STONNE

A **S**imulation **TO**ol for **N**eural **N**etwork **E**ngines

José L. Abellán, PhD

jlabellan@ucam.edu

Universidad Católica de Murcia (UCAM)

Organizers

Tushar Krishna
Associate Professor, School of ECE
Georgia Institute of Technology
tushar@ece.gatech.edu

José L. Abellán
Associate Professor
Universidad Católica de Murcia
jlabellan@ucam.edu

Manuel E. Acacio
Full Professor
University of Murcia
meacacio@um.es

Raveesh Garg
Ph.D Student
Georgia Institute of Technology
raveesh.g@gatech.edu

Francisco Munoz-Martinez
Ph.D Student
University of Murcia
francisco.munoz2@um.es

Agenda

Attention: Tutorial is being recorded

Time (CET)	Time (ET)	Topic	Presenter
14:00 – 14:40	8:00 – 8:40	**Flexible Accelerators**	Tushar Krishna
14:40 – 15:10	8:40 – 9:10	**Cycle accurate simulation and Overview of STONNE**	José Luis Abellán
15:10 – 16:10	9:10 – 10:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
16:10 – 16:40	10:10 – 10:40	**Coffee Break**	
16:40 – 17:10	10:40 – 11:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
17:10 – 17:40	11:10 – 11:40	**Dataflow exploration for Graph Neural Networks**	Raveesh Garg
17:50 – 18:00	11:50 – 12:00	**Roadmap for Future Development**	Manuel Acacio

Tutorial Website https://stonne-simulator.github.io/ASPLOSTUT.html
includes agenda and STONNE/OMEGA installation instructions

Agenda

Attention: Tutorial is being recorded

Time (CET)	Time (ET)	Topic	Presenter
14:00 – 14:40	8:00 – 8:40	**Flexible Accelerators**	Tushar Krishna
14:40 – 15:10	8:40 – 9:10	**Cycle accurate simulation and Overview of STONNE**	José Luis Abellán
15:10 – 16:10	9:10 – 10:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
16:10 – 16:40	10:10 – 10:40	**Coffee Break**	
16:40 – 17:10	10:40 – 11:10	**(Hands-on) STONNE Deep-Dive**	Francisco Muñoz-Martínez
17:10 – 17:40	11:10 – 11:40	**Dataflow exploration for Graph Neural Networks**	Raveesh Garg
17:50 – 18:00	11:50 – 12:00	**Roadmap for Future Development**	Manuel Acacio

Tutorial Website https://stonne-simulator.github.io/ASPLOSTUT.html

includes agenda and STONNE/OMEGA installation instructions

Outline

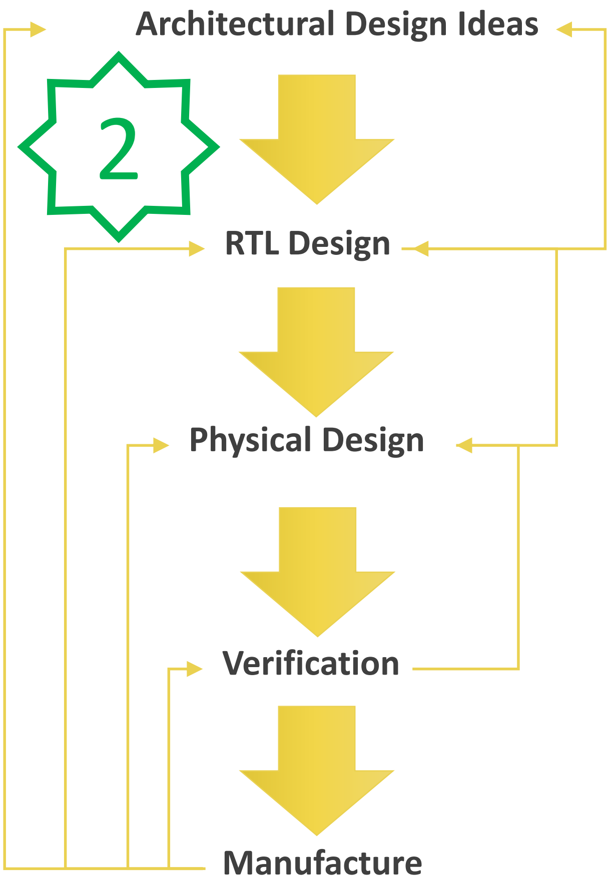
- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

