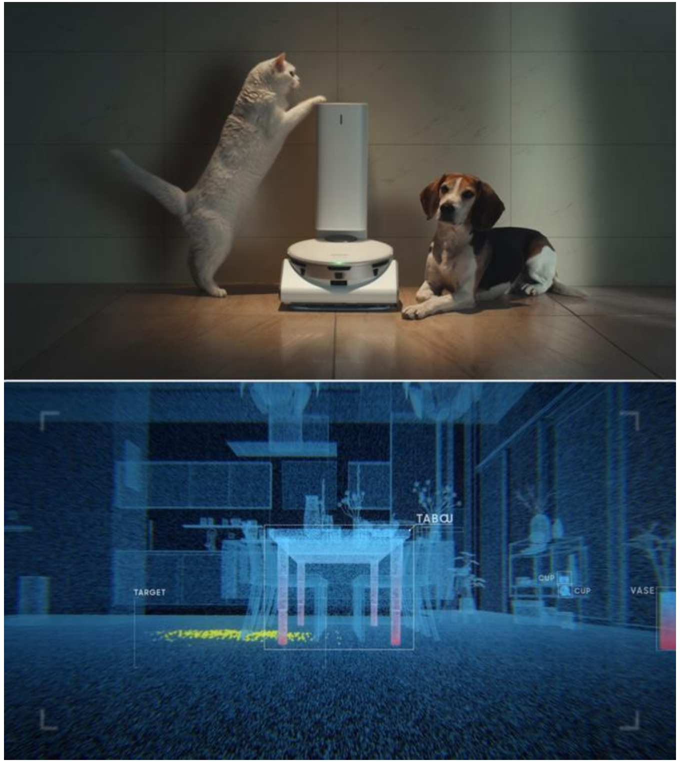
Samsung Research

On-Device AI for Mobile and Consumer Devices: From DNN Model Compression to Domain-Specific Accelerators

