NeuroSim: A Benchmark Framework of Compute-in-Memory Hardware Accelerators from Devices/Circuits to Architectures/Algorithms

Prof. Shimeng Yu

School of Electrical and Computer Engineering,

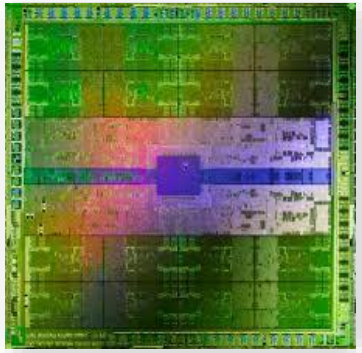
Georgia Institute of Technology

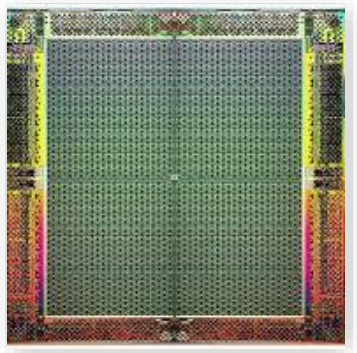
Email: shimeng.yu@ece.gatech.edu

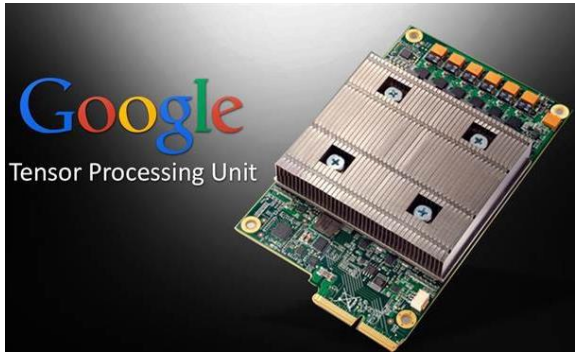
Web: https://shimeng.ece.gatech.edu/

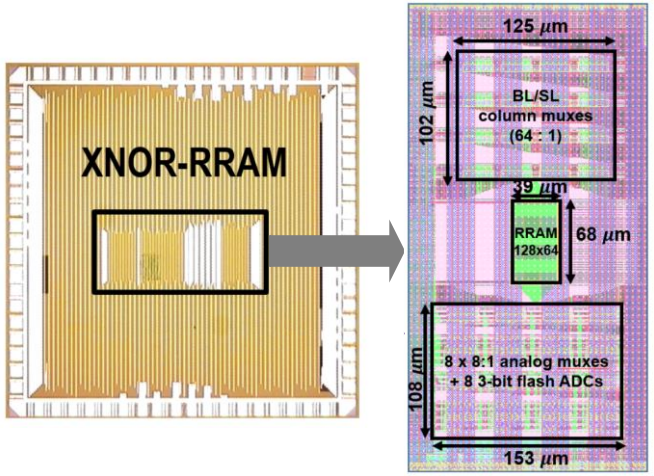
Hardware Accelerators for AI

- GPU still dominates the training in cloud, FPGA is good for inference for fast prototyping
- TPU (or similar digital ASIC) is ramping up in cloud as well as edge





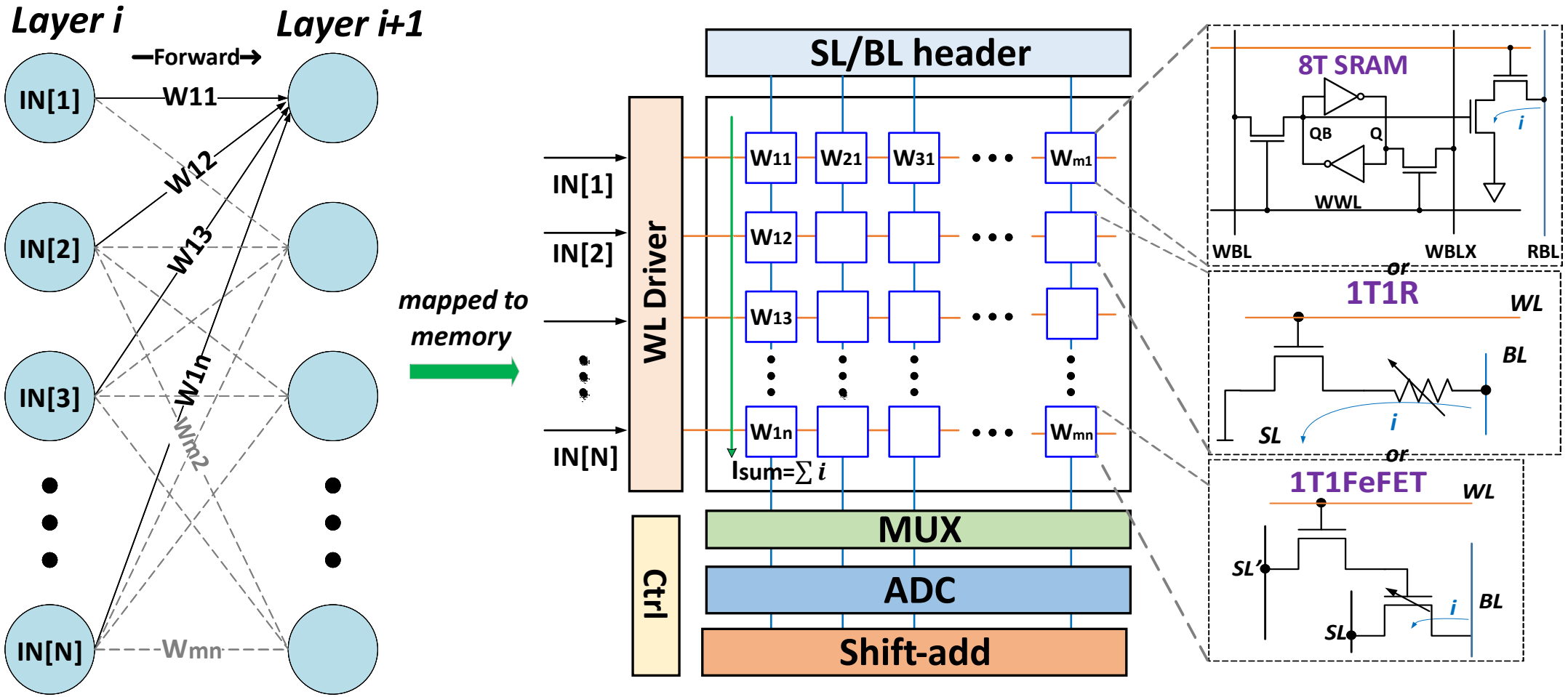




GPU	FPGA	TPU	Compute-in-memory (CIM)
Conventional computing platforms		Digital CMOS ASICs	Analog CMOS (or eNVMs)
~ 0.1 TOPS/W		~ 1-10 TOPS/W	~ 10-100 TOPS/W
Floating-point		Fixed-point	Low-precision→ accuracy?

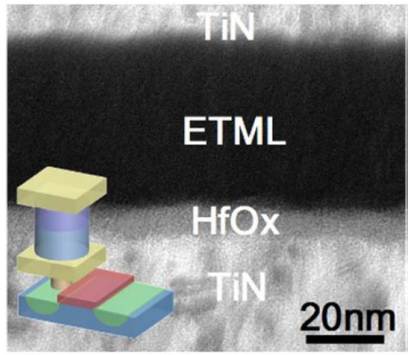
- To further improve energy efficiency (TOPS/W), analog CIM (possibly with eNVMs) is promising especially in the edge inference where the model is pre-trained.
- CIM chip could also support incremental learning with continuous (possibly unlabeled) new data when deployed to the field.

CIM Basics: Mixed-Signal Compute

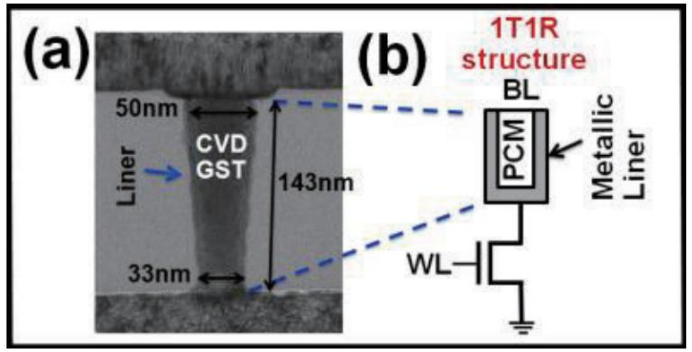


Needs for Early Design Exploration

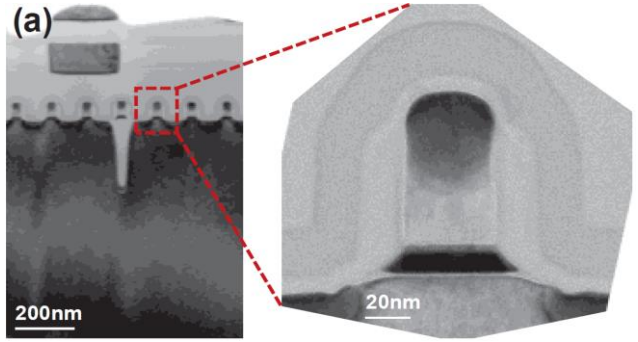
Versatile candidates as "analog" synaptic devices



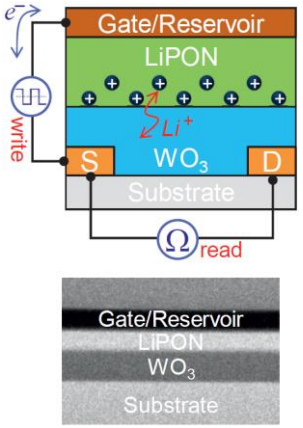
RRAM VLSI'18



PCM VLSI'19

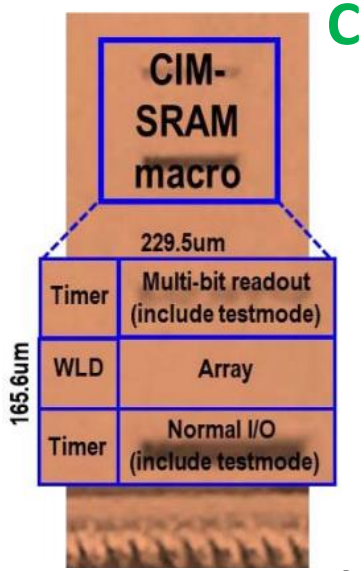


FeFET IEDM'18

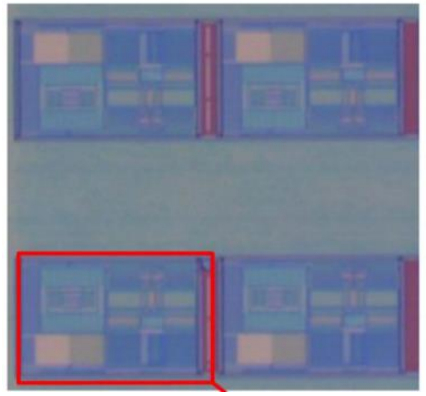


ECRAM IEDM'18

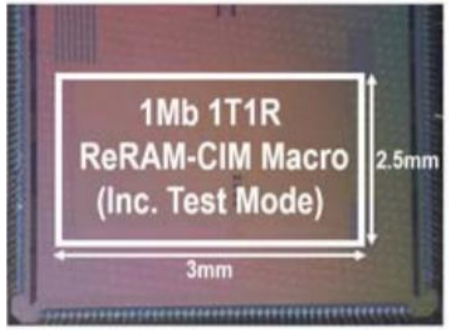
CIM macro demonstration



55nm SRAM ISSCC'19



7nm SRAM ISSCC'20



55nm RRAM ISSCC'19

22nm RRAM ISSCC'19

Critical Needs: Evaluate system-level multi-macro performance with new synaptic devices

NeuroSim: Open-Source Simulator for CIM

Why GitHub? Team Enterprise Explore Marketplace Pricing Search / Sign in Sign up

Overview | Repositories 10 | Projects | Packages

https://github.com/neurosim

➢ Pre-RTL simulator for early-stage research

NeuroSim

neurosim

Follow

Researchers from Prof. Shimeng Yu's group at Georgia Tech

80 followers · 0 following · 1

Pinned

MLP_NeuroSim_V3.0

Benchmark framework of synaptic device technologies for a simple neural network

C++ 65 46

DNN_NeuroSim_V2.1

Benchmark framework of compute-in-memory based accelerators for deep neural network (on-chip training chip focused)

C++ 15 7

DNN_NeuroSim_V1.3

Benchmark framework of compute-in-memory based accelerators for deep neural network (inference engine focused)

C++ 11 5

3D_NeuroSim_V1.0

Benchmark framework of 3D integrated CIM accelerators for popular DNN inference, support both monolithic and heterogeneous 3D integration

C++ 8 2

167 contributions in the last year

NeuroSim History

The first appearance:

P.-Y. Chen, **S. Yu**, "NeuroSim: A circuit-level benchmark simulator for neuro-inspired architectures," Workshop on Hardware and Algorithms for Learning On-a-chip **(HALO) 2015**, Austin, TX.

MLP+NeuroSim (2-layer MLP)

P.-Y. Chen, IEDM'17,
P.-Y. Chen, TCAD'18

Interface w/ PyTorch

DNN+NeuroSim V1 (Inference)

X. Peng, IEDM'19

Interface w/ PyTorch

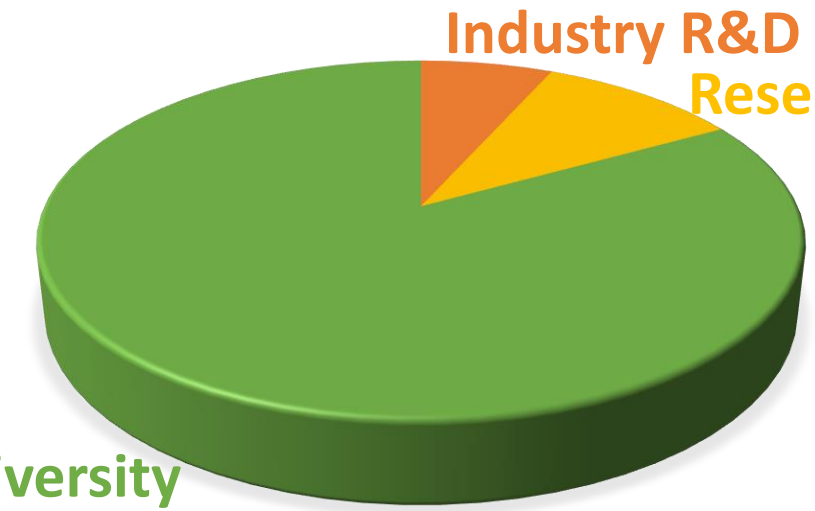
DNN+NeuroSim V2 (Training)

X. Peng, TCAD'20

Contributors:
Pai-Yu Chen, Xiaochen Peng
Shanshi Huang, Yandong Luo
Anni Lu, etc...

Impact of NeuroSim Family

CURRENT GLOBAL USERS > 300



Resulted in publications 100+

DNN+NeuroSim Methodologies (Python & C++)

- End-to-end frameworks for: [1*]inference engine [1], [2*]online training system [2], [3*]3D-CIM design [3]
- Input parameters related to: ***device → circuits → architecture hierarchy and data-flow → algorithm***
- Software-hardware co-simulation reports: ***inference/training accuracy, TOPS, TOPS/W and TOPS/mm²***

Algorithm accuracy estimation based on WAGE method [4]

- Hardware-aware quantization for weight, activation, gradient, error, as well as partial sum quantization based on ADC precision.
- Support various network models for CIFAR-10/-100 and ImageNet

Hardware metrics estimation based on analytic models that are calibrated with SPICE at module-level.