Outline

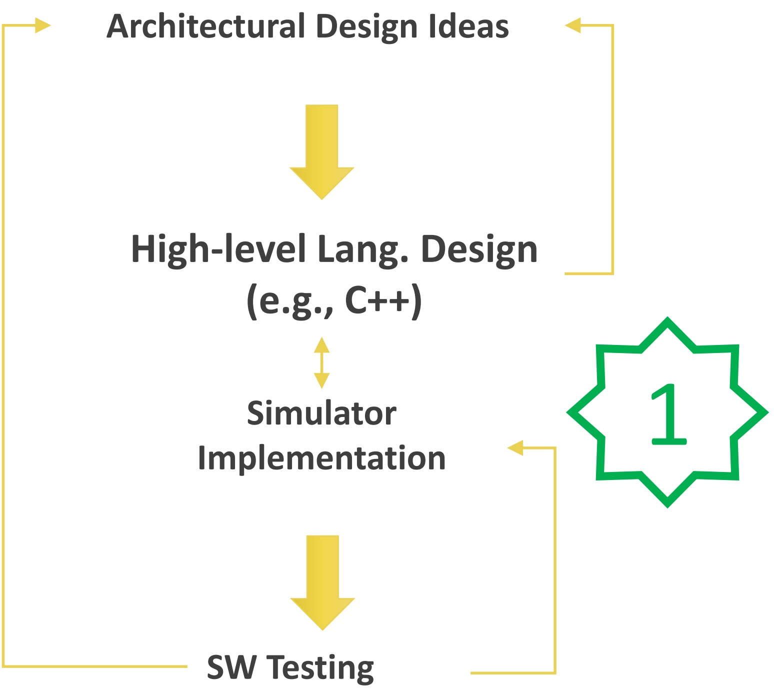
- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

Cycle-level Architectural Simulation

Chip-Designing Process



Architectural Simulation



Accurate Architectural Design Prototyping!

Cycle-level Architectural Simulators

- Microarchitectural simulators have been extensively used during the design process of CPUs (**Gem5**) and GPUs (**MGPUSim**)
- However, how can we simulate the wide diversity of DNN accelerators (i.e., rigid, flexible and data-dependent optimizations)?
  - We need cycle-level simulation.
  - We need to support flexible (dense and sparse) and rigid accelerators.
  - We need to perform end-to-end simulation.

Cycle-level Architectural Simulators

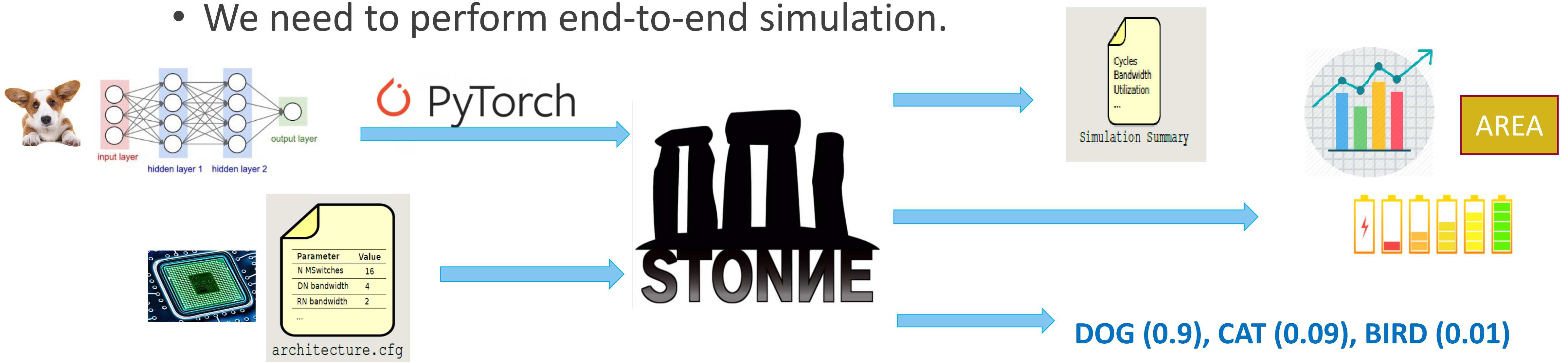
- Microarchitectural simulators have been extensively used during the design process of CPUs (**Gem5**) and GPUs (**MGPUSim**)
- However, how can we simulate the wide diversity of DNN accelerators (i.e., rigid, flexible and data-dependent optimizations)?
  - We need cycle-level simulation.
  - We need to support flexible (dense and sparse) and rigid accelerators.
  - We need to perform end-to-end simulation.

STONNE (A Simulation Tool for Neural Network Engines)

STONNE

Cycle-level Architectural Simulators

- Microarchitectural simulators have been extensively used during the design process of CPUs (**Gem5**) and GPUs (**MGPUSim**)
- However, how can we simulate the wide diversity of DNN accelerators (i.e., rigid, flexible and data-dependent optimizations)?
  - We need cycle-level simulation.
  - We need to support flexible (dense and sparse) and rigid accelerators.
  - We need to perform end-to-end simulation.



Why another simulator for DNN accelerators?

Why another simulator for DNN accelerators?