Daehyun Kim

Samsung AI at CES 2021

Robots

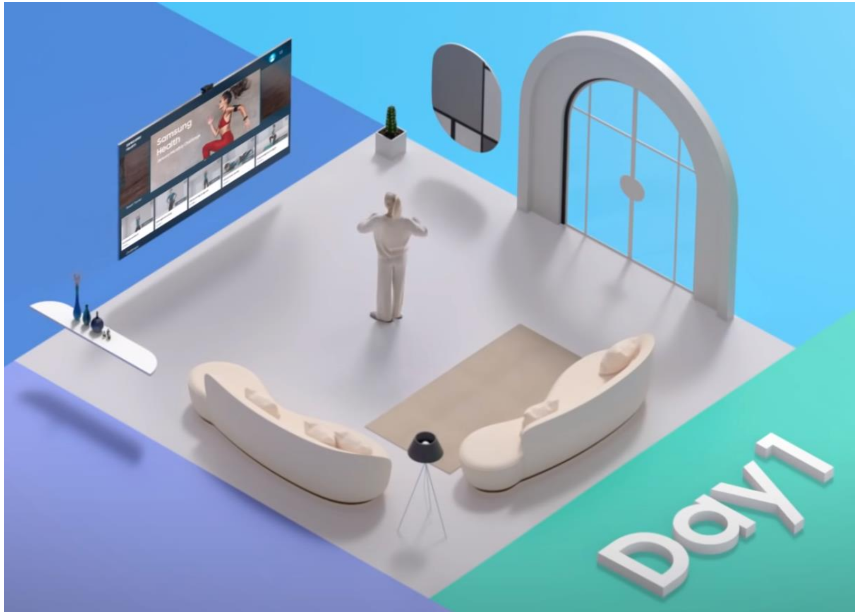


JetBot 90 AI+

Bot Care & Bot Handy

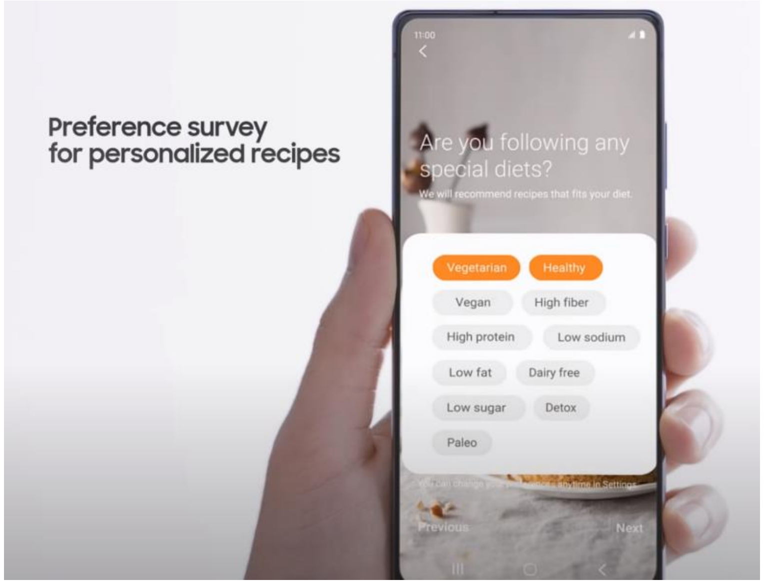
TVs

Neo Quantum Processor



Smart Trainer

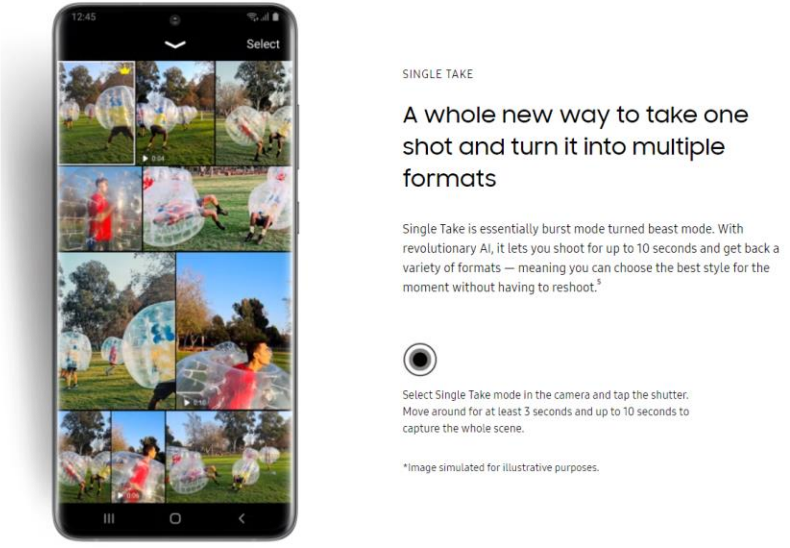
Home Appliances



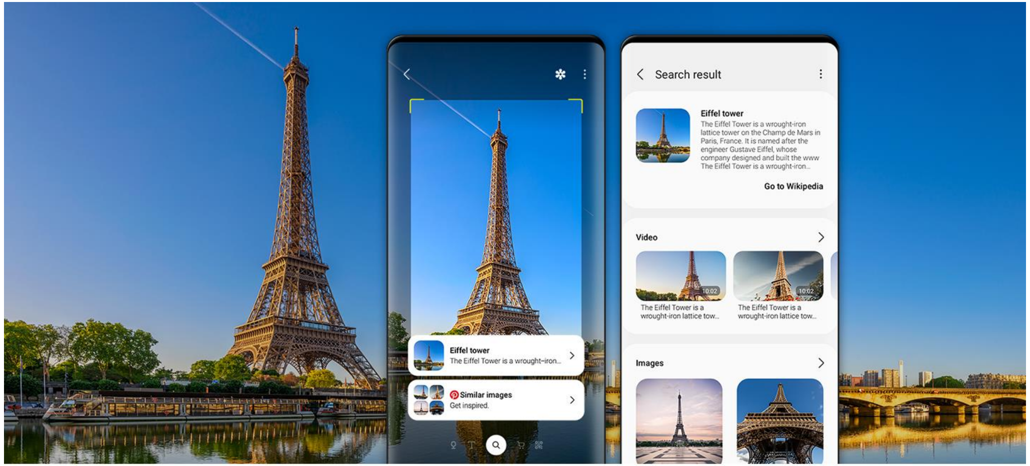
SmartThings Cooking

Family Hub

Mobile Phones



Single Take Camera



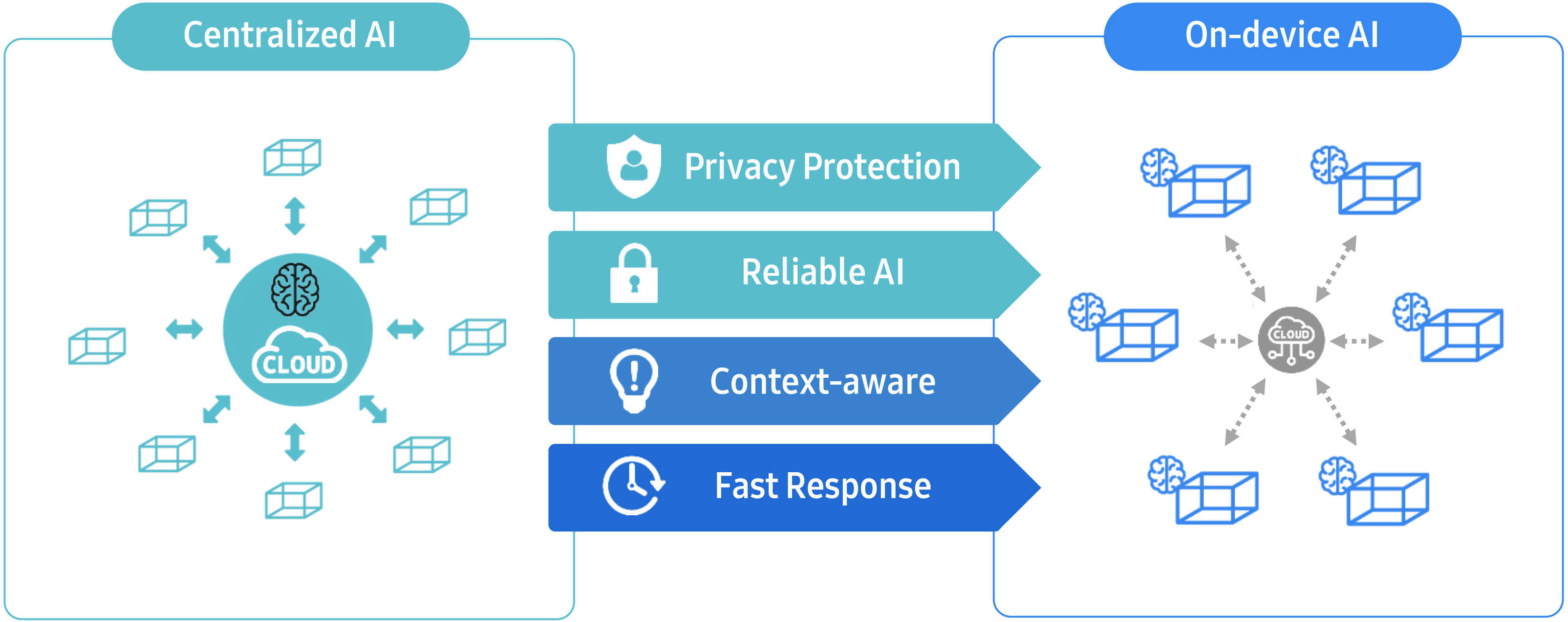
Bixby Vision

Our AI Vision

Why On-device AI ?

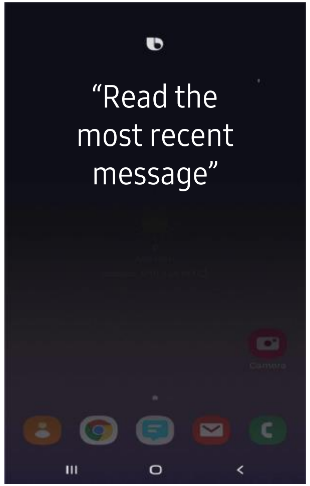
Two ways to implement AI : Cloud vs. On-device

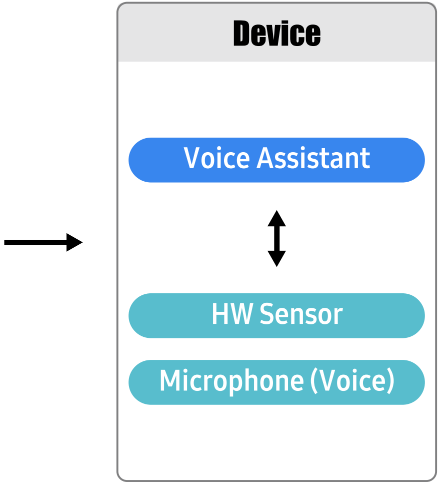
On-device AI can enhance cloud AI in user experience



Enhancing User Experience : Privacy

Important Information about users can be processed on device





No need to

- send the voice to cloud
- send ‘most recent message’ to cloud

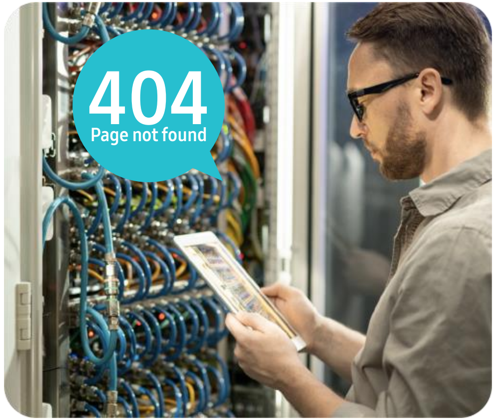
- send private information to cloud

Enhancing User Experience : Reliable AI

No matter what happens on network connection or servers, it works!

In elevator

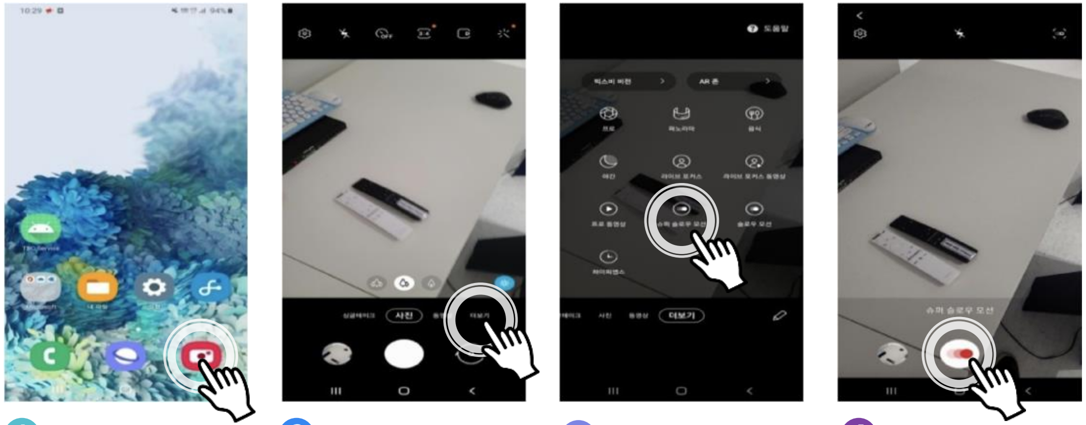
In flight



Server error

Enhancing User Experience : Context-aware

Device knows you fairly well ⟶ AI can suggest the best option



1 **Launch**
- Camera

2 **Select**
- More

3 **Select**
- Super Slow Motion

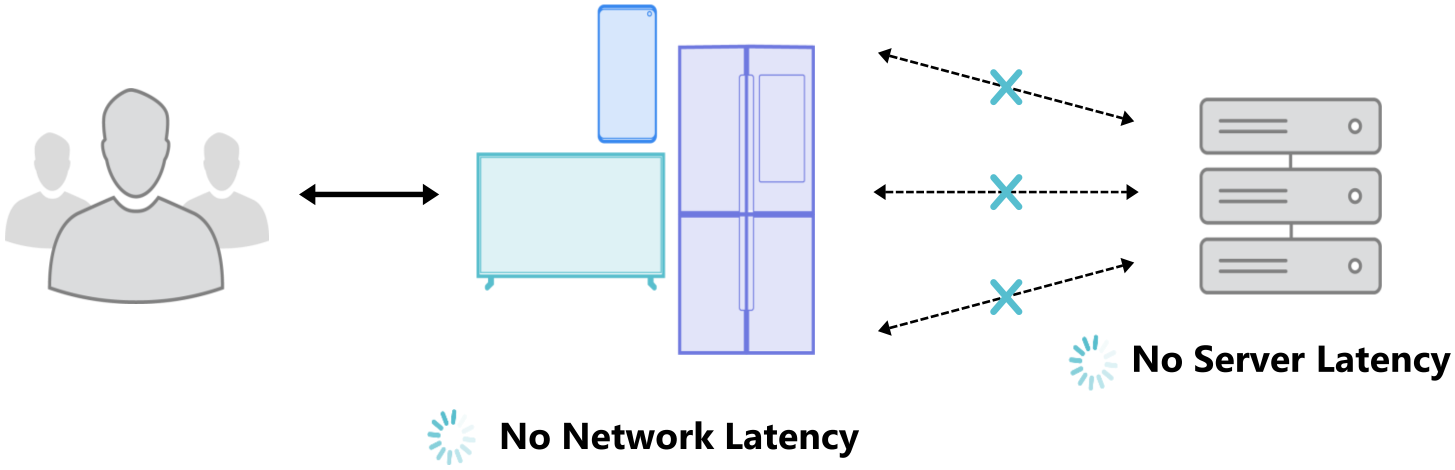
4 **Start to Record**

Recommendation

You can use this with
"Record a Super slow motion video"