- Massive options (tech-node 130nm to 7nm, memory device type, R_{ON}, ADC option…)
- Analog modules (e.g. ADC) calibrated with Cadence custom simulation;
- Digital modules estimated with standard cell area and logic gate delay/dynamic power/leakage power;
- Interconnect modules (e.g. H-tree) estimated with parasitic RC delay and power;

[1] X. Peng, et al., IEDM'2019; [2] X. Peng, et al., TCAD'2020; [3] X. Peng, et al., IEDM'2020. [4] S. Wu, et al. ICLR'2018

Area Estimation

eNVM **SRAM**

1T1R or crossbar ?

1: Get array size

e.g. cell size=4F*4F, synaptic array size=128*128
→ Array area = (128*4F) * (128*4F)

2: Auto-define peripheries

According to the synaptic array size 128*128
→ Define switch matrix or decoder specs (e.g. 7-bit)
According to the wire loading cap
→ Define transmission gate (TG) size (avoid large IR drop)

Note

Assume $R_{TG}=R_{ON}*Drop_{tolerance}$ (default = 0.1)
Define TG size according to R_{TG}

3: Other user-defined peripheries

e.g. ADC (MLSA/SAR-ADC) User define: *precision, ADC-mode, ADC-mux-sharing*

Align area according to the synaptic array (w/ width or height)

Latency Estimation

Read

Write

1: RC delay (digital block)

Tech-file provides temperature-dependent I_{ON} & I_{OFF} → get R (based on transistor size)

Similarly, get C (loading cap can be wire cap)

1*: Cadence fitting function (analog ADC)

Column resistance R_{COL} is calculated based on real-trace (mapped conductance and input vector)

R_{COL} as input of latency fitting function

2: Consider operation scheme

Each block needs to consider number of operations (i.e. T=N*t)

Total latency needs to be "∑" or "MAX" according to the operation scheme (depends on working in sequence or in parallel)

Dynamic Energy Estimation

Read **Write**

1: CV^2 dynamic energy (array)

Loading wire in array:

$E \mathrel{+}= C_{WL} * V_{dd}^2 * \#WL_{Selected}$

$E \mathrel{+}= C_{BL} * V_{read}^2 * \#BL_{Selected}$

$E \mathrel{*}= \#op$

2: CV^2 dynamic energy (digital block)

Assume critical operation scheme, count number of transistors that need to be charged-up;

Sum-up CV^2 for all charged-up transistors;

Also times #op (number of operation).

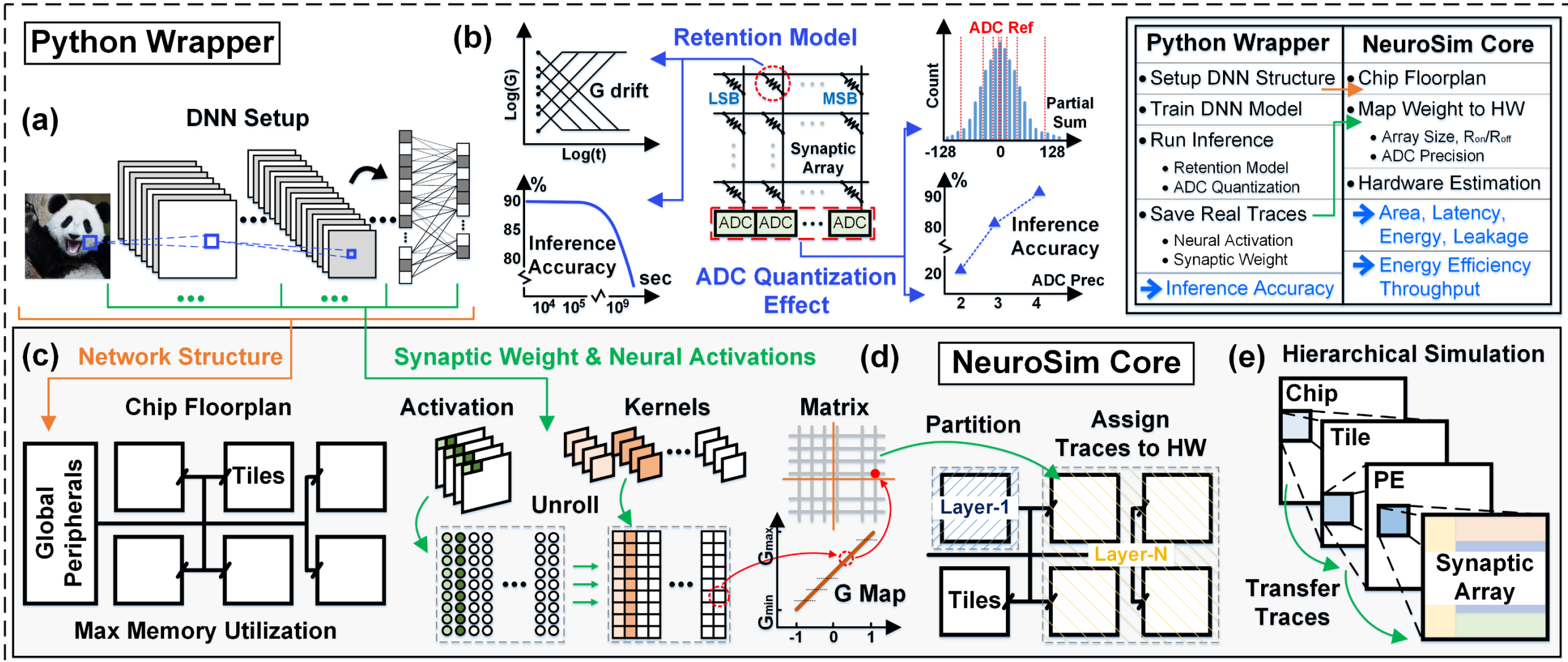
3: Cadence fitting function (analog block)

Column resistance R_{COL} is calculated based on real-trace (mapped conductance and input vector)

R_{COL} as input of power fitting function

Energy is calculated based on latency and power fitting function

DNN+NeuroSim V1 for Inference Engine



X. Peng, et al., IEDM'2019.

DNN+NeuroSim V1 for Inference Engine

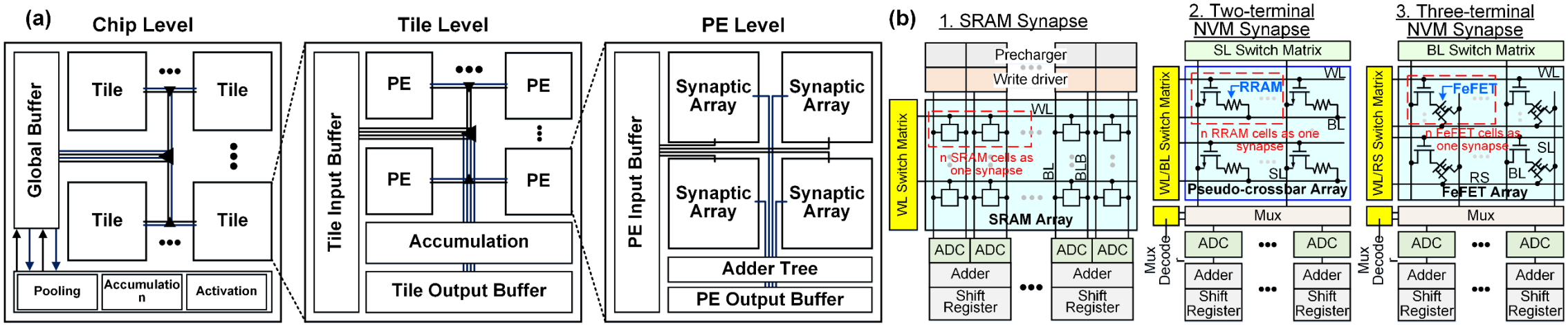


Fig.1 Architecture hierarchy of CIM accelerator (defined in DNN+NeuroSim V1.0).

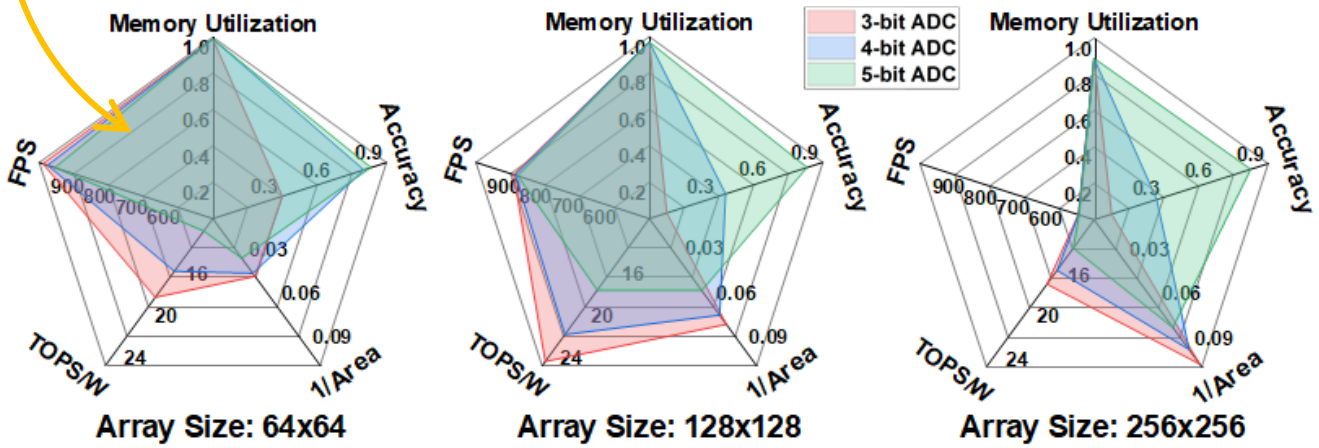


Fig. 2 System-level benchmark (VGG-8) with various design options (4-bit/cell).

Sweep array size & cell precision & ADC precision on CIFAR-10 accuracy

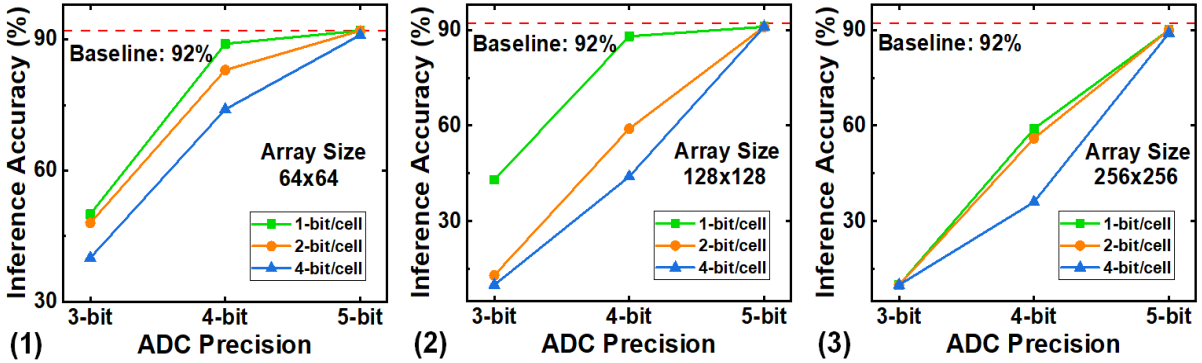
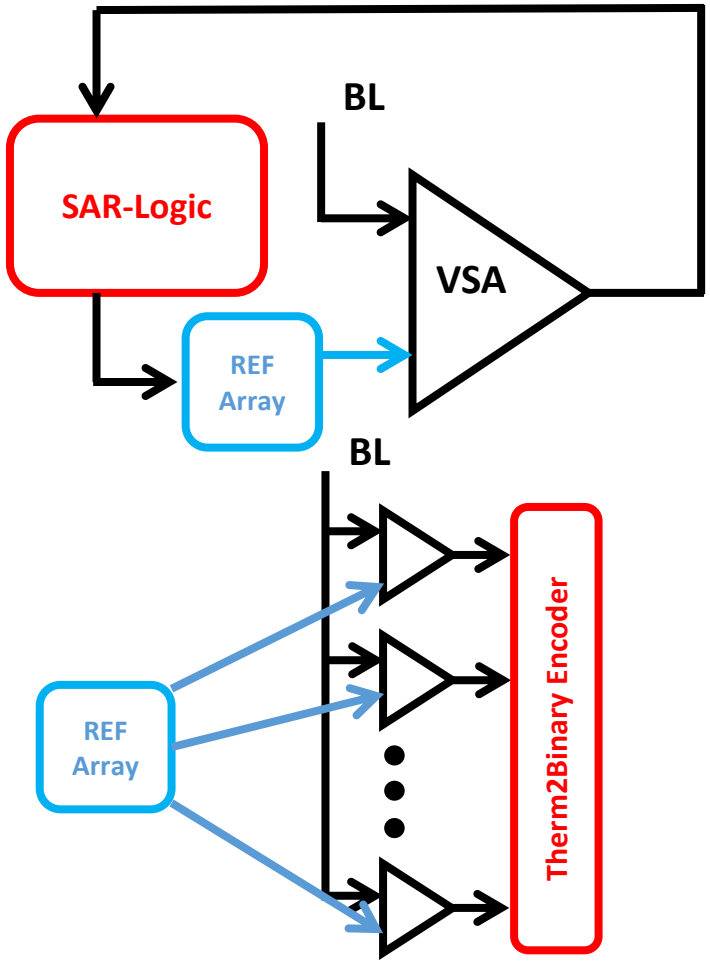
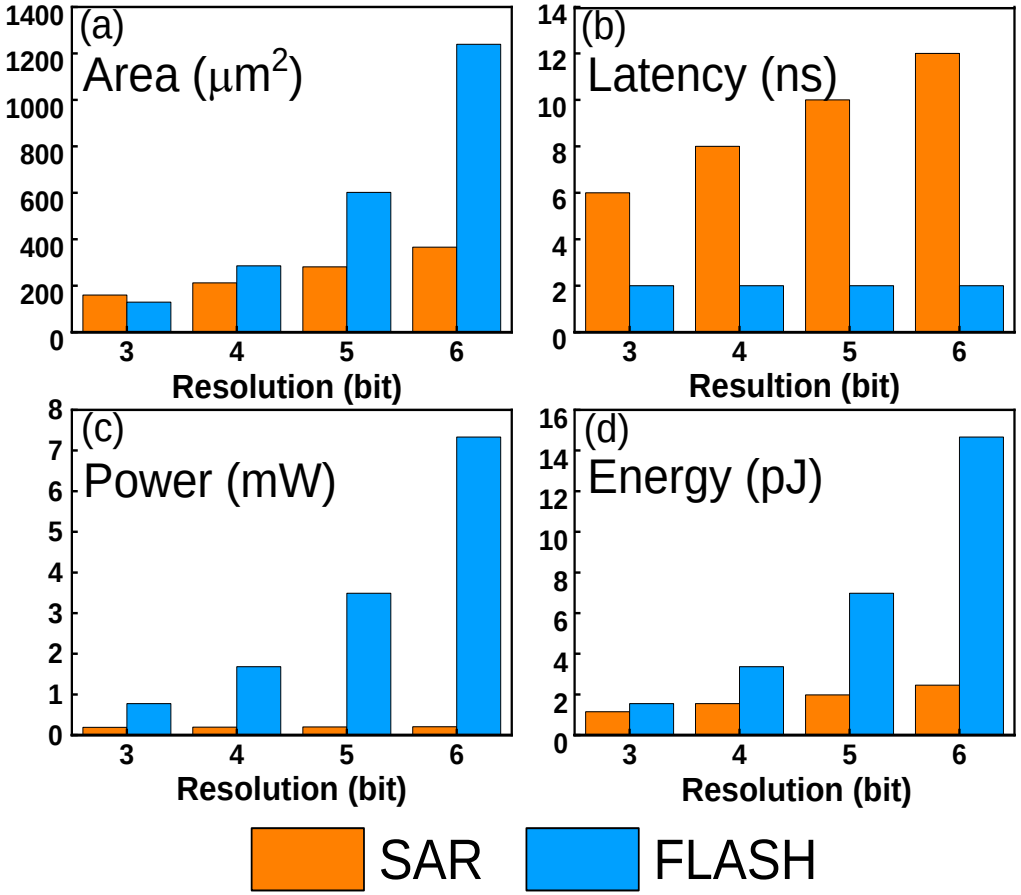


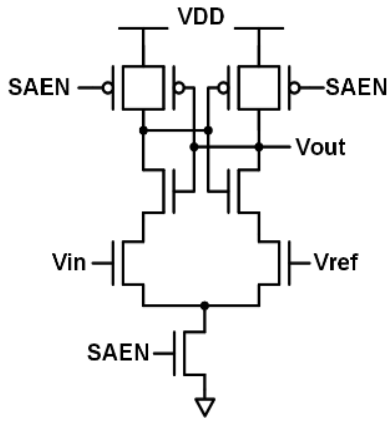
Fig.3 Inference accuracy with different ADC quantization: sweep subarray size and cell precision.

