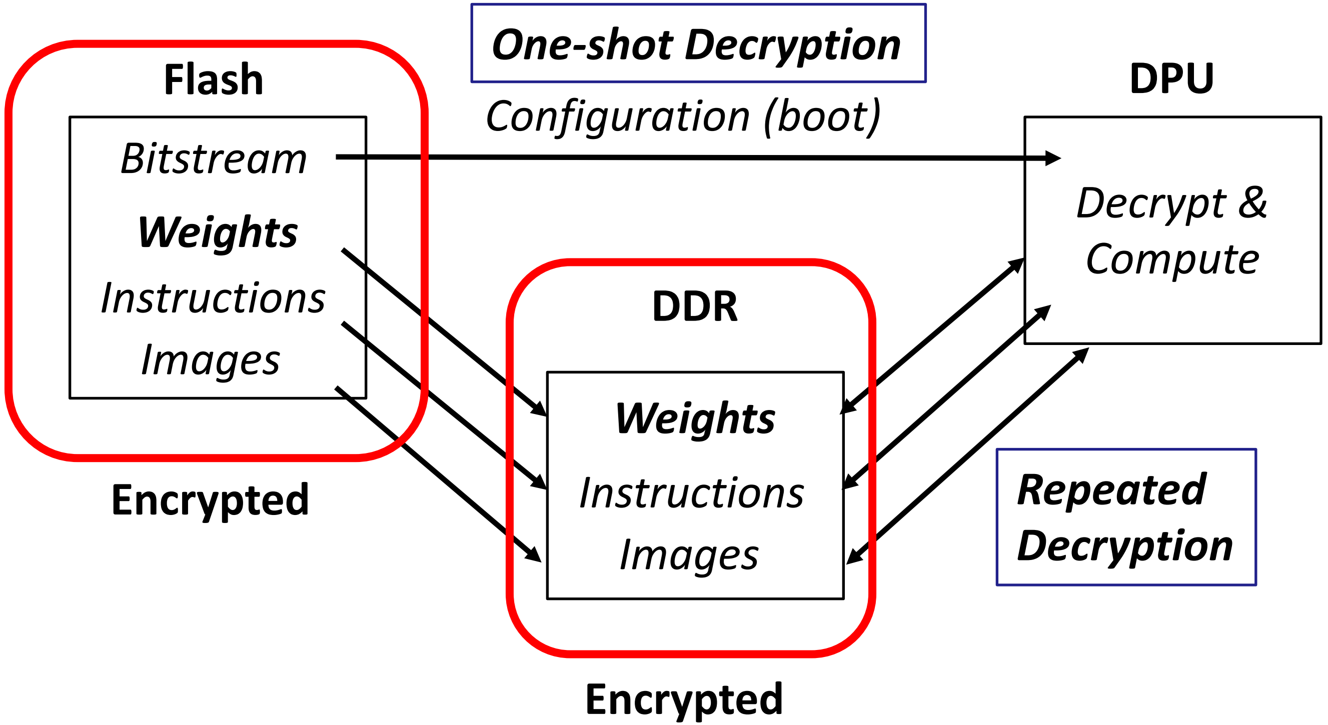
Current process to protect security of FPGA bitstreams



However, how to secure the deployed NN models?

https://www.xilinx.com/support/documentation/application_notes/xapp1267-encryp-efuse-program.pdf

Our method can be transferred to secure FPGA NN designs



Decrypting part of the weights – lower latency, more secure

Summary

What we have or we can expect (power efficiency, computation power)

- AI platforms are covering everywhere.
- We already have many methods to make the chips powerful.