Enhancing User Experience : Fast response

Without latency in network and servers, it executes considerably quickly



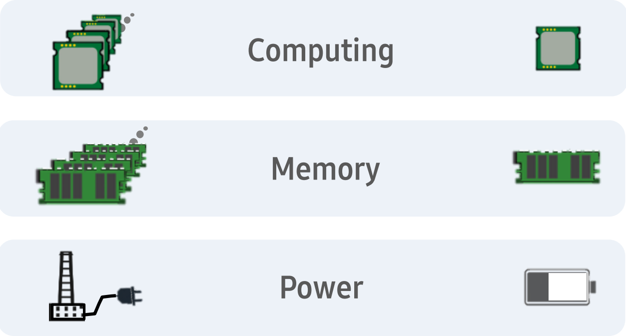
On-device AI Research

On-device AI Challenges

On-device AI is required to run on limited resources, compared to cloud

Clouds

Resource Gap



Devices

Our On-Device AI Approach

AI
+
SW
+
HW

Co-Design

① Neural Network Model Optimization

Model Compression
Pruning, Quantization, ...

Model Architecture
NAS, Multi-taking, ...

→ FleXOR
BiQGEMM

② AI System Software

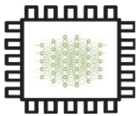
NN Compiler/Runtime
Model Analysis, Scheduling,
Memory Optimization, ...

NN Pipeline
Multi-Model Pipeline,
Multi-Device Pipeline

NN Training
On-Device Training

→ nnStreamer
nnTrainer

③ AI HW Accelerator



Neural Processor
Power Efficient Specialized Accelerator HW IP

→ SNP

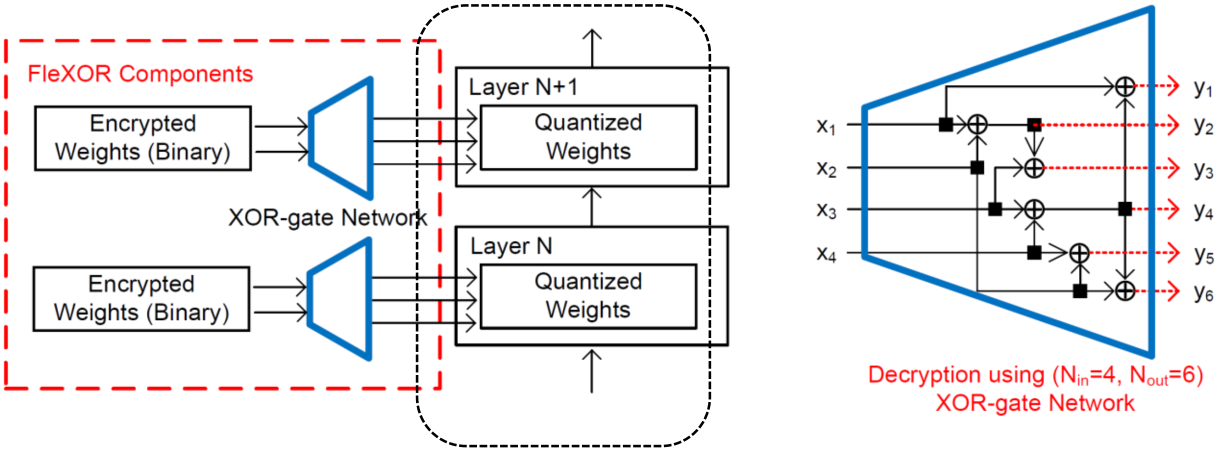
FleXOR

FleXOR: Trainable Fractional Quantization

D. Lee, S Kwon, B. Kim, Y Jeon, B Park, J Yun
Neurips 2020

FleXOR

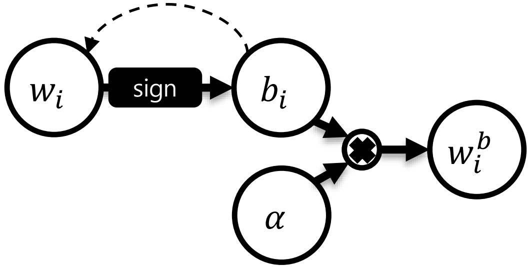
Samsung Research



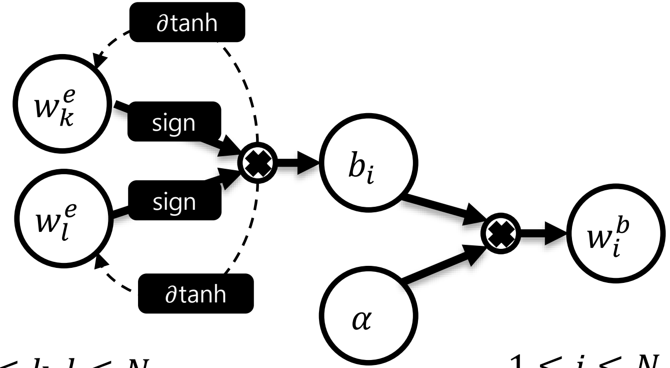
- A flexible encryption algorithm/architecture (called "FleXOR") to enable fractional sub 1-bit numbers to represent each weight
- Contributions
  - XOR-based encryption of quantized bits enhances compression ratio
  - XOR-aware training algorithm learns encrypted weights
  - High model accuracy with sub 1-bit quantization

Differentiable XOR-gate Network

Existing STE (straight-through estimator)

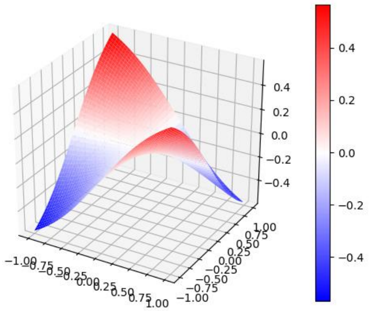


Our FleXOR

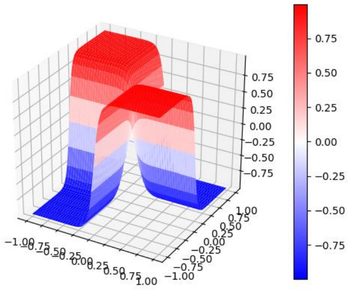


$1 \le k, l \le N_{in}$

$1 \le i \le N_{out}$



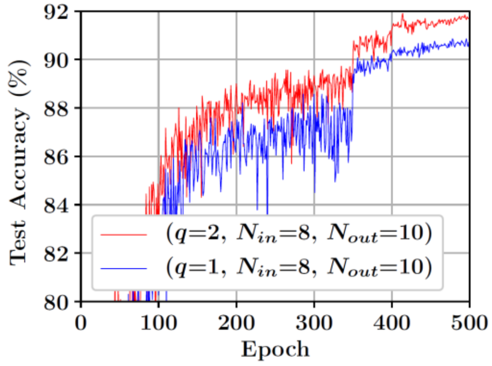
$y = tanh(x_1) \times -tanh(x_2)$

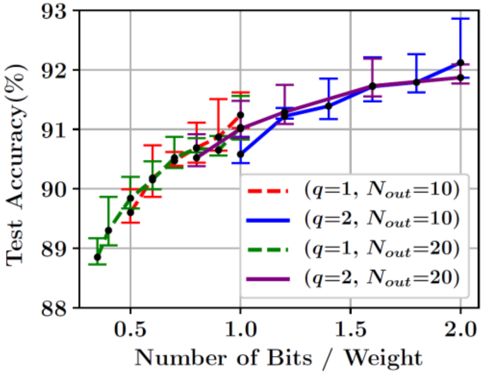


$y = tanh(\mathbf{10}x_1) \times -tanh(\mathbf{10}x_2)$

FleXOR should be able to select the best out of $2^{N_{in}}$ possible outputs that are randomly selected from larger $2^{N_{out}}$ search space.