ADC Design Considerations

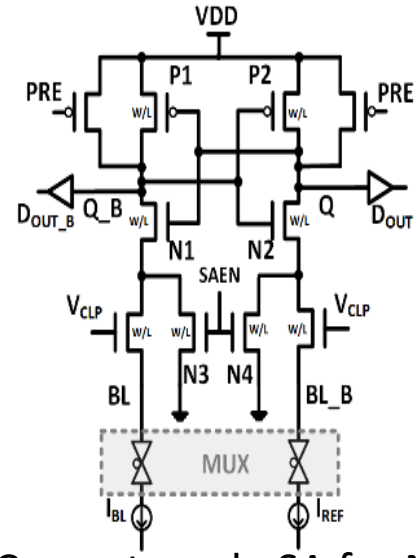


TSMC 40nm PDK





Voltage mode SA
for Flash ADC and SAR ADC



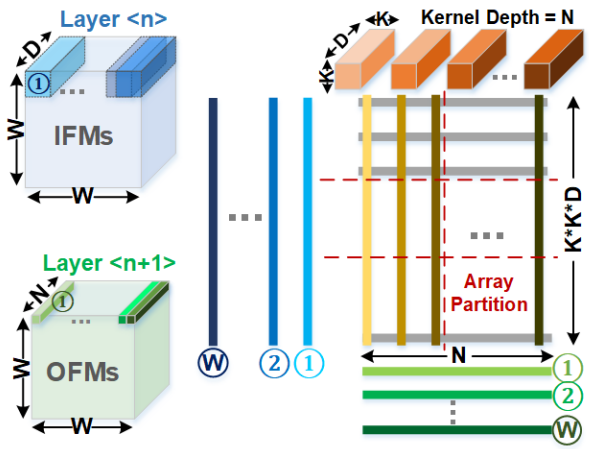
Current mode SA for MLSA

S. Yu. et al. CICC 2020

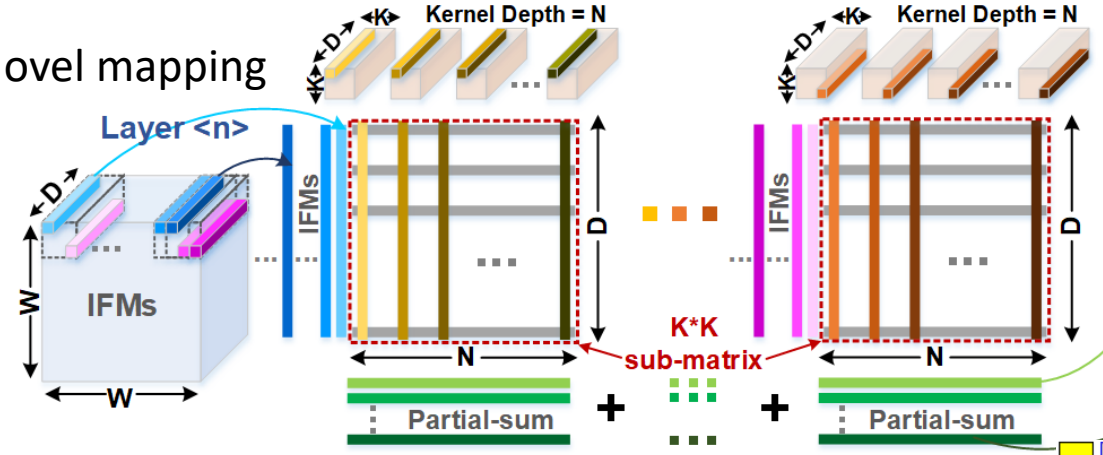
For partial sum precision larger than 4bit, SAR ADC provides better tradeoffs than Flash ADC

Weight Mapping and Duplication

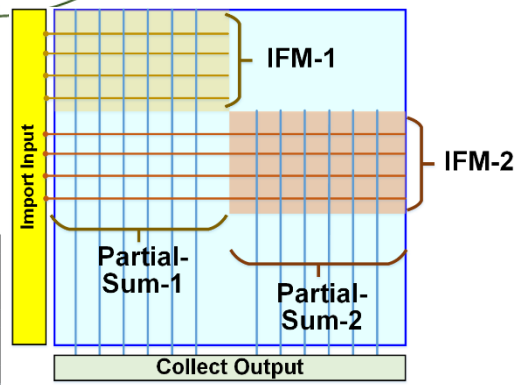
Conventional mapping



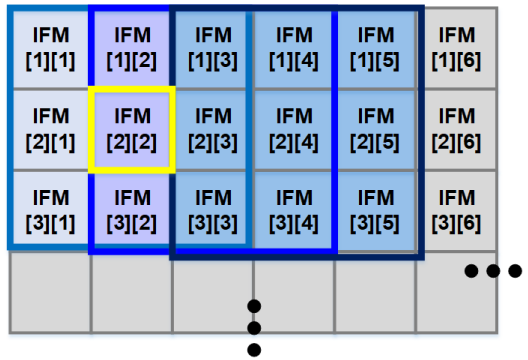
Novel mapping



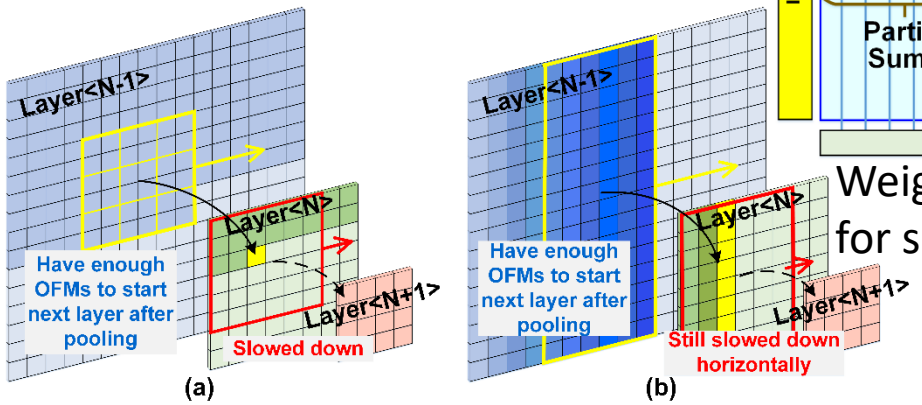
- IFM size = W*W*D, Kernel size = K*K*D*N, OFM size = W*W*N
- Number of sub-matrix = K*K



Input reuse in novel mapping



X. Peng TCAS-I 2020



Weight duplication for speed up

DNN+NeuroSim V1 Reveals Key Factors in CIM

Table 1 Benchmark CIM accelerators on 8-bit VGG-8, across versatile device technologies.

VGG-8 (8-bit activation; 8-bit weight) on CIFAR10, with Novel Weight Mapping and Dataflow							
Technology node (LSTP)	7nm	22nm					
Device	8T-SRAM	8T-SRAM	RRAM (Intel)	TaOx/HfOx (TsingHua)	GST PCM (IBM)	Si:HfO2 FeFET (GF)	ECRAM (IBM)
MLSA-ADC precision	4-bit	5-bit					
Cell Precision	1-bit	2-bit	4-bit				
Ron (Ω)	\	\	6k	100k	40k	240k	500M
On/Off Ratio	\	\	17	10	12.5	100	40
Inference Accuracy (%)					91%		
Area (mm^2)	13.61	59.05	60.78	33.26	33.39	32.69	33.26
Memory Utilization (%)	98.73%		96.86%	93.47%	93.47%	93.47%	93.47%
L-by-L Latency (ms)	0.63	0.76	0.79	0.61	0.61	0.61	1.09
L-by-L DynamicEnergy (uJ)	22.86	56.56	33.70	17.24	17.85	16.28	16.77
L-by-L Leakage power (mW)	1.47	1.11	0.17	0.09	0.09	0.09	0.09
Energy Efficiency (TOPS/W)	51.10	21.36	36.37	71.17	68.73	75.36	72.86
Throughput (FPS)	1589.24	1311.87	1260.87	1639.62	1637.63	1641.56	916.43

- Desired R_{ON}: 100k ~ 1M Ω
- eNVM must be "multilevel" to beat SRAM @ advanced tech-node
- For practical eNVM (e.g. Intel RRAM), can out-perform 7nm-SRAM at edge device (with low compute activity).

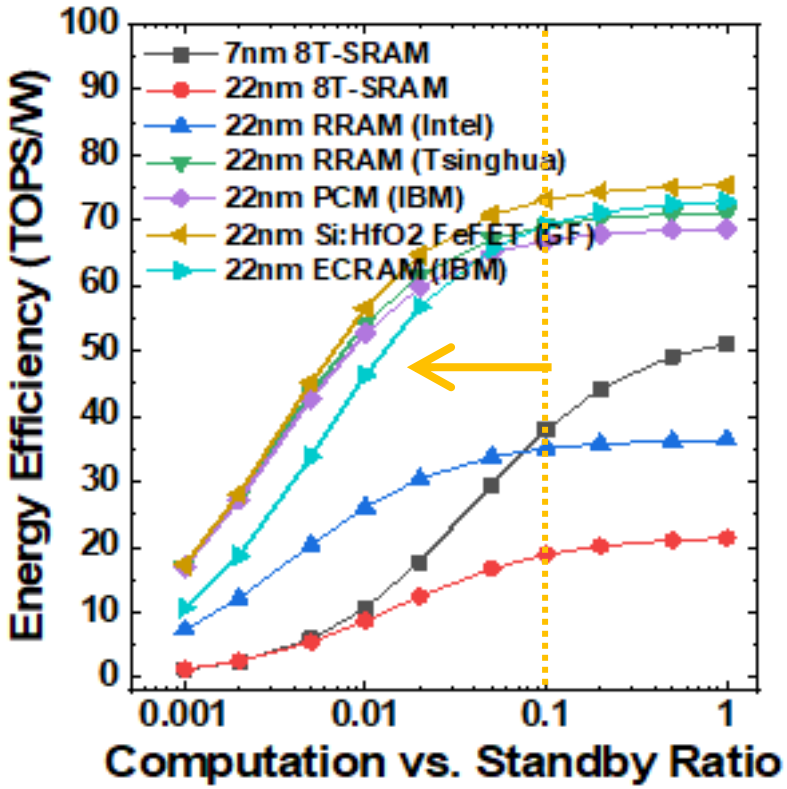
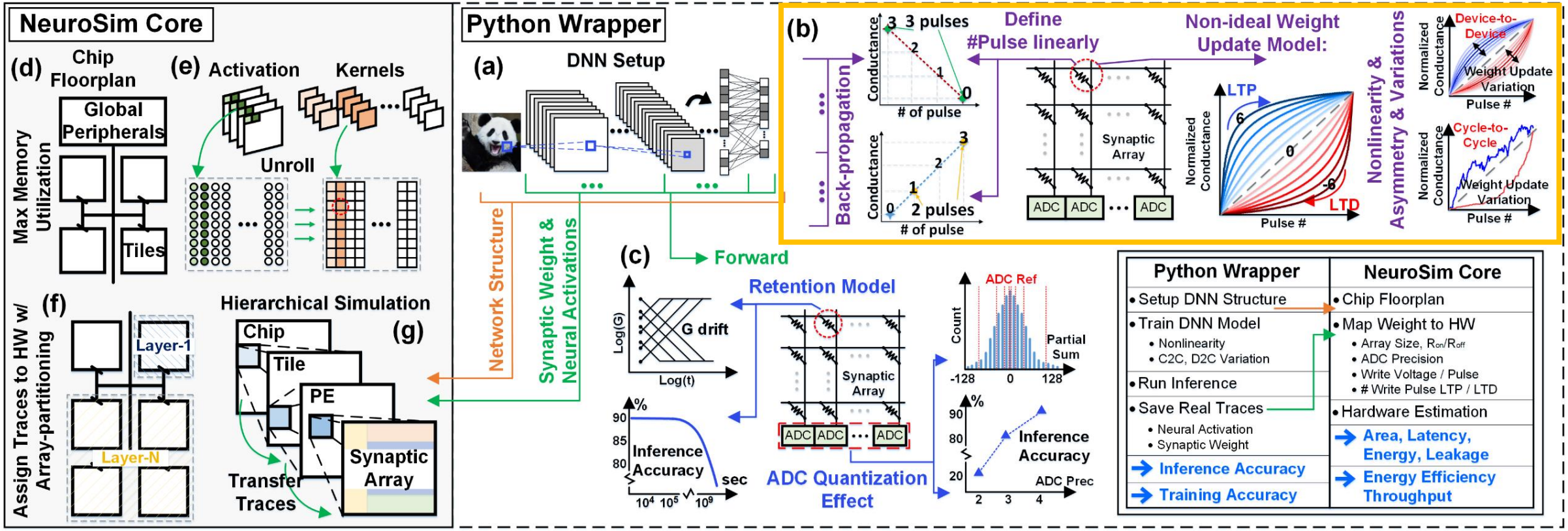


Fig. 1 Energy efficiency as a function of computation vs. standby ratio of CIM accelerators (across versatile device technologies).