	Cycle Level	Architecture Type	Sparsity Support	FullModel Eval	DataDep Opt
MAERI BSV	✓	Flexible	✗	✗	✗
SIGMA RTL	✓	Flexible	✓	✗	✗
SCALE-Sim	✗	Rigid	✗	✗	✗
MAESTRO TimeLoop	✗	Both	✗	✗	✗
DNNSim	✗	Rigid	✓	✗	✗
SMAUG	✓	Rigid	✗	✓	✗
STONNE	✓	Both	✓	✓	✓

State-of-the-art Simulators for DNN Accelerators

Why another simulator for DNN accelerators?

	Cycle Level	Architecture Type	Sparsity Support	FullModel Eval	DataDep Opt
MAERI BSV	✓	Flexible	✗	✗	✗
SIGMA RTL	✓	Flexible	✓	✗	✗
SCALE-Sim	✗	Rigid	✗	✗	✗
MAESTRO TimeLoop	✗	Both	✗	✗	✗
DNNSim	✗	Rigid	✓	✗	✗
SMAUG	✓	Rigid	✗	✓	✗
STONNE	✓	Both	✓	✓	✓

RTL implementations are slow to modify

State-of-the-art Simulators for DNN Accelerators

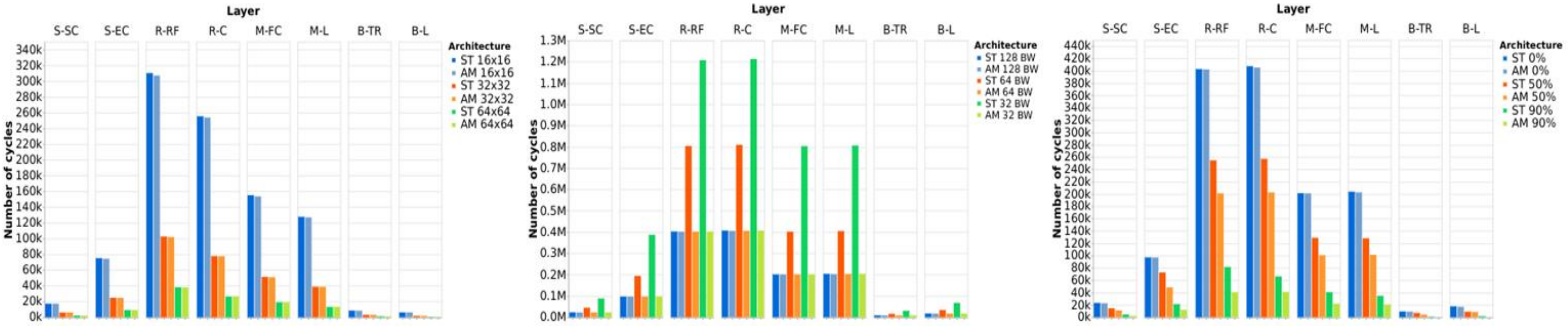
Why another simulator for DNN accelerators?

	Cycle Level	Architecture Type	Sparsity Support	FullModel Eval	DataDep Opt
MAERI BSV	✓	Flexible	✗	✗	✗
SIGMA RTL	✓	Flexible	✓	✗	✗
SCALE-Sim	✗	Rigid	✗	✗	✗
MAESTRO TimeLoop	✗	Both	✗	✗	✗
DNNSim	✗	Rigid	✓	✗	✗
SMAUG	✓	Rigid	✗	✓	✗
STONNE	✓	Both	✓	✓	✓

Analytical models are not accurate for complex designs

State-of-the-art Simulators for DNN Accelerators

Why another simulator for DNN accelerators?



(a) Rigid TPU varying array sizes.

(b) Flexible MAERI varying bandwidth (BW).

(c) Flexible SIGMA varying sparsity (%).

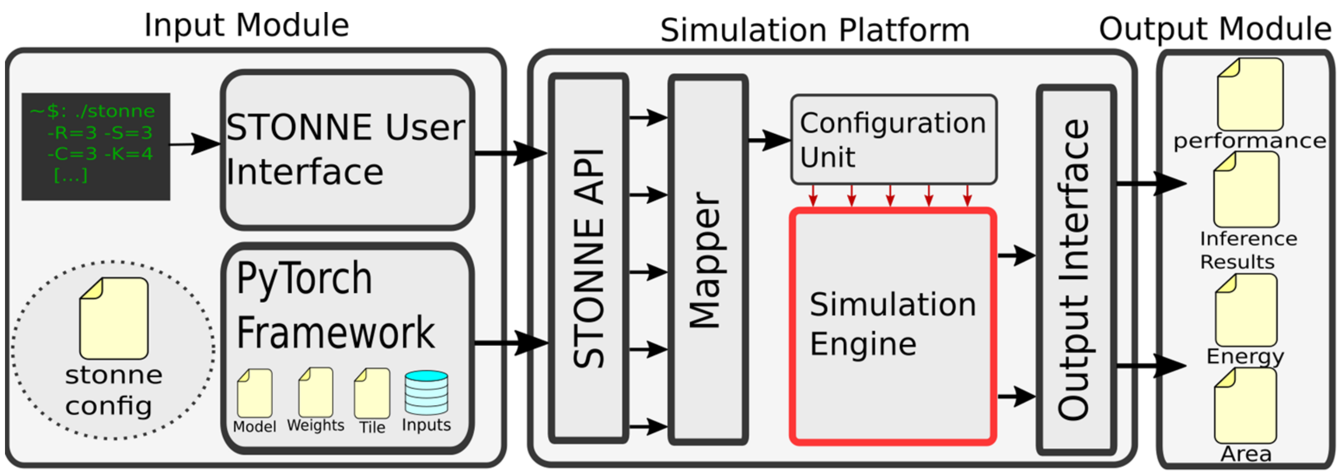
Cycle-level simulation is required to faithfully model complex DNN architectures