Experiments





- FleXOR allows reduced memory footprint and bandwidth which are critical for energy-efficient inference designs.

- Even though achieving the best accuracy for 1.0 bit/weight is not the main purpose
(e.g., XOR gate may be redundant for N_{in}=N_{out}), FleXOR shows the minimum accuracy drop.

Table 1: Weight compression comparison of ResNet-20 and ResNet-32 on CIFAR-10. For FleXOR, we use warmup scheme, $S_{\tanh}$=10, and N_{out}=20.

	ResNet-20			ResNet-32		
	FP	Compressed	Diff.	FP	Compressed	Diff.
BWN (1 bit)	92.68%	87.44%	-5.24%	93.40%	89.49%	-4.51%
BinaryRelax (1 bit)	92.68%	87.82%	-4.86%	93.40%	90.65%	-2.80%
LQ-Net (1 bit)	92.10%	90.10%	-1.90%	-	-	-
DSQ (1 bit)	90.70%	90.24%	-0.56%	-	-	-
FleXOR (1.0 bit)	91.87%	90.44%	-1.47%	92.33%	91.36%	-0.97%
FleXOR (0.8 bit)		89.91%	-1.90%		91.20%	-1.13%
FleXOR (0.6 bit)		89.16%	-2.71%		90.43%	-1.90%
FleXOR (0.4 bit)		88.23%	-3.64%		89.61%	-2.72%

Table 3: Weight compression comparison of ResNet-18 on ImageNet.

Methods	Bits/Weight	Top-1	Top-5	Storage Saving
Full Precision [10]	32	69.6%	89.2%	1×
BWN [20]	1	60.8%	83.0%	∼ 32×
ABC-Net [18]	1	62.8%	84.4%	∼ 32×
BinaryRelax [26]	1	63.2%	85.1%	∼ 32×
DSQ [7]	1	63.7%	-	∼ 32×
FleXOR (N_{out} = 20)	0.8	63.8%	84.8%	∼ 40×
	0.63 (mixed)[2]	63.3%	84.5%	∼ 50.8×
	0.6	62.0%	83.7%	∼ 53×

[2]To 4 groups of 3×3 conv layers in ResNet-18 (except the first conv layer connected to the inputs), we assign 0.9, 0.8, 0.7, and 0.6 bits/weight, respectively. To the remaining 1×1 conv layers (performing downsampling), we assign 0.95, 0.9, and 0.8 bits/weight, respectively.

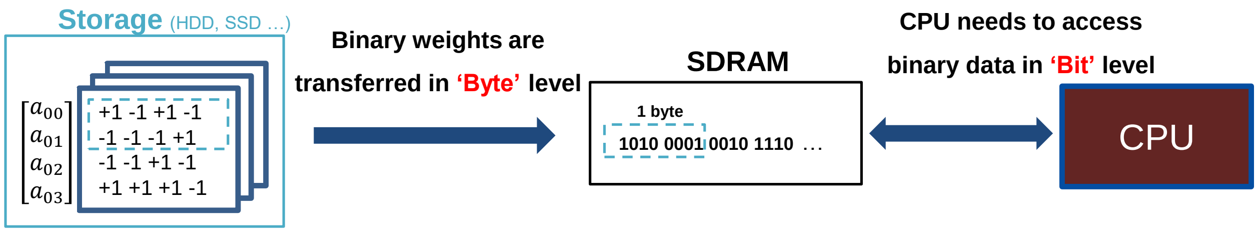
BiQGEMM

BiQGEMM: Matrix Multiplication with Lookup Table for Binary-Coding-based Quantized DNNs

Y. Jeon, B. Park, S. Kwon, B. Kim, J. Yun, D. Lee
SC20

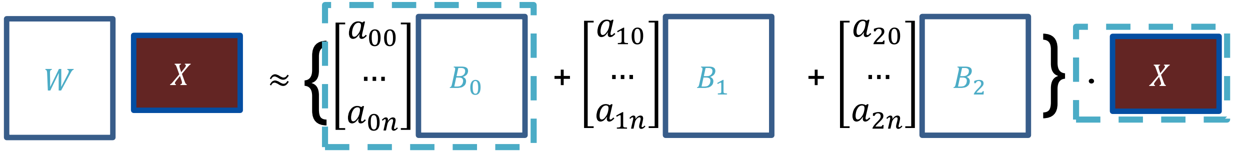
BiQGEMM

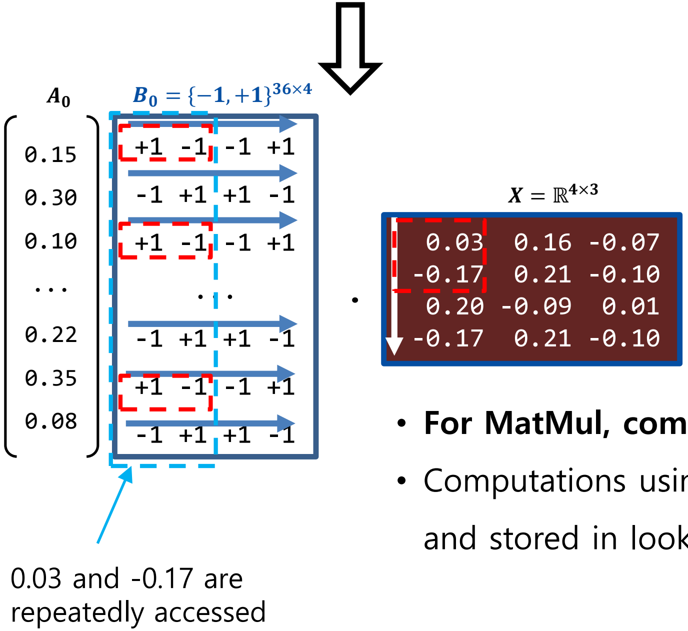
Samsung Research



- **Previous Binary-Code Quantization Implementation**
  - Special H/W for MATMUL operation with Quantized weights
    - For non-uniform quantization, CPU/GPU/NPU needs to perform on-chip dequantization in practice.
    - Binary-code is then only to reduce memory requirements (not latency) without special H/W
- **BiQGEMM**
  - Dedicated binary-coding-based matrix multiplication unit kernel design using lookup tables
  - CPUs and GPUs can utilize quantized matrices to improve performance

BiQGEMM

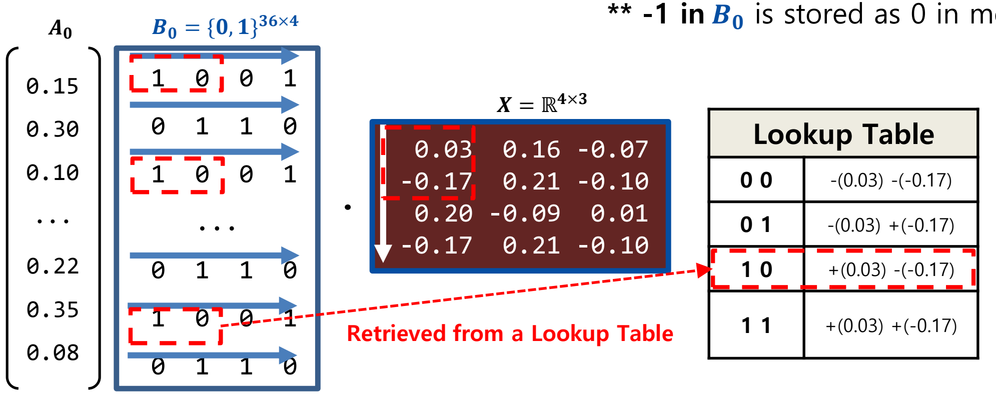




- **For MatMul, computations in [red dashed box] are redundant**
- Computations using X and B values are performed in advance and stored in lookup tables