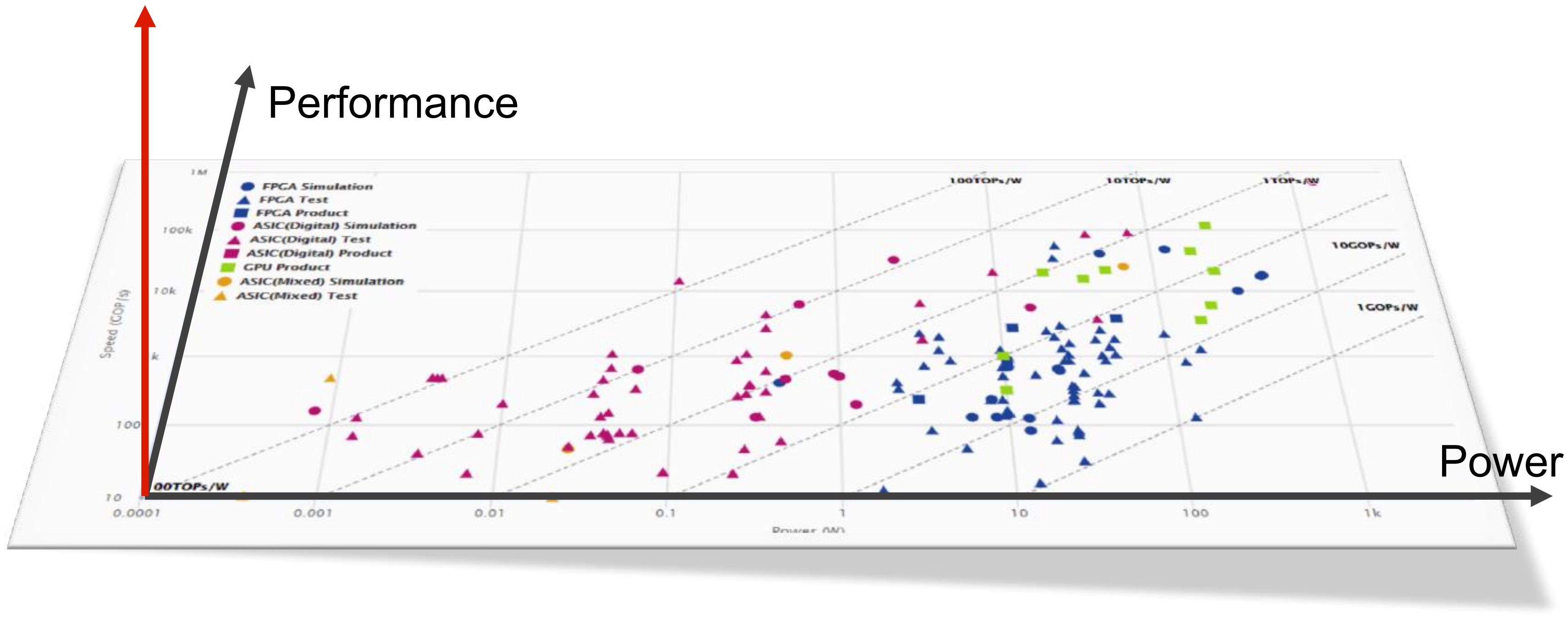
What's next: (user friendly, customizable, low cost, real-time, reliable, secure…)

- Are the chips ready for real applications?
- Can we use the large 'TOPs' to the extreme?
- Are the systems robust enough?
- Are the systems secure enough?

Learning capability: application, algorithm, and hardware arch

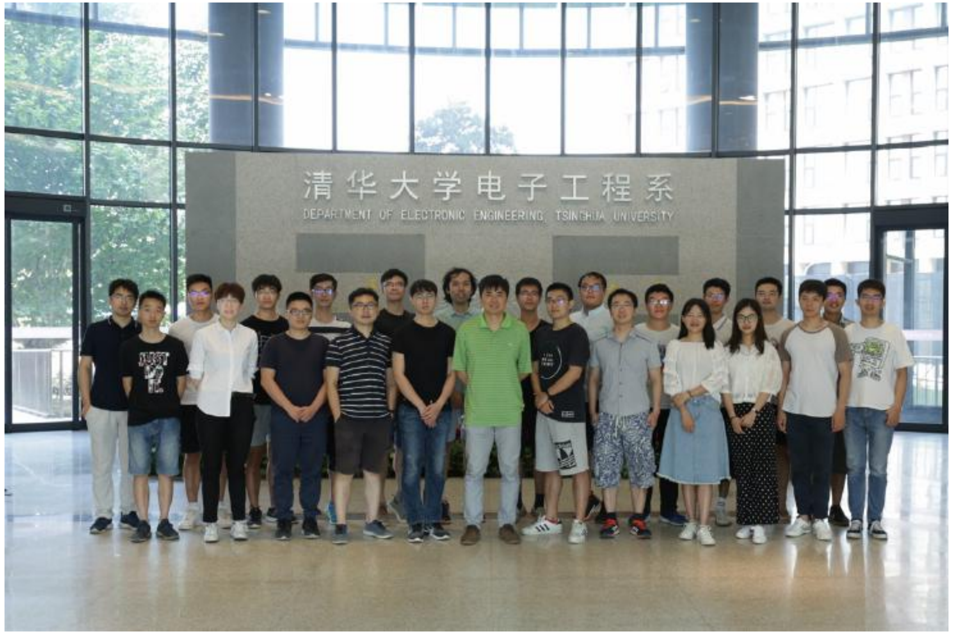
Users care much more than energy efficiency

Flexibility, Security, Robustness, …



Acknowledgement

To my students and collaborators



To supporting companies and agencies

XILINX

Micron

TOYOTA

HUAWEI

Microsoft

SYNOPSYS

IBM

Thanks for your attention!

yu-wang@tsinghua.edu.cn