DNN+NeuroSim V2 for Online Training

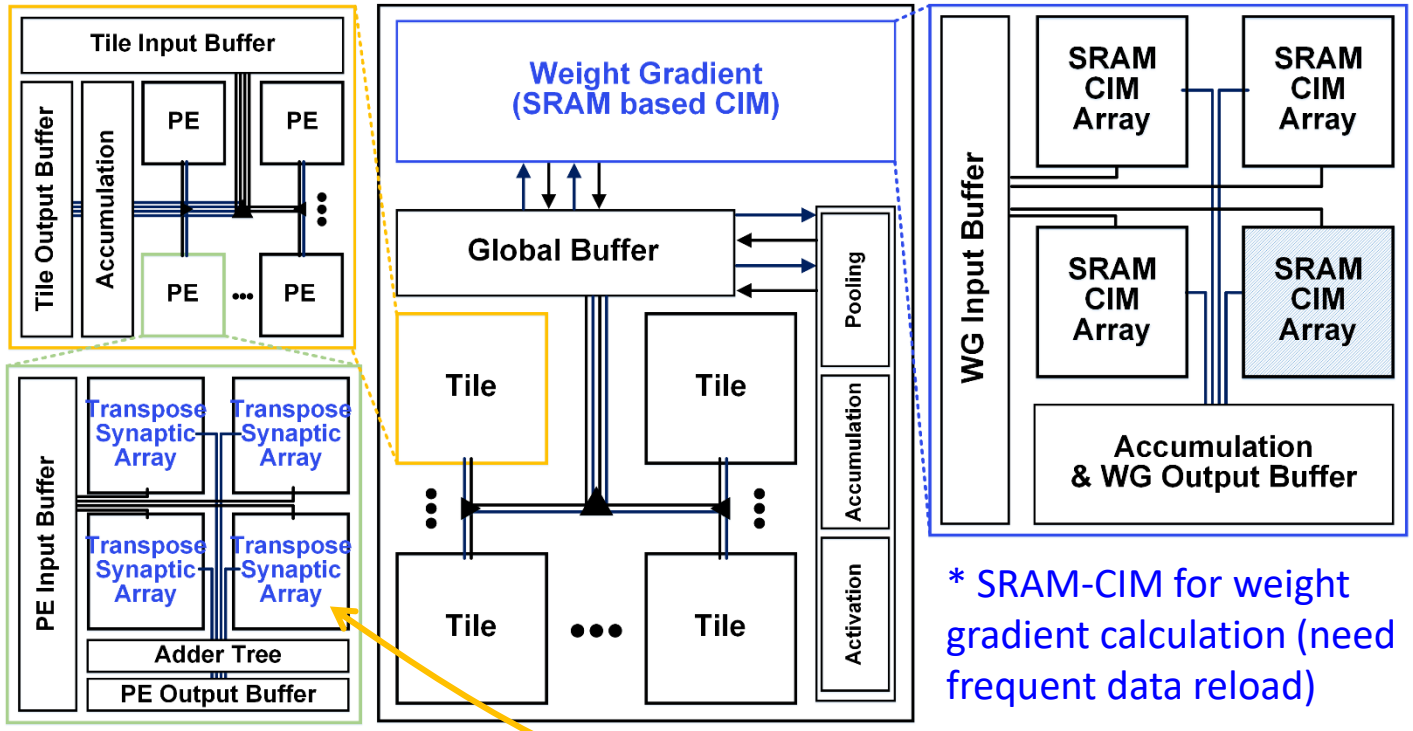
- Extends online training hardware and software analysis
- Non-ideal weight-update: asymmetry and non-linearity, device & cycle variation



X. Peng, et al., TCAD'2020.

DNN+NeuroSim V2 for Online Training

- Transposable synaptic array to support feed-forward & back-propagation (error calculation)
- Add weight gradient calculation (by SRAM-CIM), need frequent data reload (from DRAM)



* SRAM-CIM for weight gradient calculation (need frequent data reload)

Fig. 1 CIM architecture supports online training.

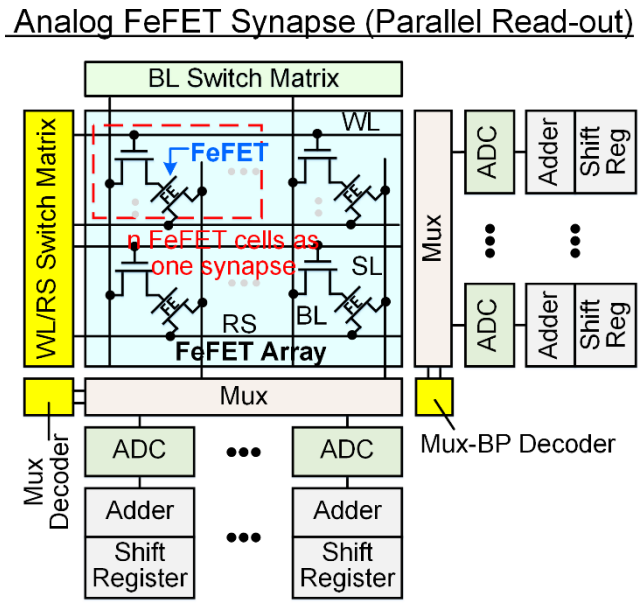
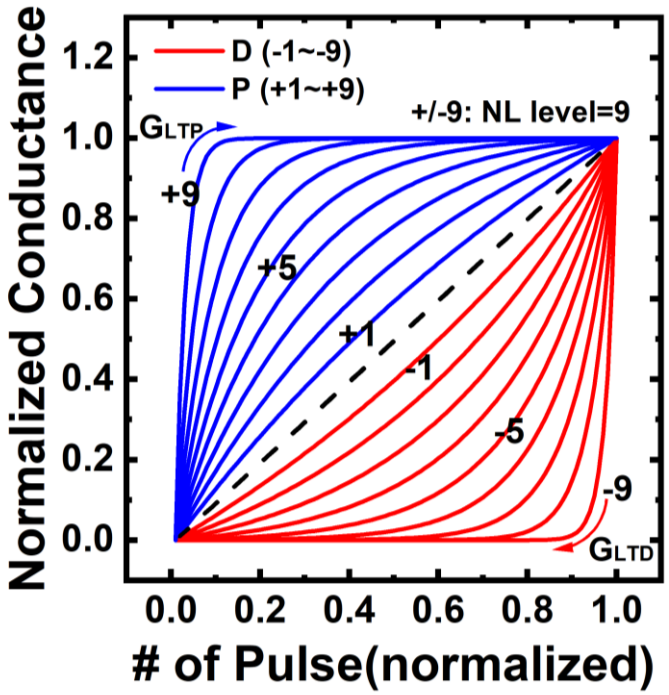
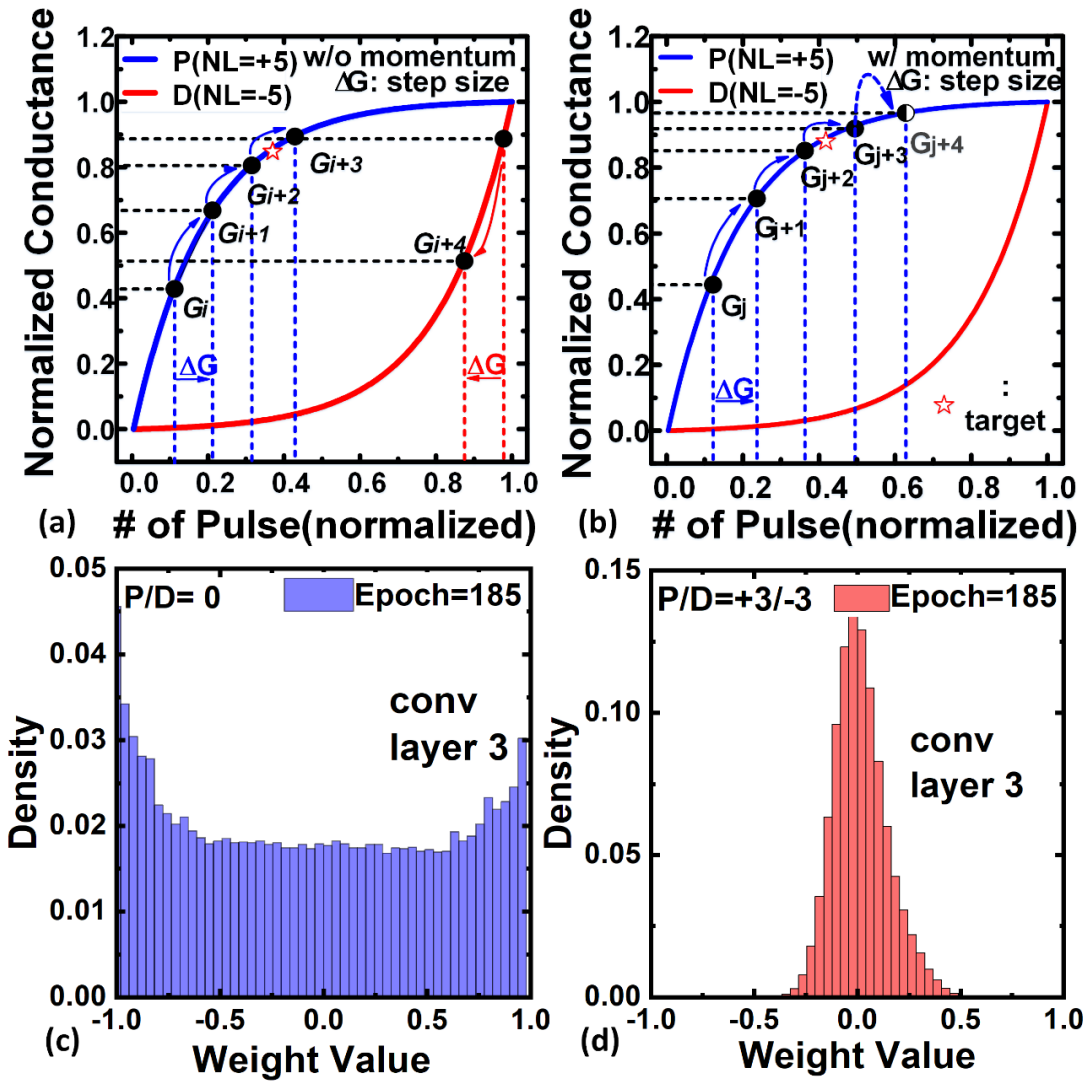


Fig. 2 Transposable synaptic array support both of feed-forward and back-propagation (take example of 3-terminal eNVM).

Asymmetry and Nonlinearity in Analog Synapse





- Asymmetry makes devices statistically easier to converge to the middle range of the conductance than approaching Gmax or Gmin.
- Frequent sign flipping causes a large unwanted conductance change towards the middle range.

Online Training: Nonlinearity & Asymmetry

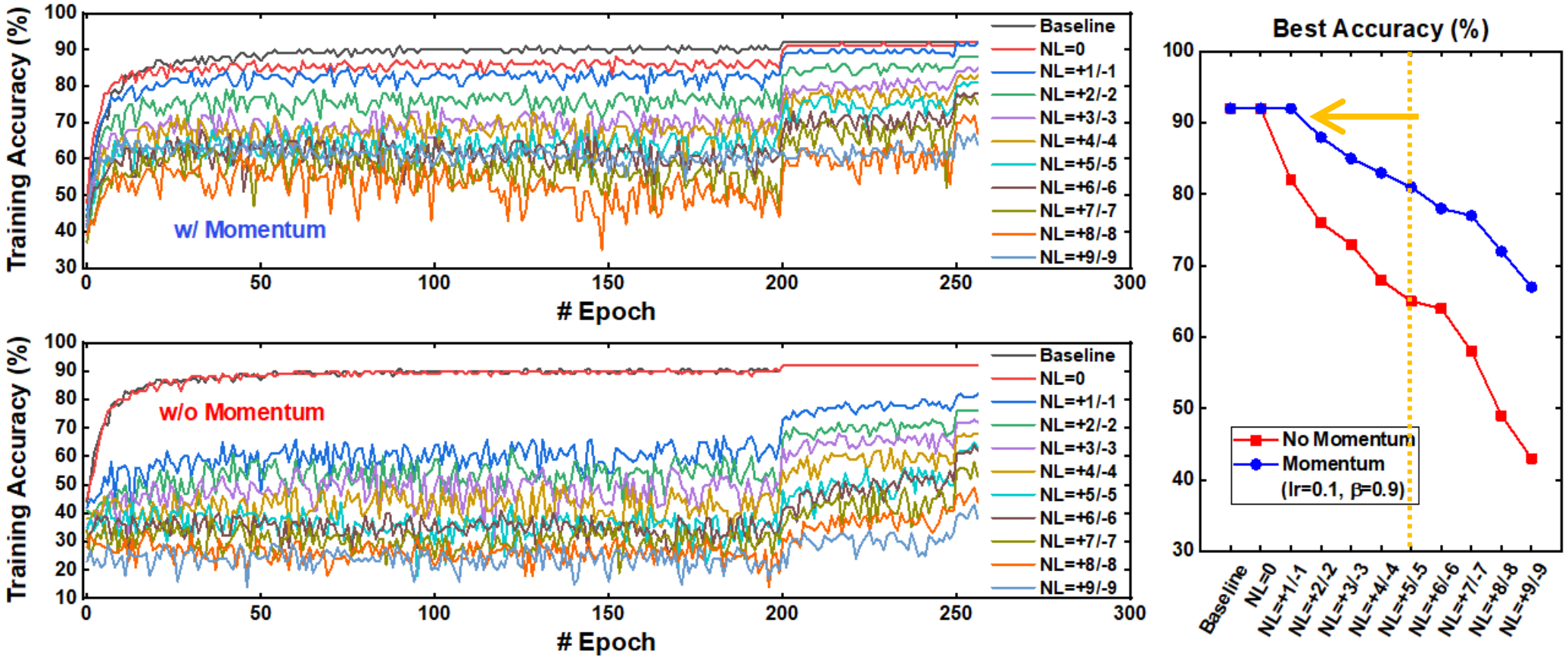


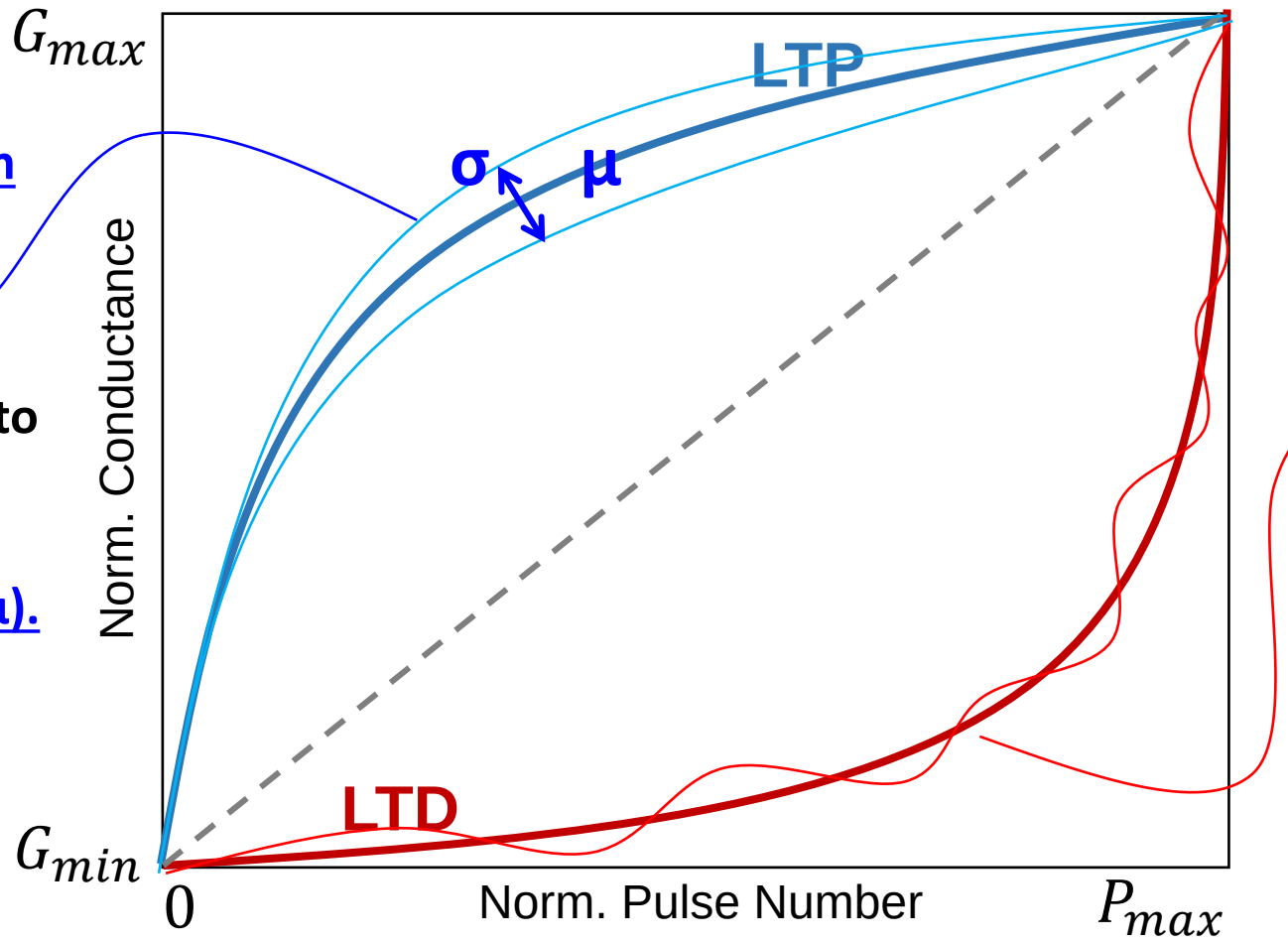
Fig. 1 Analysis of nonlinearity and asymmetry (8-bit VGG-8 for CIFAR-10), w/ and w/o momentum optimization. Results show at +5/-5, the accuracy is still >80% (w/ momentum optimization).