BiQGEMM

** -1 in B_0 is stored as 0 in memory

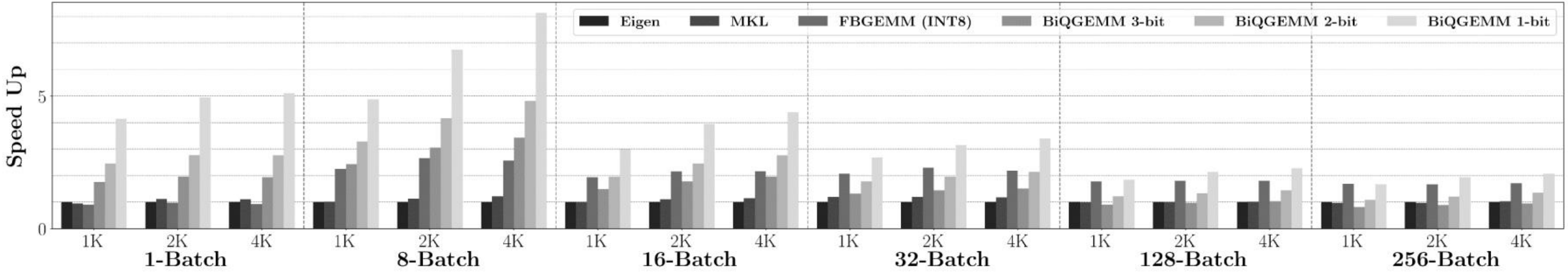


- MatMul computations are replaced with pre-computation and Lookup table access
  - Lookup-table size is empirically determined.
  - In practice, 8 bits are used as an index with 256 entries
- No redundant computations
  - Number of float multiplications are greatly reduced
  - B matrix is now access in 'Byte' level (No bit-level operation)

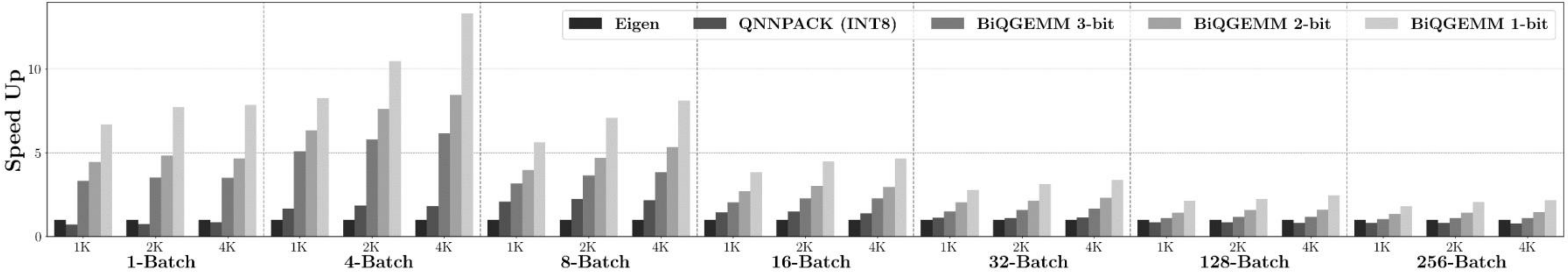
Experimental Results

Samsung Research

- Speedup over *Eigen* using 1-thread. Matrix size is given as *m-by-1K*. Output size *(m)* and batch size are annotated along the horizontal axis.



(a) PC (i7-7700)



(b) Mobile (Coretex-A76)

nnStreamer

Linux Foundation AI Project for Efficient Machine Learning Pipeline Development and Execution

M. Ham, J. Moon, G. Lim, S. Woo, W. Song, J. Jung, H. Ahn, P. Kapoor, D. Chae, G. Jang, Y. Ahn, J. Lee

https://nnstreamer.ai/

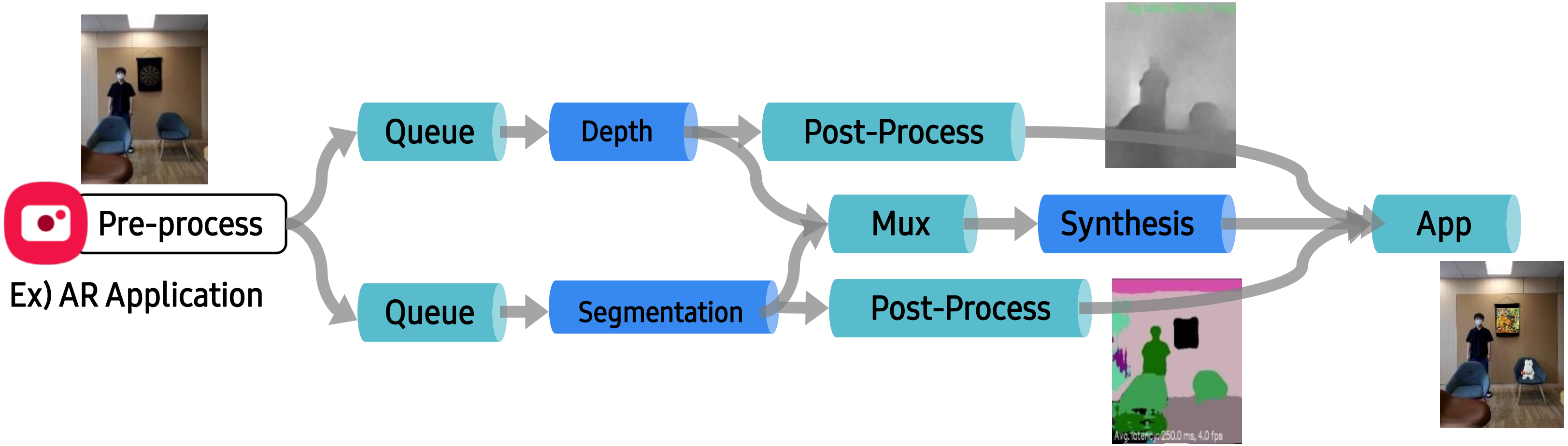
https://github.com/nnstreamer/nnstreamer

Neural Network Pipeline

Samsung Research

Efficient and flexible pipelines for Neural Networks

- 1000s Lines of Code → 10s Lines of Pipeline Description
- Manual Parallelization → Automatic Pipeline Parallelization
- Direct media/hardware Optimization → Reusable Module for media/hardware



Do Not Reinvent the Wheel

GStreamer

- https://gstreamer.freedesktop.org
- Open source multimedia pipeline framework
- Library for constructing graphs of media-handling components

nnStreamer

- But, perfect the wheel!
- Extension of GStreamer for AI processing
  - Neural network as another media filter
  - Neural network data as tensor stream

Neural Network to GStreamer Pipeline

Samsung Research

Code: `src ! sink`

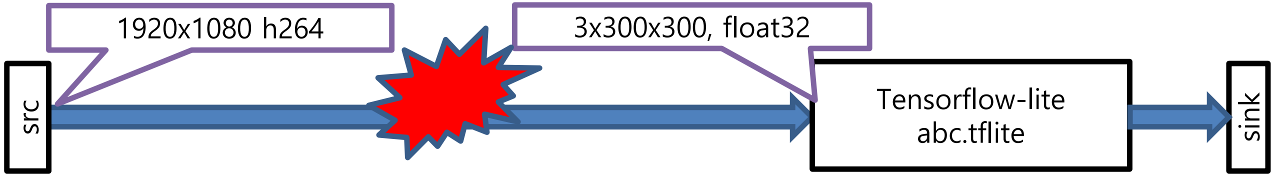
Code: `src ! tensor_filter mode=tensorflow-lite model=abc.tflite ! sink`

Code: `src ! tensor_filter mode=tensorflow model=def.pb ! sink`

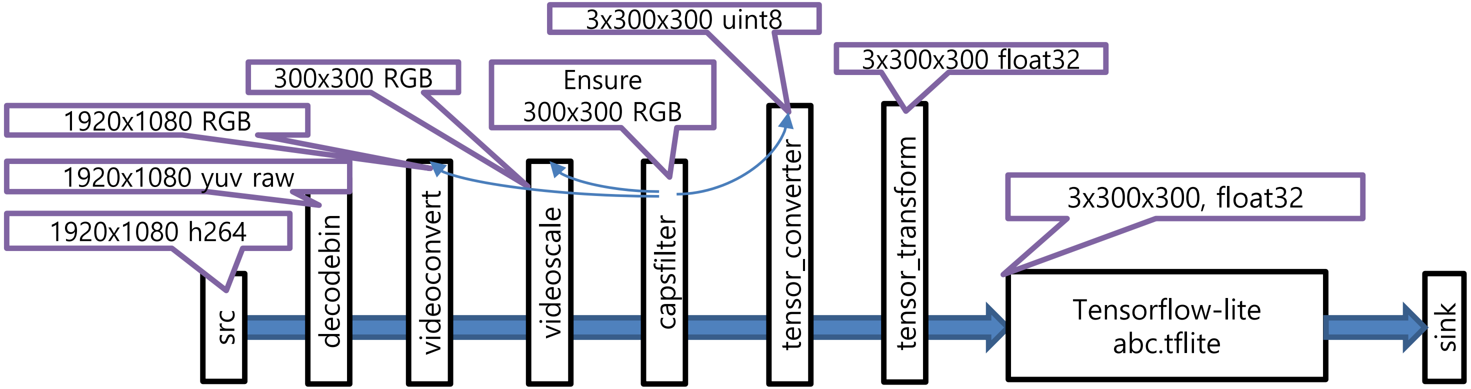
Code: `src ! tensor_filter mode=caffe2 model=ghi.pb ! sink`