Outline

- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

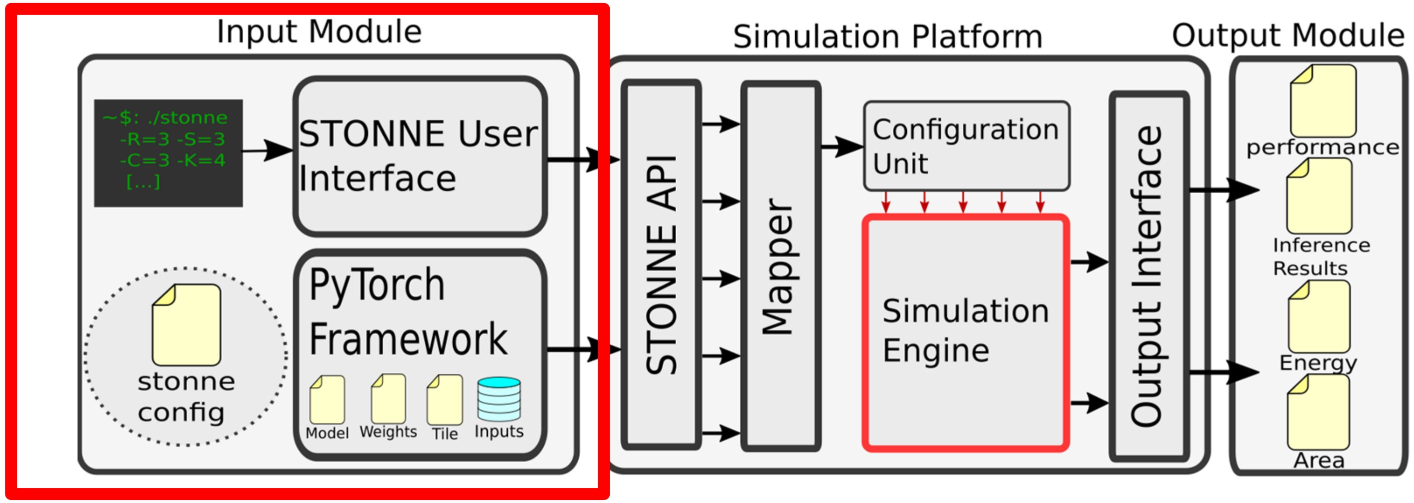
STONNE Framework

- STONNE is a cycle-level microarchitectural simulator written in C++ for DNN inference accelerators.
- Open-Sourced under the MIT License:
  - https://github.com/stonne-simulator/stonne
- STONNE is composed of 3 main building blocks:



Input Module

- STONNE is a cycle-level microarchitectural simulator written in C++ for DNN inference accelerators.
- Open-Sourced under the MIT License:
  - https://github.com/stonne-simulator/stonne
- STONNE is composed of 3 main building blocks:



Input Module

- **STONNE User Interface**: A command line that the user can use to run an instance of the simulator with random input values.

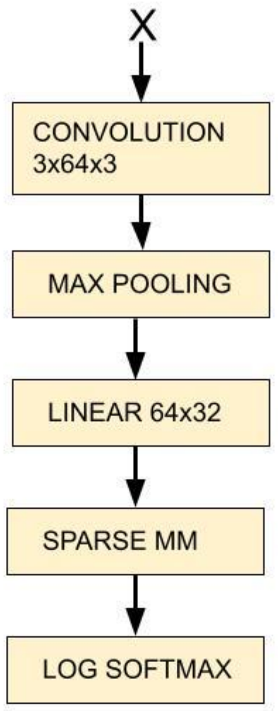
```
[PC@User $] ./stonne -CONV -R=3 -S=3 -C=1 -G=1 -K=1 -N=1 -X=4 -Y=4 -T_R=3 -T_S=3 -T_C=1
-T_G=1 -T_K=1 -T_N=1 -T_X_=1 -T_Y_=1 -num_ms=64 -dn_bw=64 -rn_bw=64
[PC@User $] Running CONV layer in STONNE Simulator...
[PC@User $] Output files generated correctly.
```