Mommentum optimization: $\Delta W(t) = \beta \Delta W(t-1) + (1-\beta) \cdot (-\frac{\partial L}{\partial W})$

Device & Cycle Variation Behavior Model

Device-to-device variation introduce different nonlinearity (NL).

Randomly generates NLs to different synapse with a **standard deviation (σ)** respect to the **mean NL (μ).**



Cycle-to-cycle variation introduce variation in conductance change at every weight-update.

Express C2C variation **standard deviation (σ)** in terms of percentage of entire weight range.

Online Training: Device & Cycle Variation

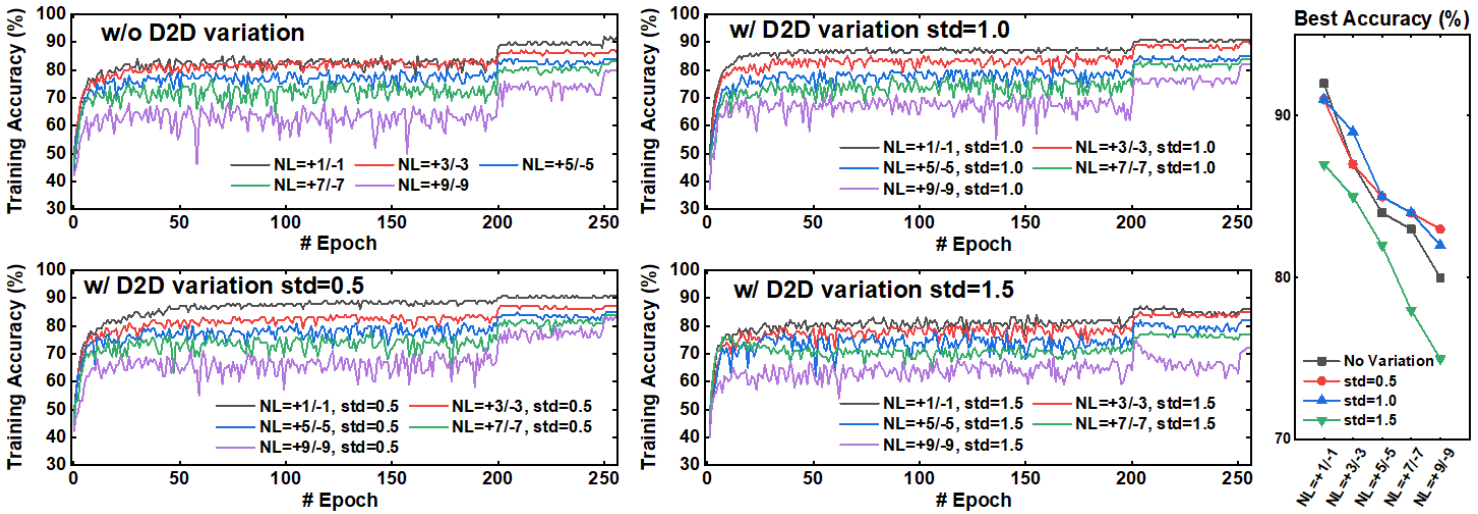
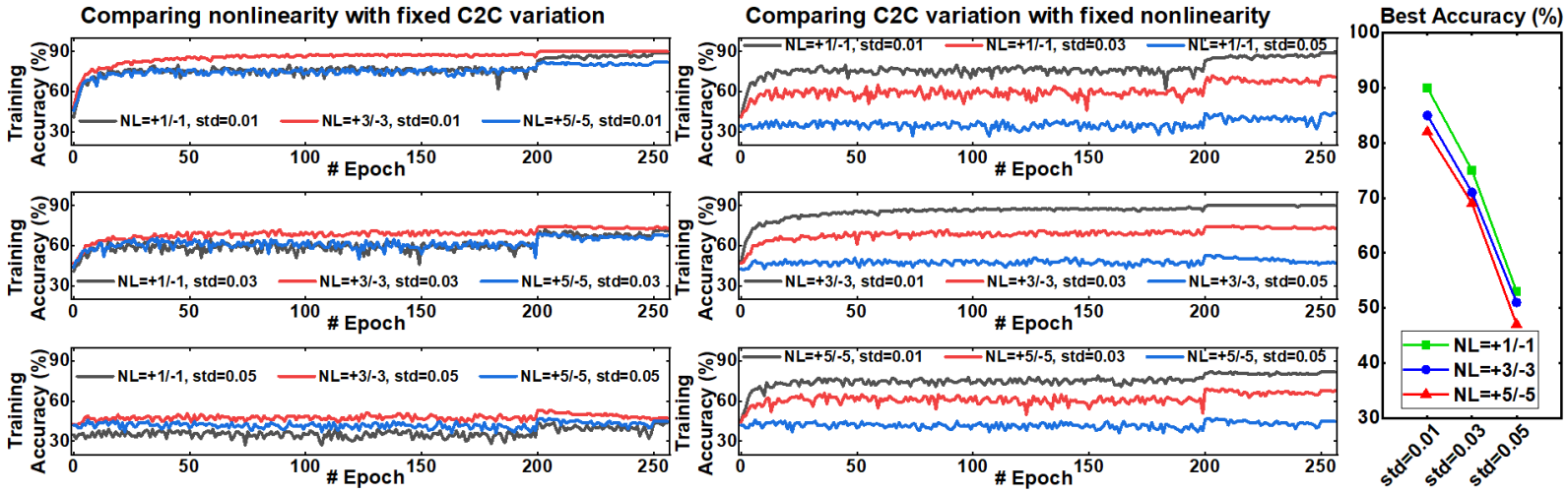
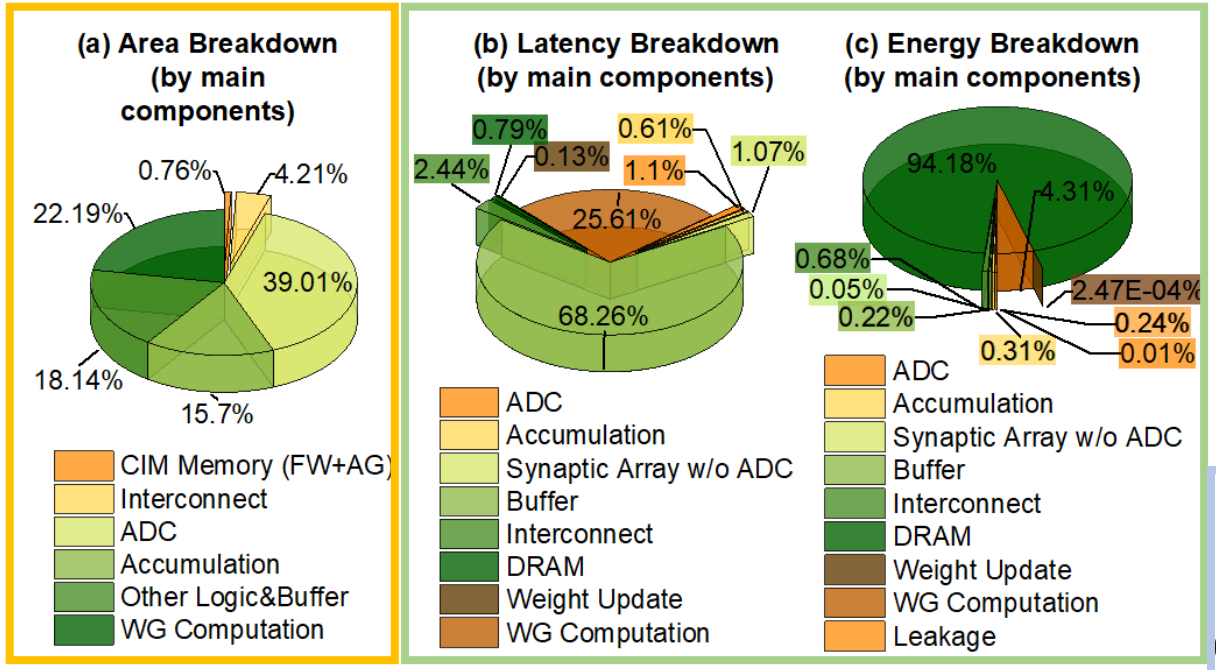


Fig. 1 Analysis of device-to-device variation under different non-linearity, w/ momentum optimization.
Behavior model randomly generates NLs to different synapse with a standard deviation (σ) respect to the mean NL (μ).

Fig. 2 Analysis of device-to-device variation under different non-linearity, w/ momentum optimization.
Behavior model generate variation in G change at every programming pulse; introduce standard deviation (σ) in terms of percentage of entire weight range.



Online Training: System Performance



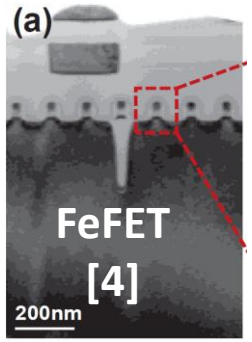
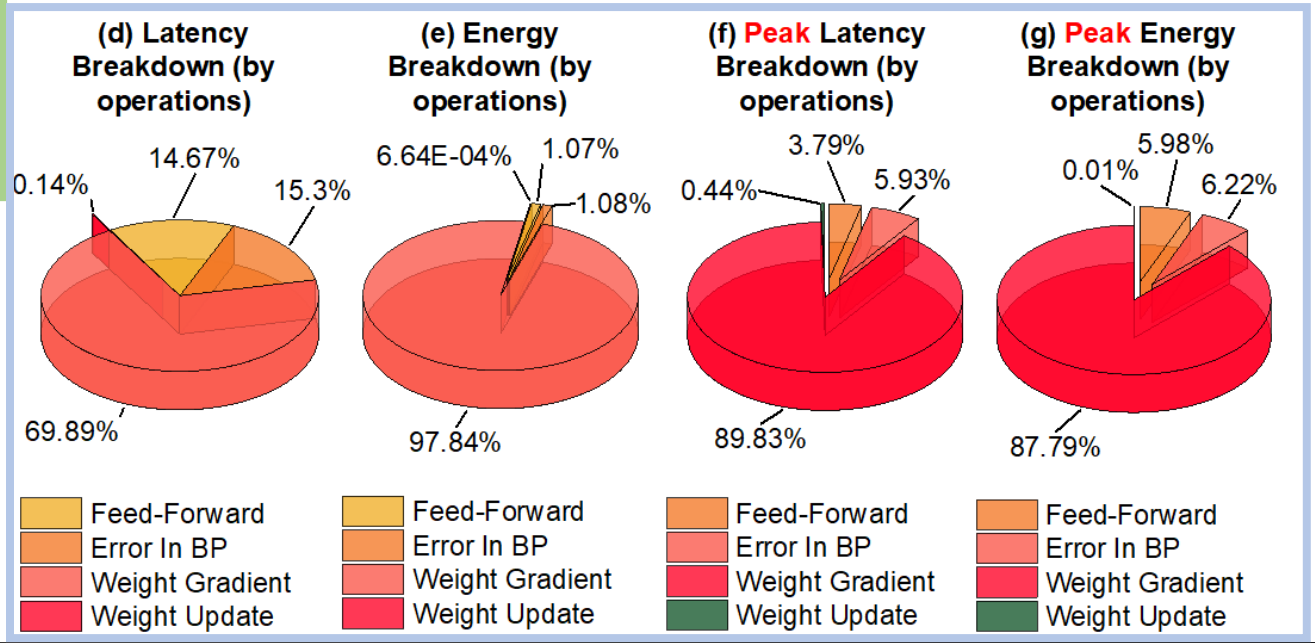


Fig. 16 The data shows the 100th epoch of FeFET-based CIM online training accelerator. (a) area breakdown by main components; (b) latency and (c) energy breakdown by main components; (d) latency and (e) energy breakdown by operations; (f) peak latency and (g) peak energy breakdown by operations.

- *ADC (6-bit) & weight gradient units (SRAM-based)* dominant in total area
- *Buffer latency & DRAM energy* is the bottleneck of performance
- Weight gradient computation is the bottleneck in the entire learning *(require frequent DRAM access)*



DNN+NeuroSim V2 Key Insights

VGG-8 on CIFAR10, with Novel Weight Mapping and Dataflow, on-chip training with 256 epochs										
Technology node (LSTP)	7 nm		32 nm							
Device	SRAM		SRAM		Ag:a-Si [23]	PCMO [24]	AlOx/HfO2 [25]	TaOx/HfOx (TsingHua) [2]	EpiRAM [4]	HZO FeFET (NotreDame) [6]
# of Conductance States	\	\	\	\	97	50				
ADC precision	Sequential	4-bit	Sequential	4-bit						
Weight / ΔWeight / Cell Precision	5-bit / 5-bit / 1-bit		5-bit / 5-bit / 1-bit							
Ron (Ω)	\	\	\	\					1 K	240 K
On/Off Ratio	\	\	\	\					0.2	100
Nonlinearity	\	\	\	\					-0.5	1.75/1.46
C2C Variation	\	\	\	\					00%	<0.5%
Write Pulse Voltage	\	\	\	\					-3V	3.65V/-2.95V
Write Pulse Width	\	\	\	\	300us/300us	1ms/1ms	100us/100us	50ns/50ns	5us/5us	75ns/75ns
Area (mm^2)	11.12	13.05	120.89	138.95	48.29	48.29	49.88	48.50	48.59	48.29
Memory Utilization (%)	94.62%				88.59%					
Training Accuracy (%)	91.00%				49.00%	56.00%	37.00%	81.00%	85.00%	91.00%
Training Latency (s) / Epoch	244.50	105.51								176.85
Training Dynamic Energy (J) / Epoch										92.11
Training Peak Latency (s) / Epoch										51.75
Training Peak Dynamic Energy (J) / Epoch										8.98
Training Throughput (TOPS)										1.04
Training Energy Efficiency (TOPS/W)	1.74	2.28	1.24	1.94	2.00	2.00	1.98	2.00	2.00	2.01
Training Peak Throughput (TOPS)	1.04	4.96	0.40	3.34	0.16	0.03	0.38	3.52	2.68	3.57
Training Peak Energy Efficiency (TOPS/W)	13.03	40.90	1.63	11.98	20.54	20.50	17.27	20.43	20.11	20.57

(1) Parallel-learning plays an important role to performance

(3) Desired nonlinearity: below +3/-3; cycle-to-cycle variation: <1%; to guarantee learning accuracy