Data Conversion

Samsung Research

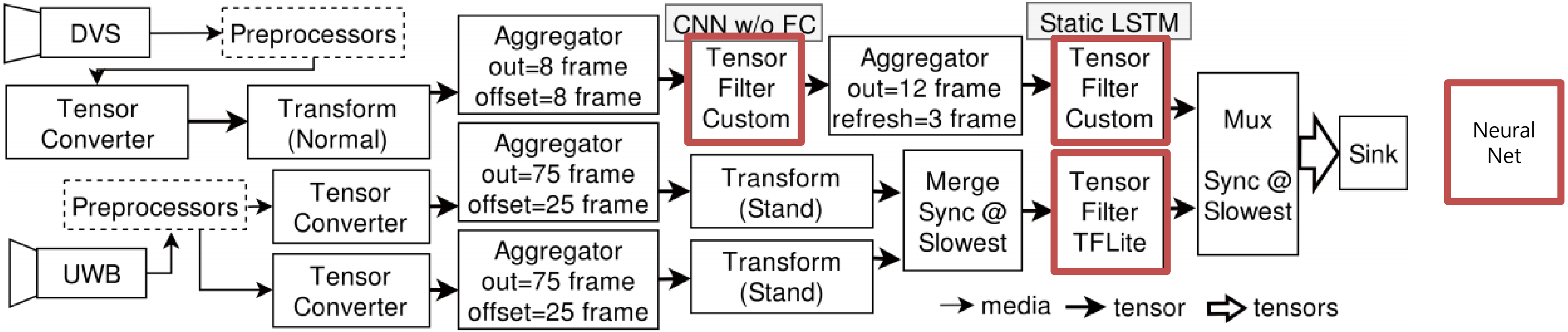


Code: src ! tensor_filter mode=tensorflow-lite model=abc.tflite ! sink



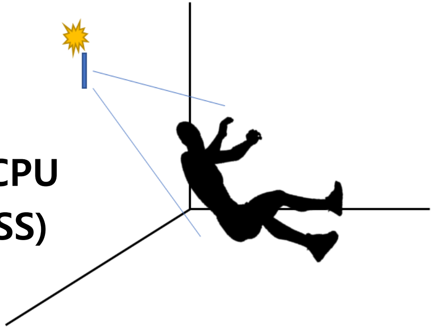

```
Code: src ! decodebin ! videoconvert ! videoscale
          ! video/x-raw,format=RGB,width=300,height=300
          ! tensor_convert
          ! tensor_transform mode=typechg option=float32
          ! tensor_filter mode=tensorflow-lite model=abc.tflite ! sink
```

Example: Activity Recognition Sensors




```
$ gst-launch-1.0 tensor_mux name=mux ! fakesink
! tensor_converter ! tensor_trans mode=arith
! tensor_aggregator in=1 out=8 flush=8
! tensor_filter frame=custom m=./cnn.so
! tensor_aggregator in=1 out=12 flush=3
! tensor_filter frame=custom m=./lstm.so
! mux.sink_0
tensor_merge name=merge sync-mode=slowest
! tensor_filter frame=tflite m=./uwb.tflite
! mux.sink_1
multifilesrc location="./input_uwb0_%04d.data"
! tensor_converter dim=1:1:32:1 type=float32
! tensor_aggregator in=1 out=75 flush=25
! tensor_trans mode=stand ! merge.sink_0
multifilesrc location="./uwb1_%04d.data"
! tensor_converter dim=1:1:32:1 type=float32
! tensor_aggregator in=1 out=75 flush=25
! tensor_transform mode=stand ! merge.sink_1
```

- **~1000 lines ➔ 16 lines**
- **@ 30FPS input, 90.4% → 51.4% CPU**
- **~40 MiB → ~17 MiB Memory (RSS)**



nnTrainer

nnTrainer: Towards On-Device Learning for Personalization

Submitted to ATC 21

https://github.com/nnstreamer/nntrainer

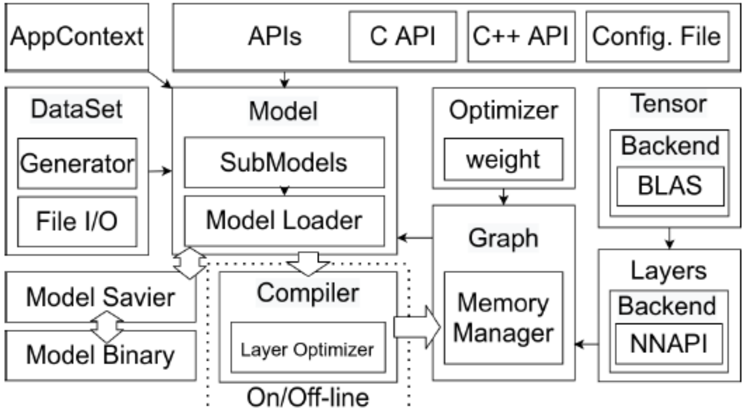
Light-Weight On-Device Training Framework

Samsung Research

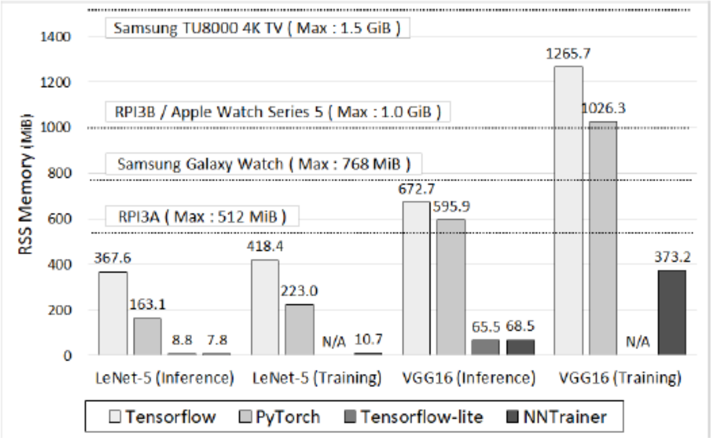
- **nnTrainer**
  - Software framework to train neural network on embedded devices
- **Personalization**
  - As users keep using AI applications, they get
    - Faster (ex. 100ms to 50ms)
    - More accurate (ex. 88% to 95%)
    - Personalized (ex. A Dog to My Dog)
  - While providing privacy
    - Personal data stay at user devices
- **Challenges**
    - Small data for training
    - Limited compute/memory resources

nnTrainer Overview

- Optimization of memory usage and training time
- Transfer learning & Meta-learning
- TFLite / Pytorch model-lelve compatibility
- Easy to implement custom operators
- Supports Android, Tizen, Linux



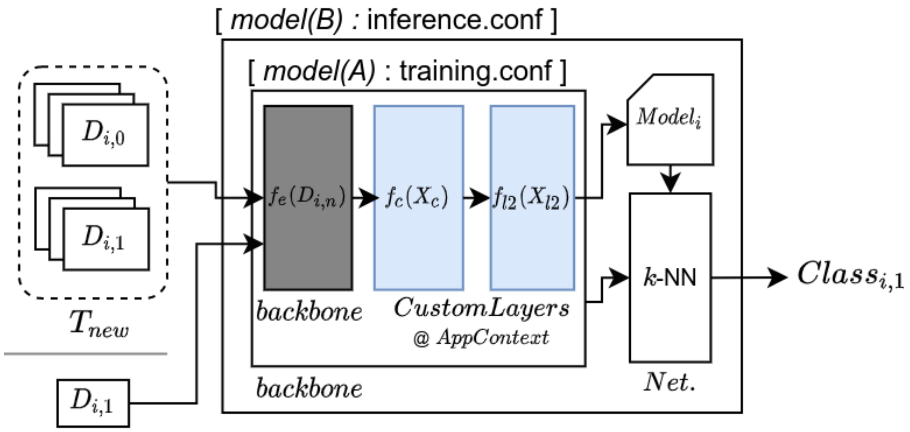
nnTrainer System Architecture



Peak Memory Consumption

- **PyTorch : 1.2 GiB**
- **TensorFlow : 1.02 GiB**
- **NNTrainer : 0.37 GiB**

Few-Shot Learning: SimpleShot



SimpleShot Implementation

ResNet50	UN		L2N		CL2N	
shots	Acc.	Std.	Acc.	Std.	Acc.	Std.
1	29.28	9.02	45.44	14.63	30.12	9.64
5	61.32	8.12	63.16	8.15	60.56	7.69
10	69.24	7.47	71.08	7.27	69.04	7.76
20	69.72	8.59	75.16	6.54	72.80	6.74