Input Module

- **PyTorch Framework**: We integrate Pytorch with STONNE so that instances of operations are off-loaded to the simulator.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class My_DNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv = nn.conv2d(3, 64, 3)
    self.max = nn.MaxPool2d(kernel_size=3)
    self.fc = nn.Linear(64, 32)
  def forward(self, x, y):
    x = self.conv(x) #on cpu
    x = self.max(x) #on cpu
    x = self.fc(x) #on cpu
    x = F.sparse_mm(x, y) #on cpu
    x = F.log_softmax(x, dim=1) #on cpu
    return x
```

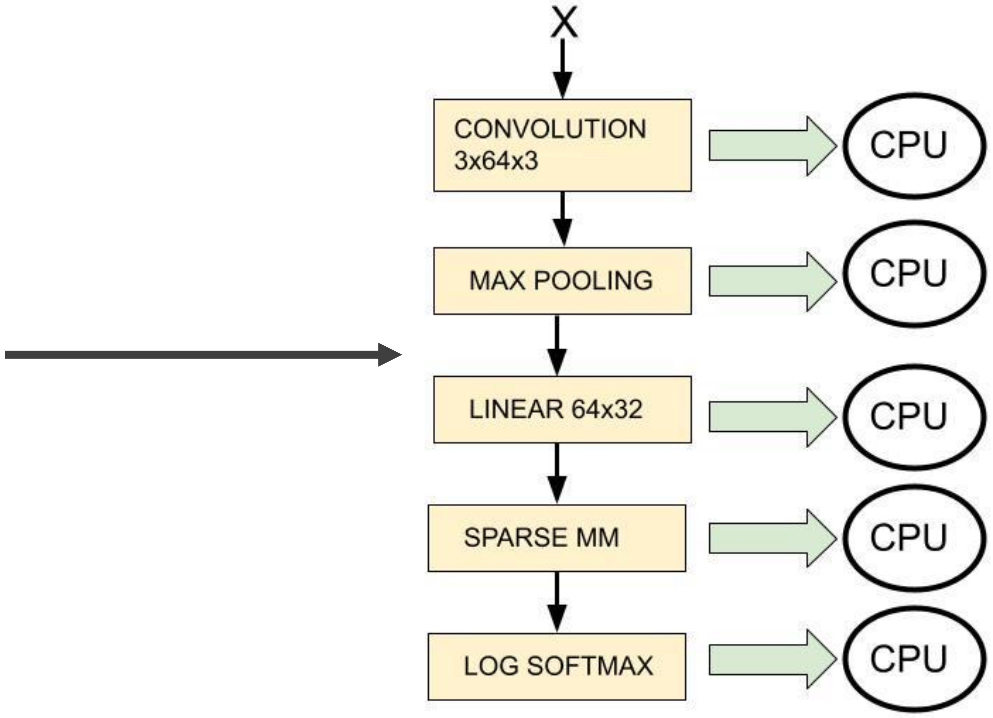


Input Module

- **PyTorch Framework**: We integrate Pytorch with STONNE so that instances of operations are off-loaded to the simulator.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class My_DNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv = nn.conv2d(3, 64, 3)
    self.max = nn.MaxPool2d(kernel_size=3)
    self.fc = nn.Linear(64, 32)
  def forward(self, x, y):
    x = self.conv(x) #on cpu
    x = self.max(x) #on cpu
    x = self.fc(x) #on cpu
    x = F.sparse_mm(x, y) #on cpu
    x = F.log_softmax(x, dim=1) #on cpu
    return x
```