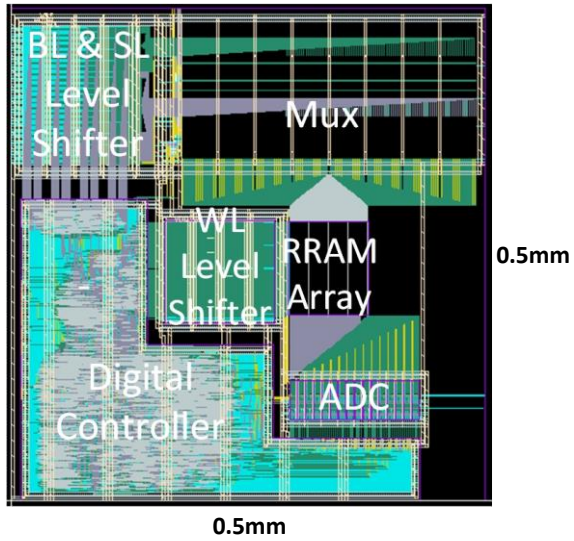
the weight-update will affect the throughput (batch-size=200)

(4) 7nm Parallel-SRAM shows superior performance; while

(5) DRAM access dominates the overall energy consumption, resulting in ~2 TOPS/W regardless of device technologies

NeuroSim Validation with Real Chip Implementation

CIM RRAM at TSMC 40nm (W. Li, et al. CICC'2021)



module		area (um^2)		latency	energy (pJ)	
		NeuroSim	Real chip		NeuroSim	Real chip
shift-add	DFF	719	681	1 cycle = 10ns	1.07 × γ = 0.67	0.68
	adder	662	663		1.36 × δ = 0.11	0.10
	inverter	2,133 INV = 1,336	1,334		1.99 × ϵ = 0.18	0.19
accum	DFF	4,968	4,706	1 cycle = 10ns	7.42 × γ = 4.67	4.67
	adder	3,089	3,291		6.28 × δ = 0.50	0.50
	inverter	10,869 INV = 6,808	6,797		10.15 × ϵ = 0.91	0.93
control	DFF	7,743	7,334		11.56 × γ = 7.28	7.28
	inverter	10,485 INV = 6,569	6,558		9.79 × ϵ = 0.88	0.91
total		31,893	31,386		15.20	15.26

Adjustment factors introduced to calibrate:

- **transistor sizing**
- **wiring area**
- **gate switching activity**
- **post-layout performance drop**

Energy for whole array	NeuroSim	Real chip
pre-layout	3,171.29pJ	10.4TOPS/W → 3,150.8pJ
post-layout	3,171.29pJ×ζ	8.48TOPS/W → 3,864.2pJ

After calibration, the prediction with post-layout simulation error rate is less than 2%.

A. Lu, et al. AICAS 2021

NeuroSim Extension Roadmap

One custom chip supports various DNN models that may have larger on-chip memory capacity than the chip could hold

Reconfigurable NeuroSim *[p1]*

Monolithic 3D partition between memory at legacy node with BEOL oxide transistors and peripheral logic at advanced 7nm node

Monolithic 3D NeuroSim *[p2]*

3D stacking with multi-tier memory interleaved with logic-tier by hybrid bonding and micro-TSV. Thermal modeling supported

Heterogeneous 3D NeuroSim *[p3]*

77K and 4K cryogenic transistor technology files calibrated for data-center computing and quantum peripheral control

Cryogenic NeuroSim *[p4]*

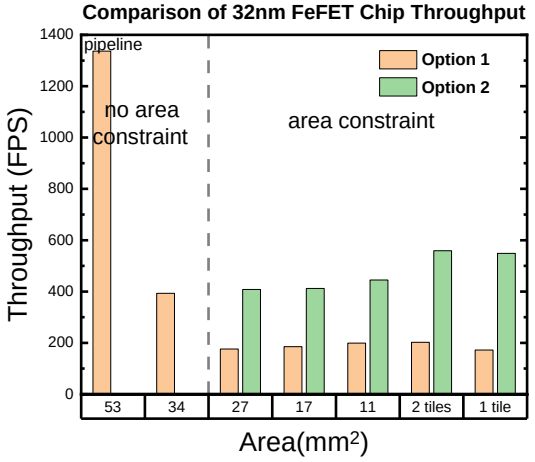
3D NAND based architecture for GB-level model for language, graph, genome,and recommendation system.

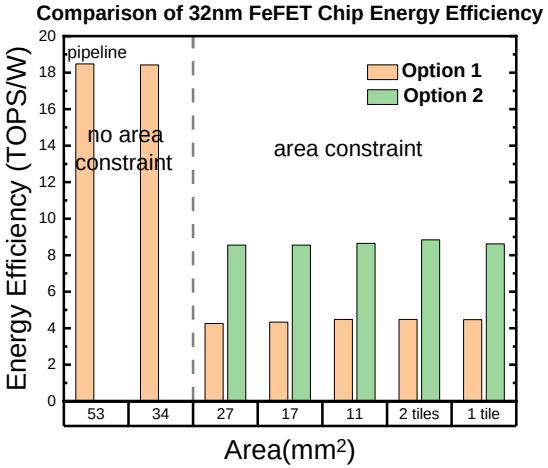
3D NAND NeuroSim *[p4]*

[p1] A. Lu et al., T-VLSI, 2021 & DATE 2020
[p2] X. Peng et al., IEDM, 2020
[p3] X. Peng et al., TED, 2021
[p4] P. Wang el al., ISCAS, 2021
[p5] W. Shim et al., EDL 2021

Reconfigurable CIM Design

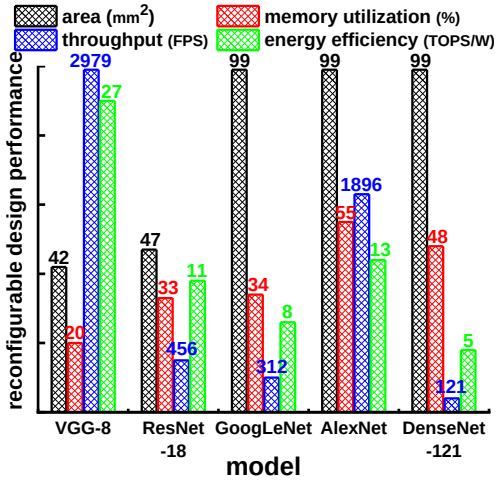
If chip area is constrained, off-chip weight reloading has two options:
Option 1: reload weights (with sequential processing)
Option 2: reload inputs (with batch processing)



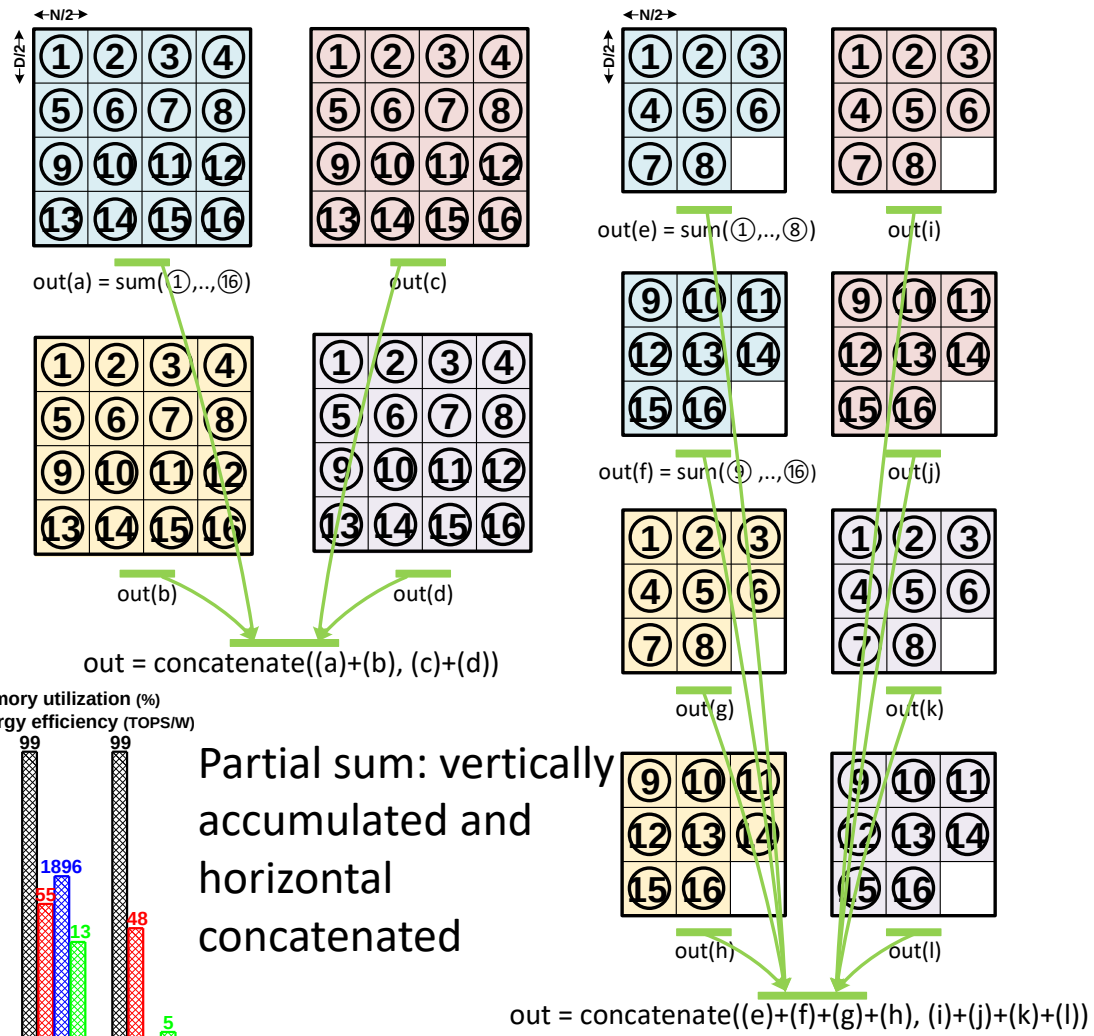


Go off-chip significantly lowers the throughput and energy efficiency ☹

A. Lu et al., T-VLSI, 2021 & DATE 2020

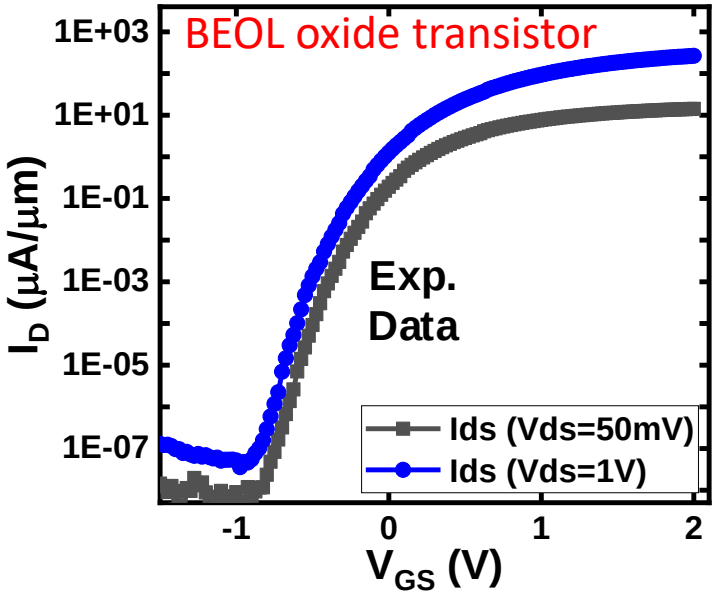


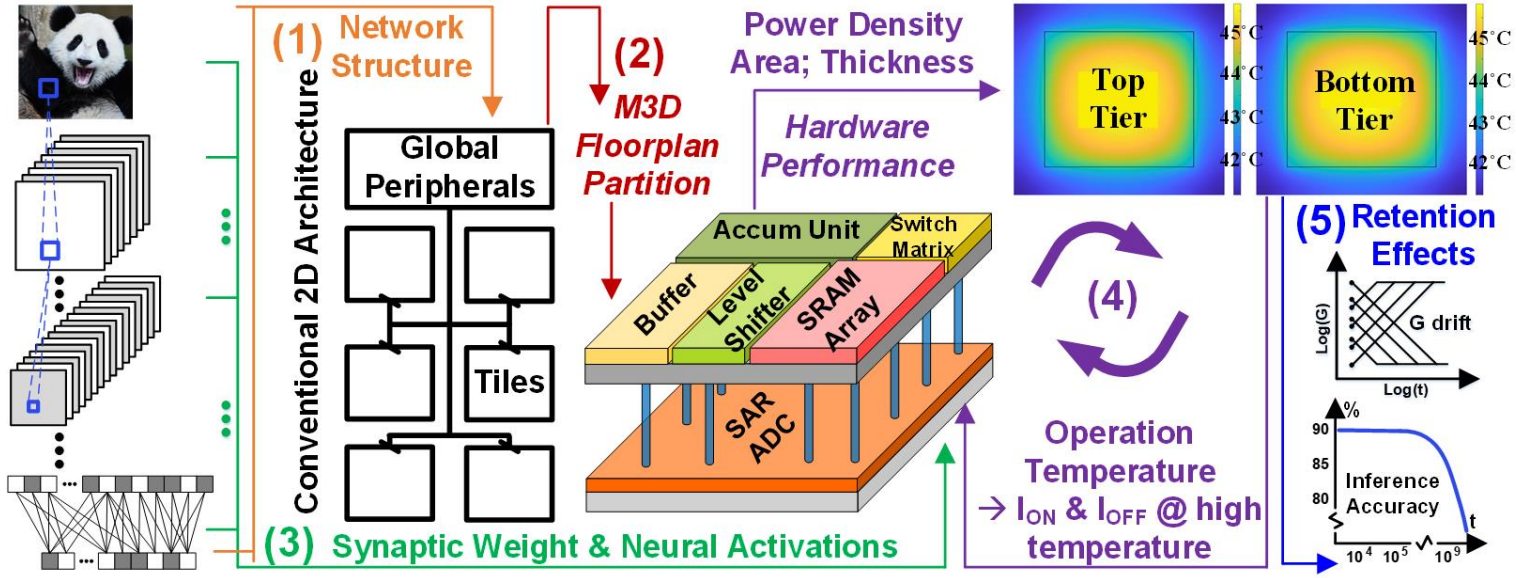
- Desired #PE = 4*4
  An example of 2*2 tiles
- Predefined #PE = 3*3



Partial sum: vertically accumulated and horizontal concatenated

Monolithic 3D





	Integration	2D			3D			
Architecture Specs	[LSTP] Tech node (nm)	45 [a]	7	7 [b]	45Si+45 [c]	45Ox+45 [d]	45Ox+7 [e]	7Si+7 [f]
	Device	RRAM		SRAM	RRAM			SRAM
	ADC precision	5-bit		4-bit	5-bit			4-bit
	Cell Precision	2-bit		1-bit	2-bit			1-bit
	Ron (Ω)	6 K		-	6 K			-
	On/Off Ratio	150		-	150			-
	numColMuxed	8			1			
Performance	Area (mm^2)	163.05	3.61	8.36	269.27	269.27	14.50	13.06
	Energy Efficiency (TOPS/W)	8.58	77.98	30.30	14.46	15.59	81.34	36.48
	Throughput (TOPS)	0.56	1.97	1.95	2.43	2.54	5.50	5.56
	Power Density (W/mm^2)	4.00E-04	7.00E-03	7.72E-03	6.23E-04	6.05E-04	4.66E-03	1.17E-02
	Via Density ($\#/mm^2$)	-	-	-	3.85E+03	3.85E+03	1.07E+05	1.56E+05