Conv4	UN		L2N		CL2N	
shots	Acc.	Std.	Acc.	Std.	Acc.	Std.
1	44.24	7.76	48.44	11.20	46.72	1.14
5	67.56	4.97	70.52	5.79	67.92	4.48
10	72.68	6.63	74.36	5.60	73.44	6.63
20	76.76	5.37	77.88	5.13	76.80	5.63

SimpleShot Inference Results

Example: HandMoji

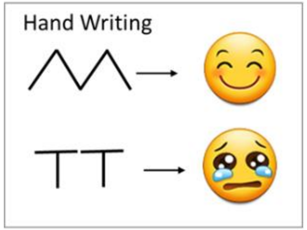
Samsung Research

```
[Model]
Type = NeuralNetwork
Learning_rate = 0.0001
Decay_rate = 0.96
...
[mobilenetv2]
Backbone = mobilenetv2.tflite
Input_Shape = 224:224:3
[flatten]
Type = flatten
input_layers = mobilenetv2
[outputlayer]
Type = fully_connected
input_layers = flatten
...
unit = 2
```

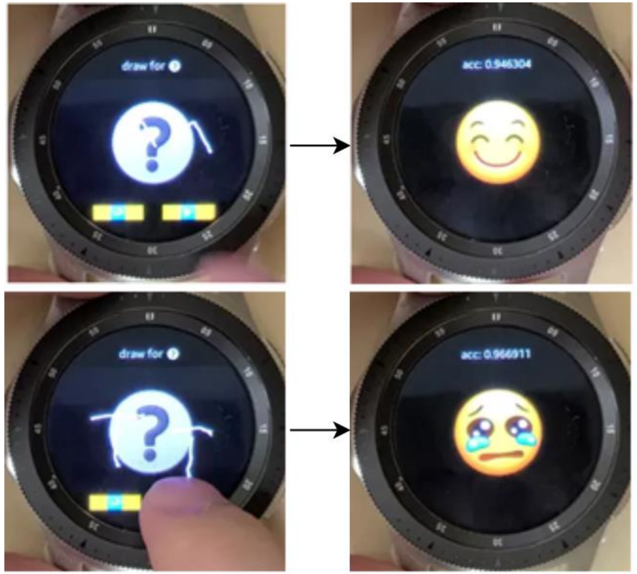
Model Conf. File

5 Times per emoji

Collect User data



Train



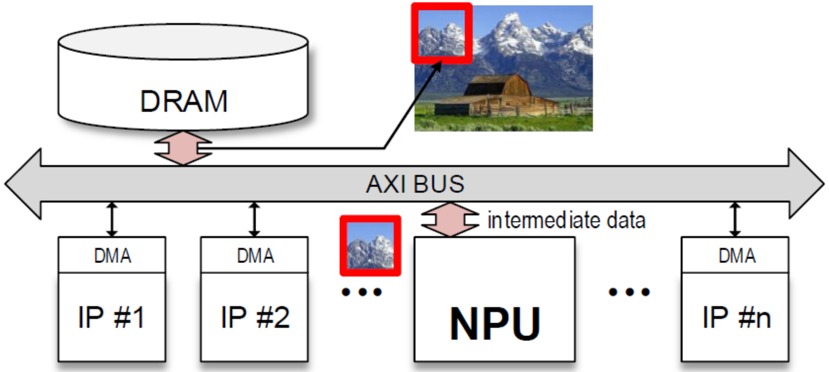
Inference

SNP

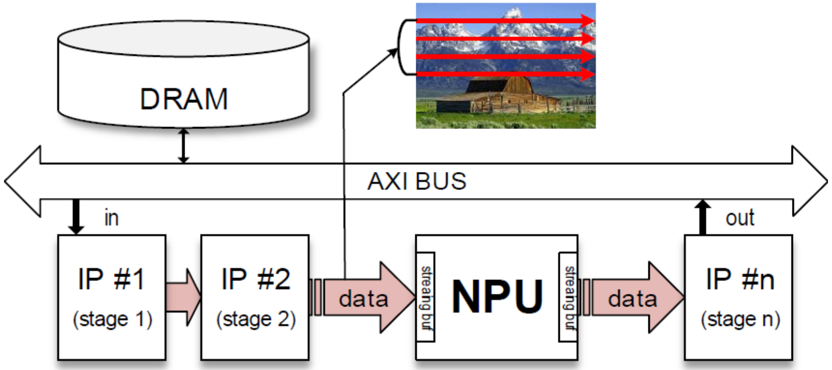
Streaming Line Processing Architecture with a Winograd Convolution Array for 4K 60fps Super-Resolution Applications

Work-in-Progress

Stream Processing for TVs



(a)

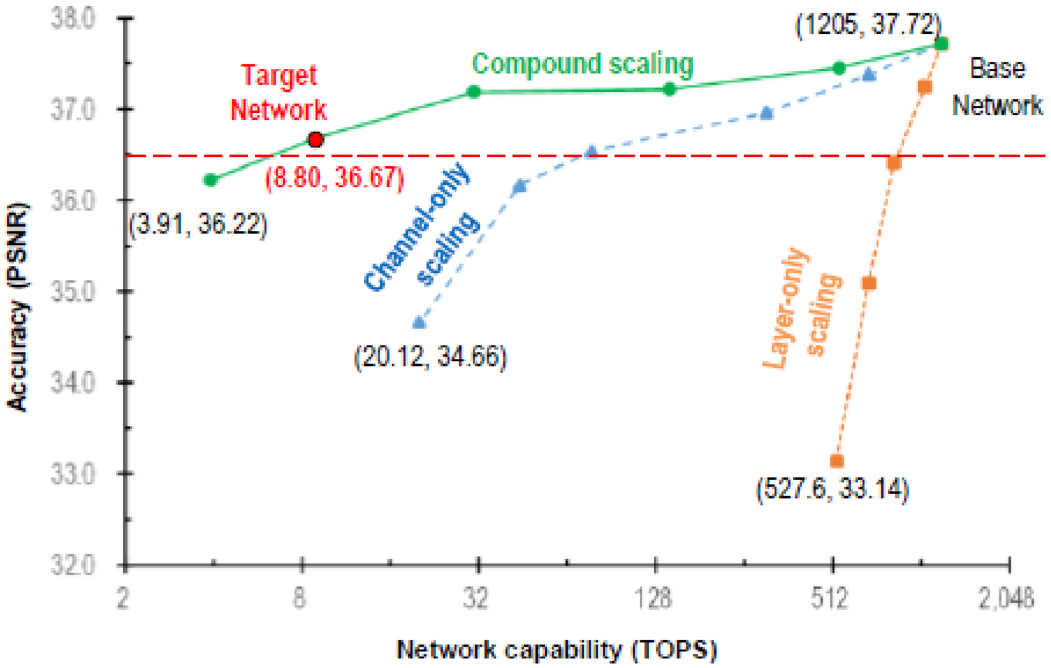


(b)