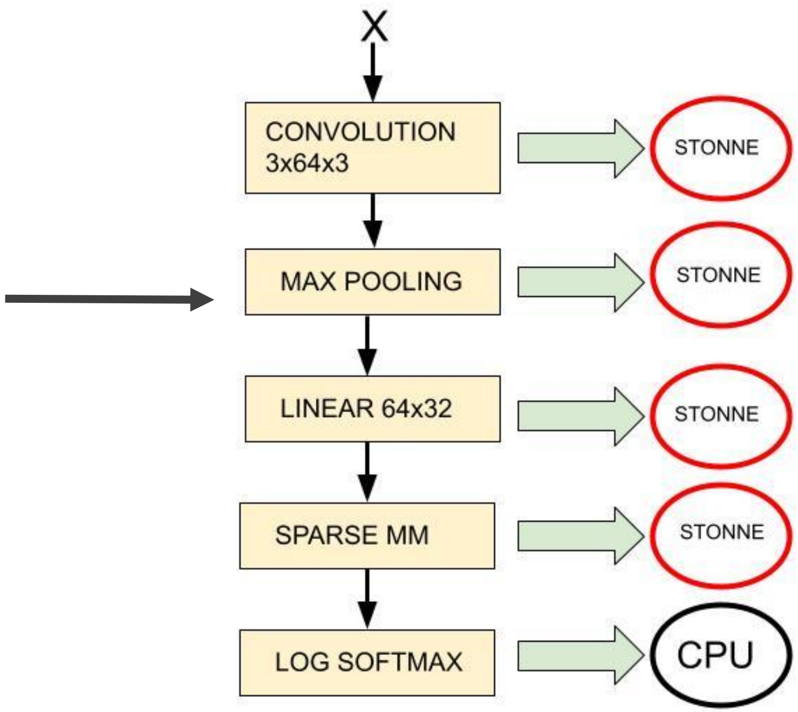


Input Module

- **PyTorch Framework**: We integrate Pytorch with STONNE so that instances of operations are off-loaded to the simulator.

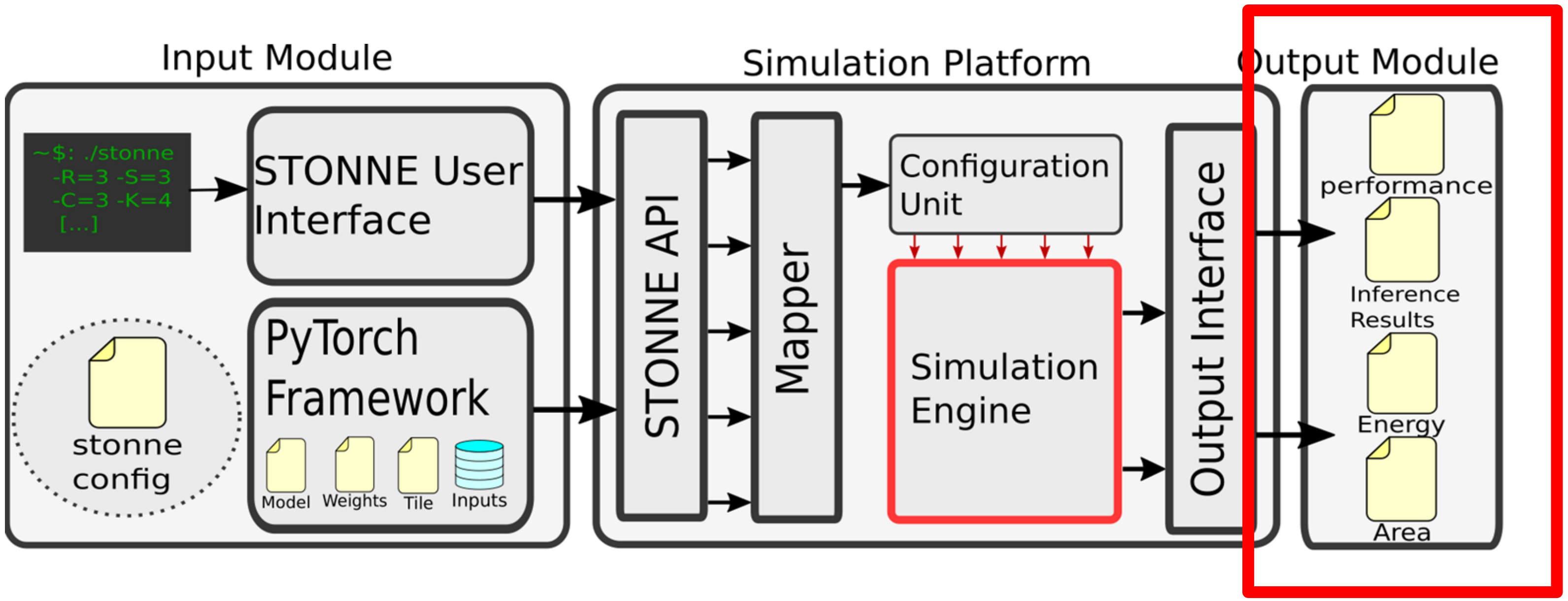
```
import torch
import torch.nn as nn
import torch.nn.functional as F

class My_DNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv = nn.Simulatedconv2d(3, 64, 3, conf='stonne_hw.cfg')
    self.max = nn.SimulatedMaxPool2d(kernel_size=3, conf='stonne_hw.cfg')
    self.fc = nn.SimulatedLinear(64, 32, conf='stonne_hw.cfg')
  def forward(self, x, y):
    x = self.conv(x) #simulated
    x = self.max(x) #simulated
    x = self.fc(x) #simulated
    x = F.simulated_sparse_mm(x,y,conf='stonne_hw.cfg')#simulated
    x = F.log_softmax(x, dim=1) #on cpu
    return x
```