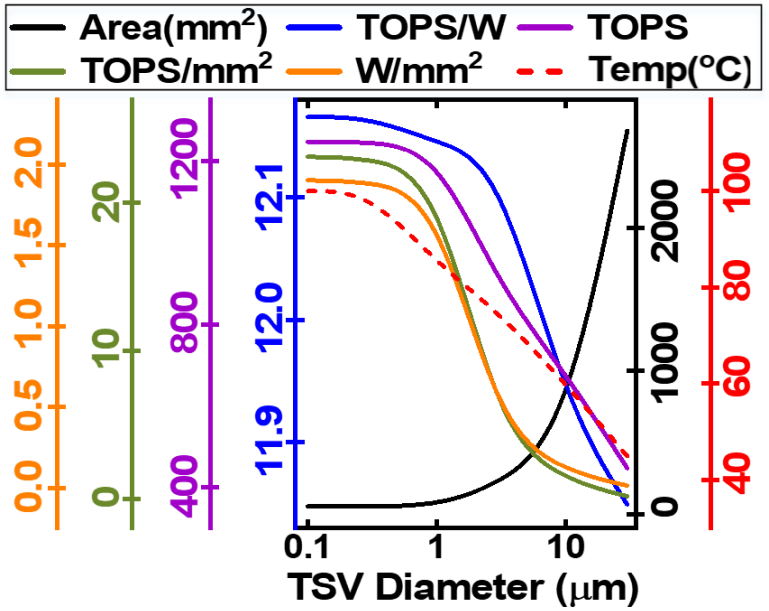
Partition the design into M3D:

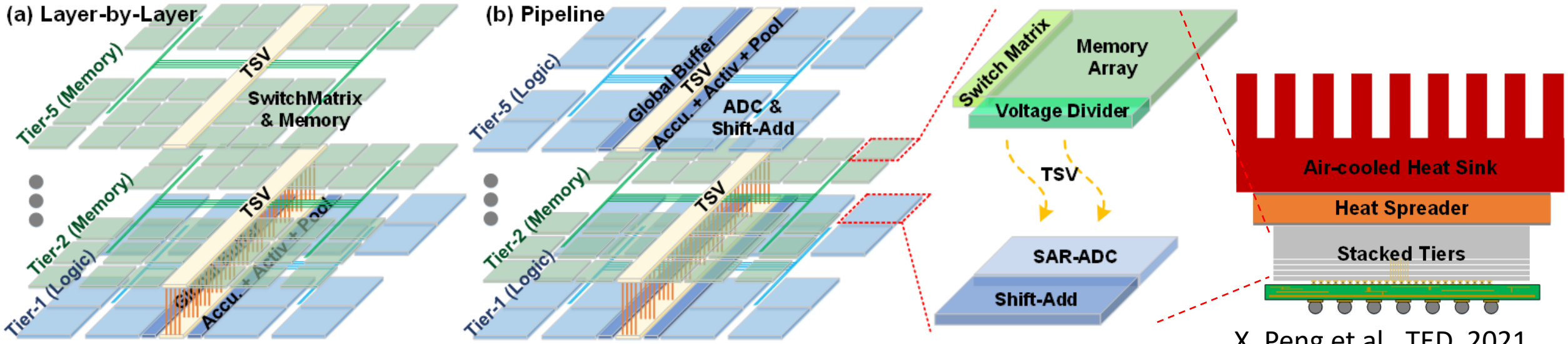
- **Si FEOL for logic and ADC modules at 7nm**
- **Oxide BEOL for memory and its periphery at 45nm**

X. Peng et al., IEDM, 2020

Heterogeneous 3D

- TSV and hybrid bonding for 5-tiers
- Two Scheme: “Layer-by-Layer” vs. “Pipeline”
- SRAM: both logic and memory tier @ 7nm
- RRAM: 2-bit/cell, RON is 6kΩ (on/off=150), memory tier @ 22nm, logic tier @ 7nm
- TSV diameter’s <u>sweet spot: 1~3um</u>

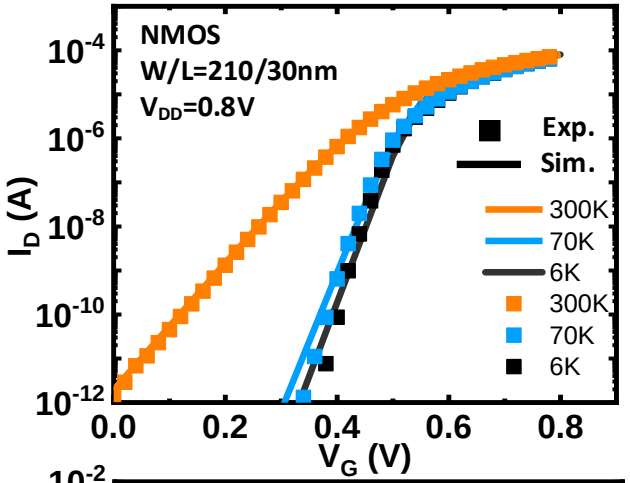


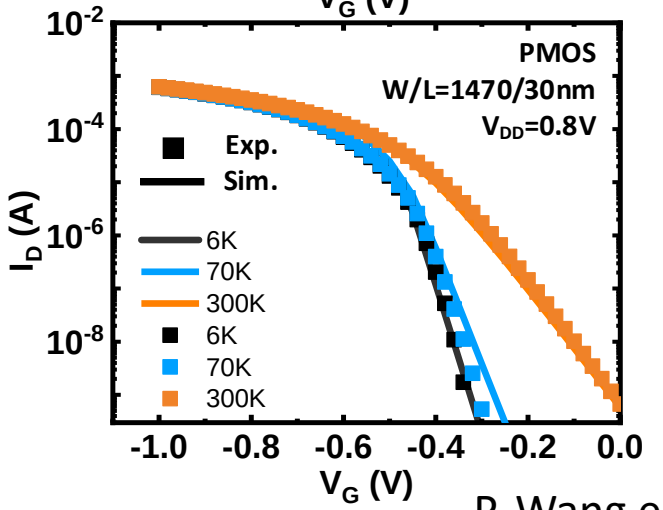


X. Peng et al., TED, 2021

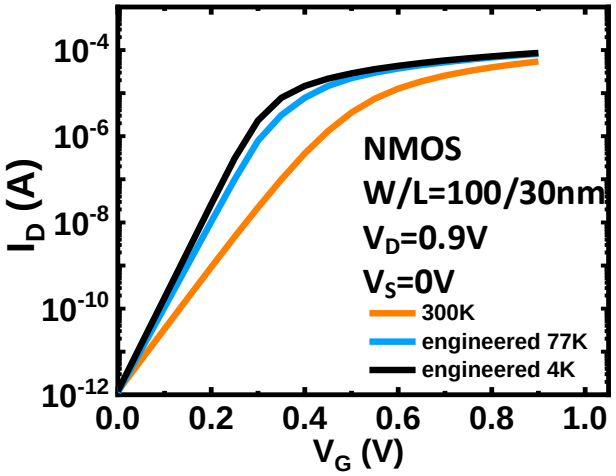
Cryogenic NeuroSim

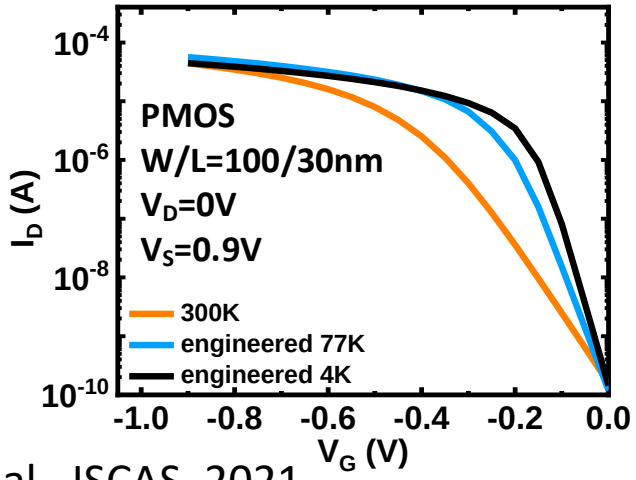
Exp Data





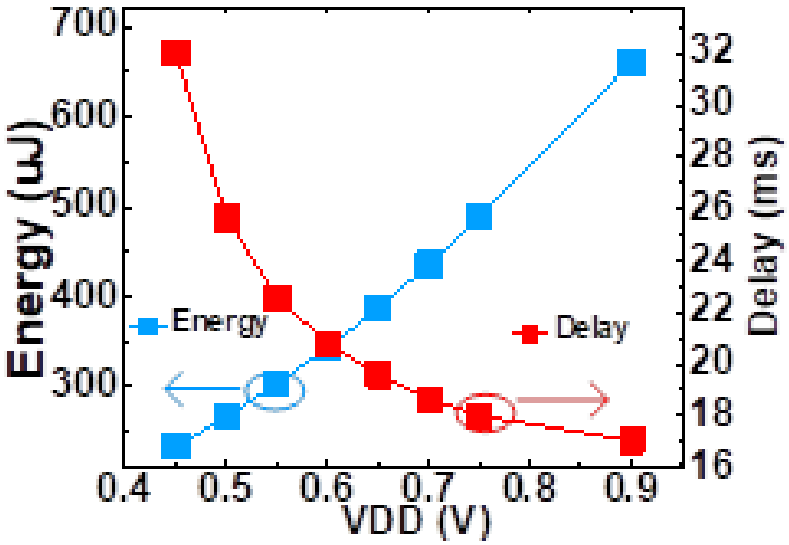
Vth engineering

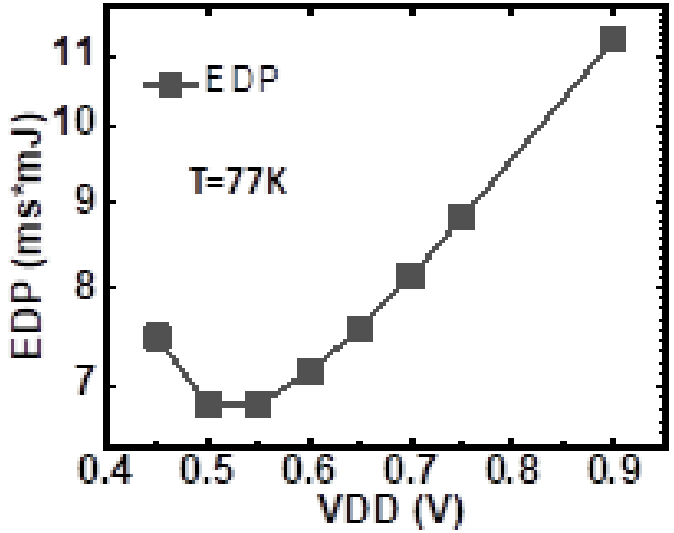




P. Wang el al., ISCAS, 2021

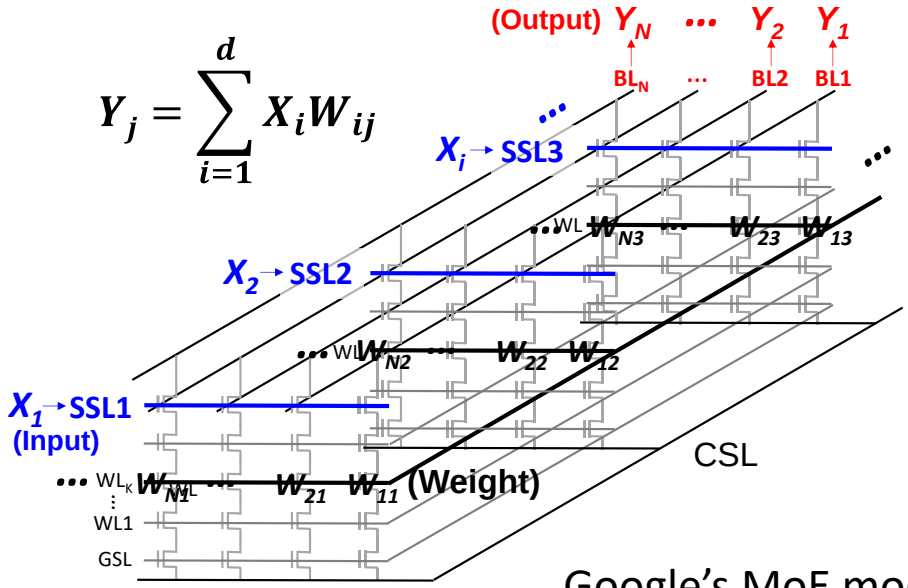
SRAM based CIM accelerator



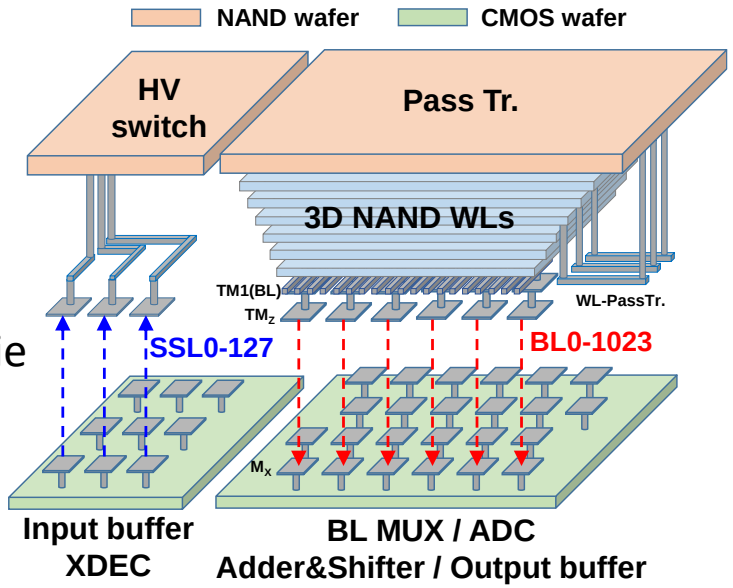


3D NAND based GB model

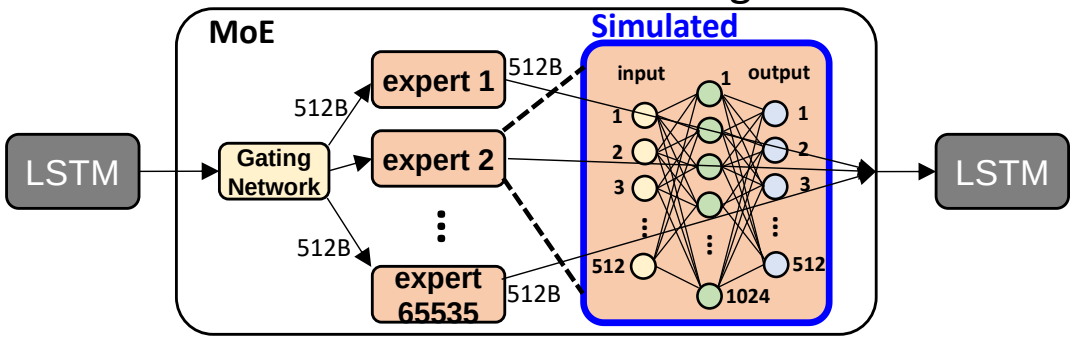
$$Y_j = \sum_{i=1}^{d} X_i W_{ij}$$



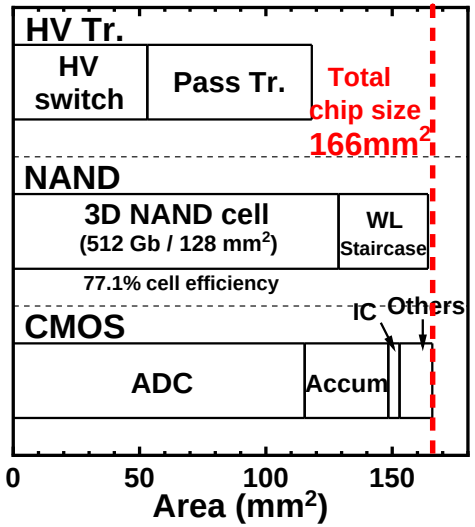
Hybrid bonding with 7nm logic die

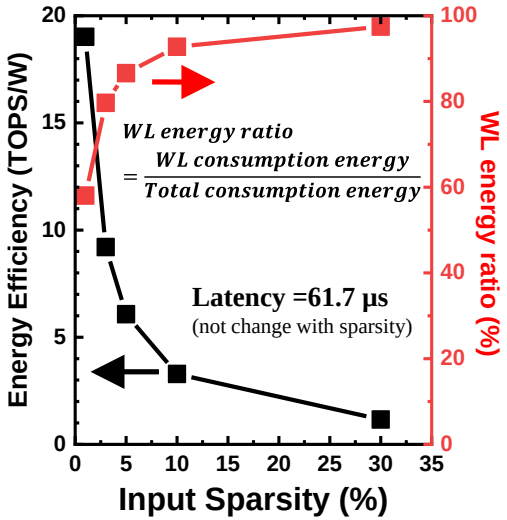


Google's MoE model



W. Shim et al., EDL 2021





Examples of Running DNN+NeuroSim V1.3

- NeuroSim core: device features in param.cpp
  - Type:

```
memcelltype = 2;            // 1: cell.memCellType = Type::SRAM
                            // 2: cell.memCellType = Type::RRAM
                            // 3: cell.memCellType = Type::FeFET
cellBit = 2;                        // precision of memory device
```