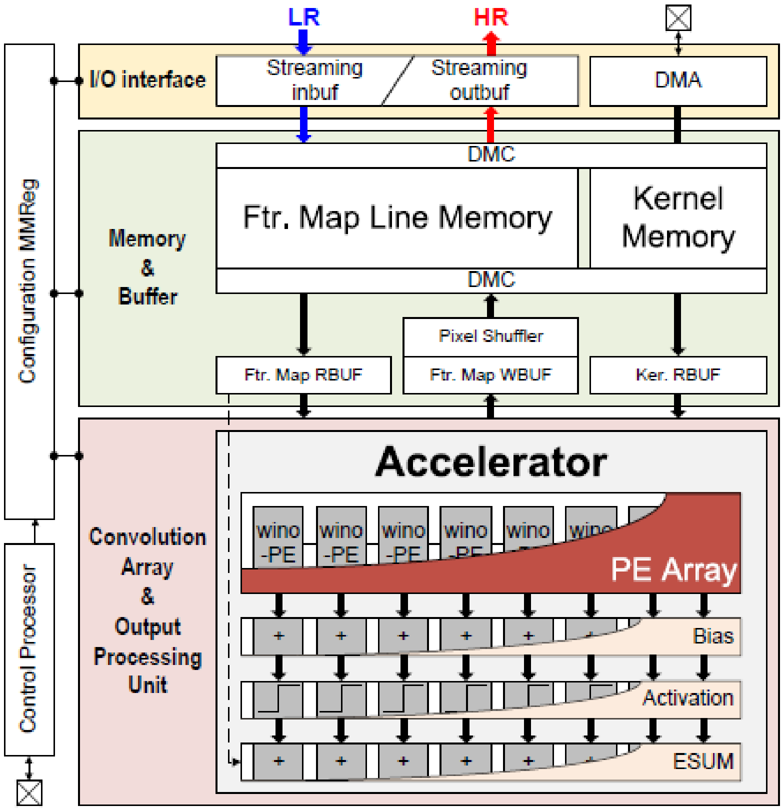
Non-Streaming Environment

Streaming Environment

Super-Resolution Neural Network Scaling



Accelerator Architecture

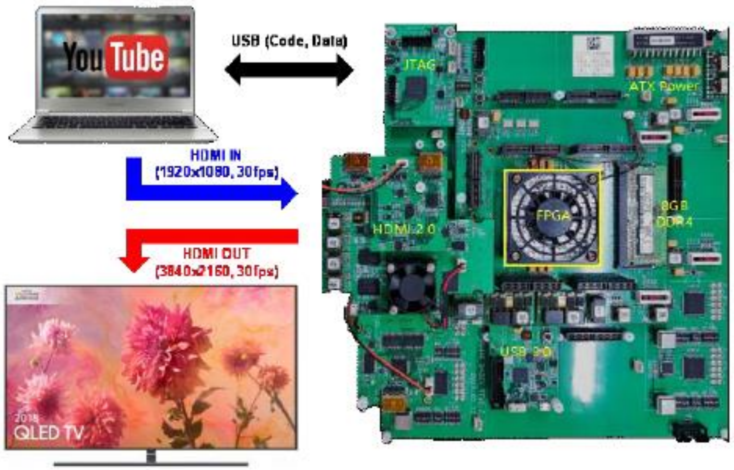


Architecture Overview

	Blocks	MG/C	Percentage
PE Array	Multiplier	2.78	8.2%
	Accumulator	0.38	1.1%
	Transform Matmul	0.31	0.9%
	RF	5.03	14.9%
	B^{T}d BUF	0.42	1.2%
	Glue Logic	1.34	4.0%
	Sub Total	**10.25**	**30.4%**
Memory	Line memory	18.82	55.8%
	Kernel memory	0.50	1.5%
	Sub Total	**19.32**	**57.3%**
ETC	OPU	1.50	4.4%
	Control Processor	0.89	2.6%
	FRBUF/FWBUF	0.77	2.3%
	DMAC/DMC	1.00	3.0%
	Sub Total	**4.16**	**12.3%**
Total		**33.73**	**100.0%**

Area Breakdown

Implementation & Evaluation



FPGA Demonstration

Publication	Kim [21]	Kim [22]	Huang [23]	This work
FPGA device or process technology	0.13μm	Xilinx XCKU040	40nm	14nm
Operating frequency	220 MHz	150 MHz	250 MHz	1 GHz
Supported scale	x 2	x 2	x 2, 4	x 2, 3, 4
Max. throughput	4K 60fps	4K 60fps	4K 30fps	4K 60fps
TOPS	-	0.764 (estimated)	41	11.52
Area	-	-	55.23 mm^2	5.95 mm^2
Area efficiency ($TOPS/mm^2$)	-	-	0.742	1.94
Power consumption	-	4.791 W	5.76 – 7.46 W	338 mW
Power efficiency (TOPS/W)	-	0.16	5.50 - 7.12	34.11
Memory size	92 KB	392 KB	2.8 MB	2.09 MB
Precision	-	10b (w), 14b (a)	8b (dynamic fixed)	8b (dynamic fixed)
PSNR (Set14)	31.63 dB	32.47 dB	33.33 – 33.70 dB	33.58 dB

Comparisons of CNN Hardware Accelerators

AI-SW-HW Co-Design

Efficient Neural Network: Voice

13x smaller neural network can show the similar performance in Speech Recognition

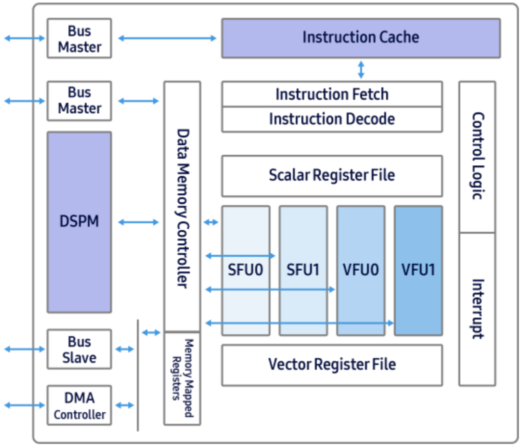
91.6% Accuracy @ 530 MB → 91.1 % Accuracy @ 38MB

Bits	Hyper LRA	Korean			English		
		WER	xRT	Size	WER	xRT	Size
32	no	9.37	4.89	530.56	9.03	4.32	530.50
32	yes	9.85	0.99	140.18	8.91	1.15	153.98
32	+MWER	9.60	1.26	140.18	8.64	1.48	153.98
8	no	9.64	1.18	132.88	9.07	0.94	132.87
8	yes	10.21	0.33	35.34	9.24	0.38	38.77
8	+MWER	9.80	0.35	35.34	8.88	0.44	38.77

[Reference] Attention based on-device streaming speech recognition with large speech corpus (Interspeech 2019)

Acceleration of Neural Network: Specialized H/W

Voice Recognition NPU: about 4x less power consumption than CPU



ASR Accelerator Architecture

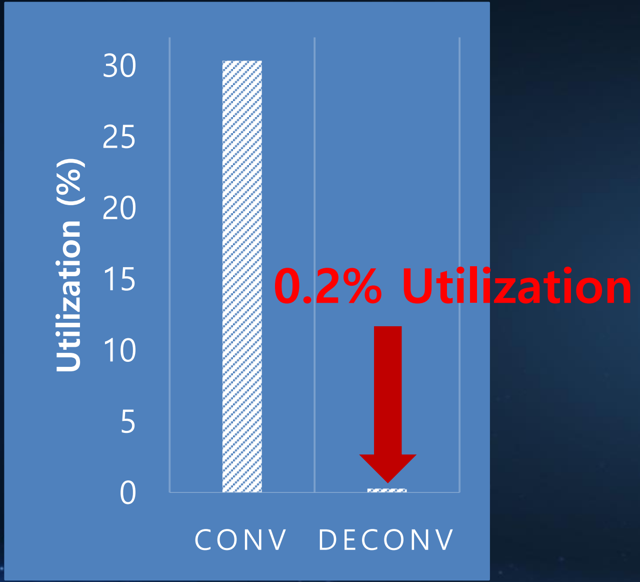
	CPU	ASR Acceleration (NPU-based)
Power Consumption	982mW	**276mW**

* Measured under xRT(real-time factor) <1

Performance Comparison

On-device AI Deployment

Real FLOPS matter!



Results from a commercial NPU IP

- Conv : 116.5 MACs/clk (30% Utul.)
- Deconv: 1.05 MACs/clk (0.2 % Util.)

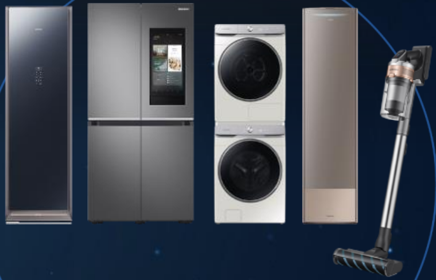
Exynos 2100 : 26 TOPS (Peak)
Snapdragon 888 : 26 TOPS (Peak)

Big challenge: How to exploit the "peak" FLOPS into "real" FLOPS ???

On-device AI will be embedded in various devices

Mobile
Communications

Visual Display



Digital Appliances

Big challenge: How to provide the “same” AI experience on variety of devices ???

Samsung Research

Thank you