Output Module

- STONNE is a cycle-level microarchitectural simulator written in C++ for DNN inference accelerators.
- Open-Sourced under the MIT License:
  - https://github.com/stonne-simulator/stonne
- STONNE is constituted by 3 main building blocks:



Output Module

- This module reports simulation statistics such as performance, compute unit utilization, and activity counts of different components such as wires, FIFOs or SRAM usage.


```
"MSNetworkStats" : {
      "MSwitchStats" : [
         {
            "Total_cycles" : 396168,
            "Idle_cycles" : 1896,
            "N_multiplications" : 394272,
            "N_input_forwardings_send" : 0,
            "N_input_forwardings_receive" : 392496,
            "N_inputs_receive_from_memory" : 1776,
            "N_weights_receive_from_memory" : 8,
            "N_weight_fifo_flush" : 7,
            "N_psums_receive" : 0,
            "N_psum_forwarding_send" : 0,
            "N_configurations" : 1
            ,"ActivationFifo" : {
               "N_pops" : 1776,
               "N_pushes" : 1776,
               "N_fronts" : 0,
               "Max_occupancy" : 1
            .....
```


Statistics.json

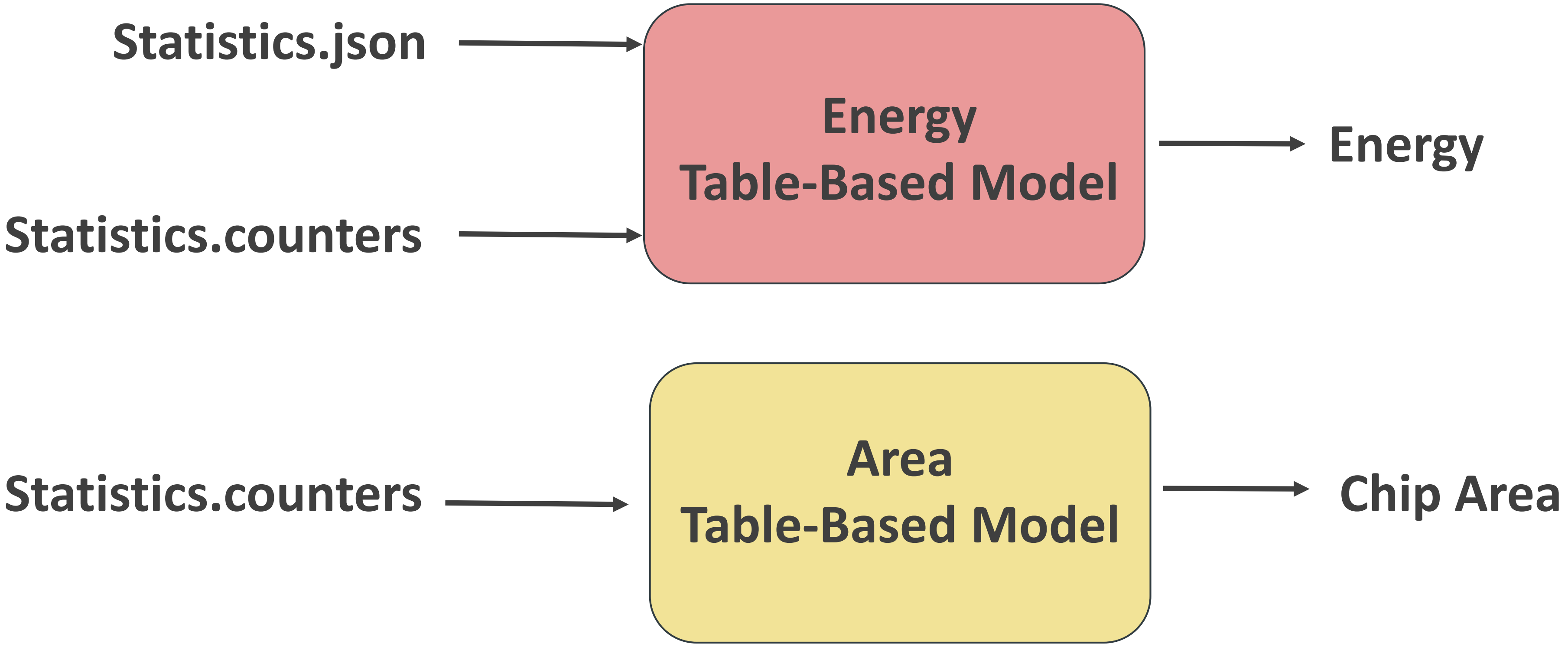

```
[MSNetwork]
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=0 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=0 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=0 READ=0
MN_WIRE WRITE=392496 READ=0
MN_WIRE WRITE=392496 READ=0
[GlobalBuffer]
GLOBALBUFFER READ=3582144 WRITE=3154176
....
```