  - SRAM:

```
/*** parameters for SRAM ***/
// due the scaling, suggested SRAM cell size above 22nm: 160F^2
// SRAM cell size at 14nm: 300F^2
// SRAM cell size at 10nm: 400F^2
// SRAM cell size at 7nm: 600F^2
heightInFeatureSizeSRAM = 10;        // SRAM Cell height in feature size
widthInFeatureSizeSRAM = 28;        // SRAM Cell width in feature size
widthSRAMCellNMOS = 2;
widthSRAMCellPMOS = 1;
widthAccessCMOS = 1;
minSenseVoltage = 0.1;
```

  - eNVM:

```
/*** parameters for analog synaptic devices ***/
heightInFeatureSize1T1R = 4;        // 1T1R Cell height in feature size
widthInFeatureSize1T1R = 12;         // 1T1R Cell width in feature size
heightInFeatureSizeCrossbar = 2;    // Crossbar Cell height in feature size
widthInFeatureSizeCrossbar = 2;     // Crossbar Cell width in feature size

resistanceOn = 6e3;               // Ron resistance at Vr in the reported measurement data (need to
resistanceOff = 6e3*150;           // Roff resistance at Vr in the reported measurement dat (need t
maxConductance = (double) 1/resistanceOn;
minConductance = (double) 1/resistanceOff;

readVoltage = 0.5;                  // On-chip read voltage for memory cell
readPulseWidth = 10e-9;             // read pulse width in sec
accessVoltage = 1.1;                // Gate voltage for the transistor in 1T1R
resistanceAccess = resistanceOn*IR_DROP_TOLERANCE;            // resistance of access CMOS in 1T1R
writeVoltage = 2;                   // Enable level shifer if writeVoltage > 1.5V
```

Examples of Running DNN+NeuroSim V1.3

- NeuroSim core: circuit features in param.cpp
  - Technology:

```
// technode: 130     --> wireWidth: 175
// technode: 90      --> wireWidth: 110
// technode: 65      --> wireWidth: 105
// technode: 45      --> wireWidth: 80
// technode: 32      --> wireWidth: 56
// technode: 22      --> wireWidth: 40
// technode: 14      --> wireWidth: 25
// technode: 10, 7   --> wireWidth: 18
technode = 22;                    // Technology
featuresize = 40e-9;              // Wire width for subArray simulation
wireWidth = 40;                   // wireWidth of the cell for Accuracy calculation
```

  - Sub-array size:

```
numRowSubArray = 128;             // # of rows in single subArray
numColSubArray = 128;             // # of columns in single subArray
```

  - ADC type/sharing/resolution:

```
SARADC = false;            // false: MLSA
                           // true: sar ADC
currentMode = true;        // false: MLSA use VSA
                           // true: MLSA use CSA

numColMuxed = 8;                  // How many columns share 1 ADC (for eNVM and FeFET) or parallel SRAM
levelOutput = 32;                 // # of levels of the multilevelSenseAmp output, should be in 2^N forms; e.g. 32 levels --> 5-bit ADC
```

Examples of Running DNN+NeuroSim V1.3

- Python wrapper that interfaces with PyTorch
  - train.py (optional): some pre-trained models provided to skip training
    VGG-8 for CIFAR-10, DenseNet-40 for CIFAR-10, ResNet-18 for ImageNet
  - inference.py: accuracy estimation w/ hardware effects

Model/dataset/mode

```
parser.add_argument('--dataset', default='cifar10', help='cifar10|cifar100|imagenet')
parser.add_argument('--model', default='VGG8', help='VGG8|DenseNet40|ResNet18')
parser.add_argument('--mode', default='WAGE', help='WAGE|FP')
```

Resolution

```
parser.add_argument('--wl_weight', default=8)
parser.add_argument('--wl_grad', default=8)
parser.add_argument('--wl_activate', default=8)
parser.add_argument('--wl_error', default=8)
```

Hardware properties

```
# Hardware Properties
# if do not consider hardware effects, set inference=0
parser.add_argument('--inference', default=0, help='run hardware inference simulation')
parser.add_argument('--subArray', default=128, help='size of subArray (e.g. 128*128)')
parser.add_argument('--ADCprecision', default=5, help='ADC precision (e.g. 5-bit)')
parser.add_argument('--cellBit', default=1, help='cell precision (e.g. 4-bit/cell)')
parser.add_argument('--onoffratio', default=10, help='device on/off ratio (e.g. Gmax/Gmin = 3)')
```

Nonideal effects (variation, retention)

```
# if do not run the device retention / conductance variation effects, set vari=0, v=0
parser.add_argument('--vari', default=0, help='conductance variation (e.g. 0.1 standard deviation to generate random variation)')
parser.add_argument('--t', default=0, help='retention time')
parser.add_argument('--v', default=0, help='drift coefficient')
parser.add_argument('--detect', default=0, help='if 1, fixed-direction drift, if 0, random drift')
parser.add_argument('--target', default=0, help='drift target for fixed-direction drift')
current_time = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
```

Examples of Running DNN+NeuroSim V1.3

- Optional: self-defined network models
  - Create your own file in “models” folder: Replace nn.Conv2d / nn.Linear as QConv2d / QLinear

```
self.conv1 = make_layers([('C', 3, num_planes, 3, 'same', 1)], args, logger)
```

  - make_layers function (just call it, no need to change)

```
def make_layers(cfg, args, logger):
    global name
    layers = []
    for i, v in enumerate(cfg):
        if v[0] == 'M':
            layers += [nn.MaxPool2d(kernel_size=v[1], stride=v[2])]
        if v[0] == 'C':
            in_channels = v[1]
            out_channels = v[2]
            if v[4] == 'same':
                padding = v[3]//2
            else:
                padding = 0
            if args.mode == "WAGE":
                conv2d = QConv2d(in_channels, out_channels, kernel_size=v[3], stride=v[5], padding=padding,
                                 logger=logger,wl_input = args.wl_activate,wl_activate=args.wl_activate,
                                 wl_error=args.wl_error,wl_weight= args.wl_weight,inference=args.inference,onoffratio=args.onoffratio,cellBit=args.cellBit,
                                 subArray=args.subArray,ADCprecision=args.ADCprecision,vari=args.vari,t=args.t,v=args.v,detect=args.detect,target=args.target,
                                 name = 'Conv'+str(name)+'_' )
            elif args.mode == "FP":
                conv2d = nn.Conv2d(in_channels, out_channels, kernel_size=v[3], stride=v[5], padding=padding, bias=False)
            name += 1
            batchnorm = nn.BatchNorm2d(out_channels)
            non_linearity_activation =  nn.ReLU()
            layers += [conv2d, batchnorm, non_linearity_activation]
            in_channels = out_channels
        if v[0] == 'L':
            if args.mode == "WAGE":
                linear = QLinear(in_features=v[1], out_features=v[2],
                                logger=logger, wl_input = args.wl_activate,wl_activate=args.wl_activate,wl_error=args.wl_error,
                                wl_weight=args.wl_weight,inference=args.inference,onoffratio=args.onoffratio,cellBit=args.cellBit,
                                subArray=args.subArray,ADCprecision=args.ADCprecision,vari=args.vari,t=args.t,v=args.v,detect=args.detect,target=args.target,
                                name='FC'+str(i)+'_' )
            elif args.mode == "FP":
                linear = nn.Linear(in_features=v[1], out_features=v[2], bias=False)
            layers += [linear]
    return nn.Sequential(*layers)
```

Examples of Running DNN+NeuroSim V1.3

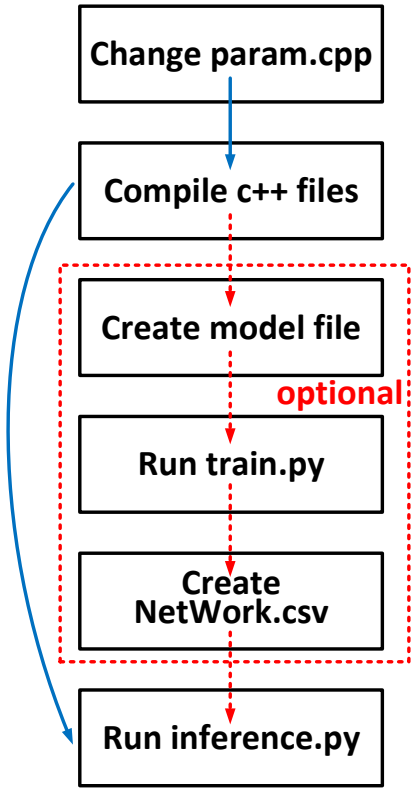
- Optional: self-defined network models
  - Create your own NetWork.csv

Example: VGG-8

	IFM Length	IFM Width	IFM Channel Depth	Kernel Length	Kernel Width	Kernel Depth	Followed by pooling or not?
Layer 1	32	32	3	3	3	128	0
Layer 2	32	32	128	3	3	128	1
Layer 3	16	16	128	3	3	256	0
Layer 4	16	16	256	3	3	256	1
Layer 5	8	8	256	3	3	512	0
Layer 6	8	8	512	3	3	512	1
Layer 7	1	1	8192	1	1	1024	0
Layer 8	1	1	1024	1	1	10	0

Examples of Running DNN+NeuroSim V1.3

- Workflow:



- Output example:
  - Accuracy
  - Floorplan

```
Test set: Average loss: 1.5865, Accuracy: 9007/10000 (90%)
---------------------------- FloorPlan ------------------------------

Tile and PE size are optimized to maximize memory utilization ( = memory mapped by synapse / total memory on chip)

Desired Conventional Mapped Tile Storage Size: 1024x1024
Desired Conventional PE Storage Size: 512x512
Desired Novel Mapped Tile Storage Size: 9x512x512
User-defined SubArray Size: 128x128

----------------- # of tile used for each layer -----------------
layer1: 1
layer2: 1
...
----------------- Speed-up of each layer ------------------
layer1: 16
layer2: 4
...
---------------- Utilization of each layer -----------------
layer1: 0.210938
layer2: 1
...
Memory Utilization of Whole Chip: 96.8584 %

---------------------------- FloorPlan Done ------------------------------
```

Examples of Running DNN+NeuroSim V1.3

- Output example:
  - each layer performance breakdown

```
------------------------------------ Hardware Performance ------------------------------------
-------------------- Estimation of Layer 1 ----------------------
layer1's readLatency is: 377982ns
layer1's readDynamicEnergy is: 1.89268e+06pJ
layer1's leakagePower is: 8.47407uW
layer1's leakageEnergy is: 147340pJ
layer1's buffer latency is: 337150ns
layer1's buffer readDynamicEnergy is: 22681.5pJ
layer1's ic latency is: 24831ns
layer1's ic readDynamicEnergy is: 487920pJ

************************* Breakdown of Latency and Dynamic Energy *************************

----------- ADC (or S/As and precharger for SRAM) readLatency is : 7385.06ns
----------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readLatency is : 7385.06ns
----------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readLatency is : 363212ns
----------- ADC (or S/As and precharger for SRAM) readDynamicEnergy is : 1.03834e+06pJ
----------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readDynamicEnergy is : 155498pJ
----------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readDynamicEnergy is : 698849pJ

************************* Breakdown of Latency and Dynamic Energy *************************
```

Latency & energy of:

- Whole layer
- Buffer
- Interconnection
- ADC
- Accumulation
- Other

Leakage

Examples of Running DNN+NeuroSim V1.3

- Output examples:
  - System-level summary

```
---------------------------- Summary ------------------------------

ChipArea : 4.5554e+07um^2
Chip total CIM array : 1.5408e+06um^2
Total IC Area on chip (Global and Tile/PE local): 6.17954e+06um^2
Total ADC (or S/As and precharger for SRAM) Area on chip : 1.46062e+07um^2
Total Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) on chip : 3.13322e+06um^2
Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, pooling and activation units) : 2.00943e+07um^2

Chip clock period is: 2.05141ns
Chip layer-by-layer readLatency (per image) is: 1.45613e+06ns
Chip total readDynamicEnergy is: 4.03921e+07pJ
Chip total leakage Energy is: 1.19011e+06pJ
Chip total leakage Power is: 628.781uW
Chip buffer readLatency is: 1.00082e+06ns
Chip buffer readDynamicEnergy is: 412969pJ
Chip ic readLatency is: 141884ns
Chip ic readDynamicEnergy is: 8.94262e+06pJ

************************ Breakdown of Latency and Dynamic Energy *************************

----------- ADC (or S/As and precharger for SRAM) readLatency is : 63462.3ns
----------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readLatency is : 246001ns
----------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readLatency is : 1.14667e+06ns
----------- ADC (or S/As and precharger for SRAM) readDynamicEnergy is : 2.08642e+07pJ
----------- Accumulation Circuits (subarray level: adders, shiftAdds; PE/Tile/Global level: accumulation units) readDynamicEnergy is : 5.16853e+06pJ
----------- Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readDynamicEnergy is : 1.43594e+07pJ

************************ Breakdown of Latency and Dynamic Energy *************************


---------------------------- Performance ------------------------------
Energy Efficiency TOPS/W (Layer-by-Layer Process): 24.282
Throughput TOPS (Layer-by-Layer Process): 0.845963
Throughput FPS (Layer-by-Layer Process): 686.75
Compute efficiency TOPS/mm^2 (Layer-by-Layer Process): 0.0185705
------------------------------------ Hardware Performance Done ------------------------------------

---------------------------- Simulation Performance ------------------------------
Total Run-time of NeuroSim: 151 seconds
---------------------------- Simulation Performance ------------------------------
```

Area

Latency & energy of:
- Whole chip
- Buffer
- Interconnection
- ADC
- Accumulation
- Other

Leakage

- Energy efficiency
- Throughput
- Compute efficiency

Acknowledgement

- NeuroSim project contributors: Pai-Yu Chen, Xiaochen Peng, Shanshi Huang, Yandong Luo, Anni Lu, etc...