Statistics.counters

Output Module

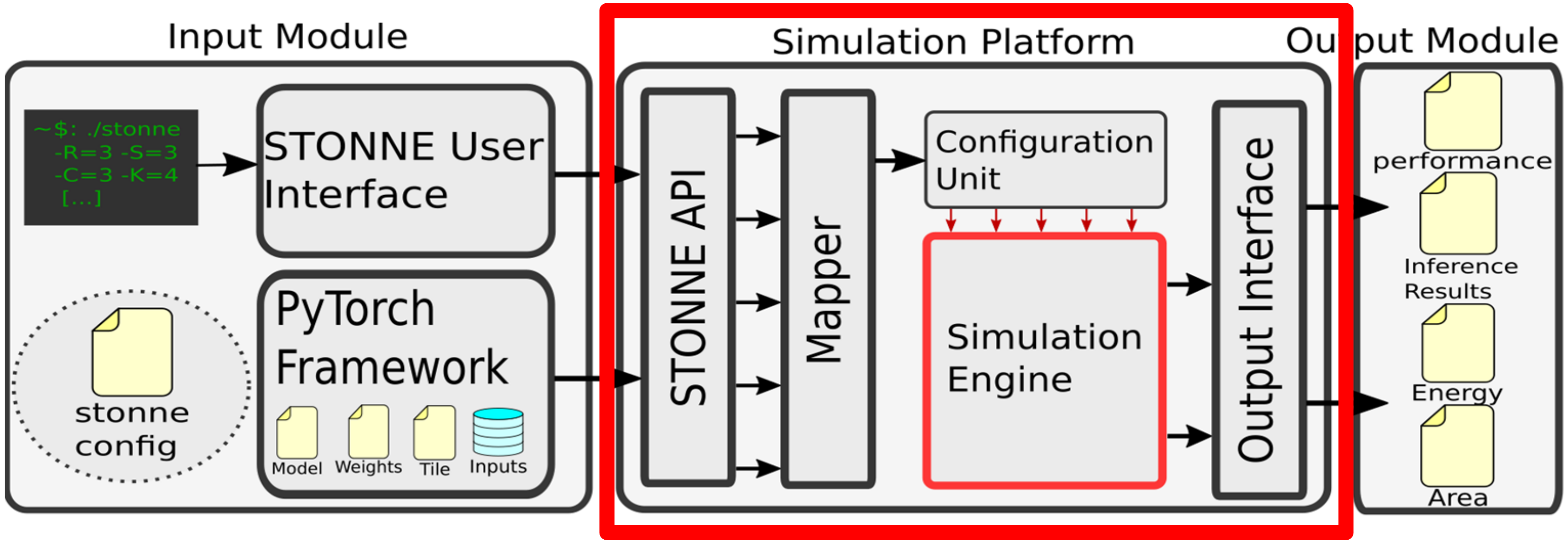
AREA

- This module reports simulation statistics such as performance, compute unit utilization, and activity counts of different components such as wires, FIFOs or SRAM usage.



Simulation Platform

- STONNE is a cycle-level microarchitectural simulator written in C++ for DNN inference accelerators.
- Open-Sourced under the MIT License:
  - https://github.com/stonne-simulator/stonne
- STONNE is constituted by 3 main building blocks:

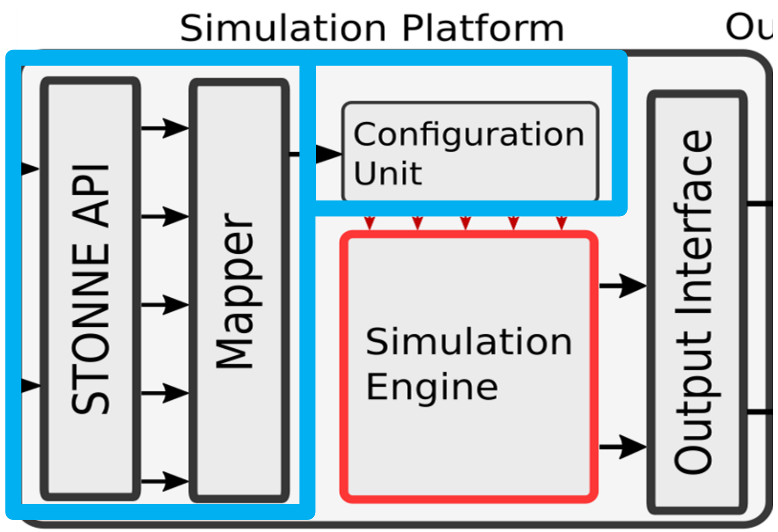


STONNE API, Mapper and Configuration Unit

```
x = self.conv(x) #simulated
x = self.max(x) #simulated
x = self.fc(x) #simulated
x = F.simulated_sparse_mm(x,y,conf='stonne_hw.cfg')#simulated
x = F.log_softmax(x, dim=1) #on cpu
```

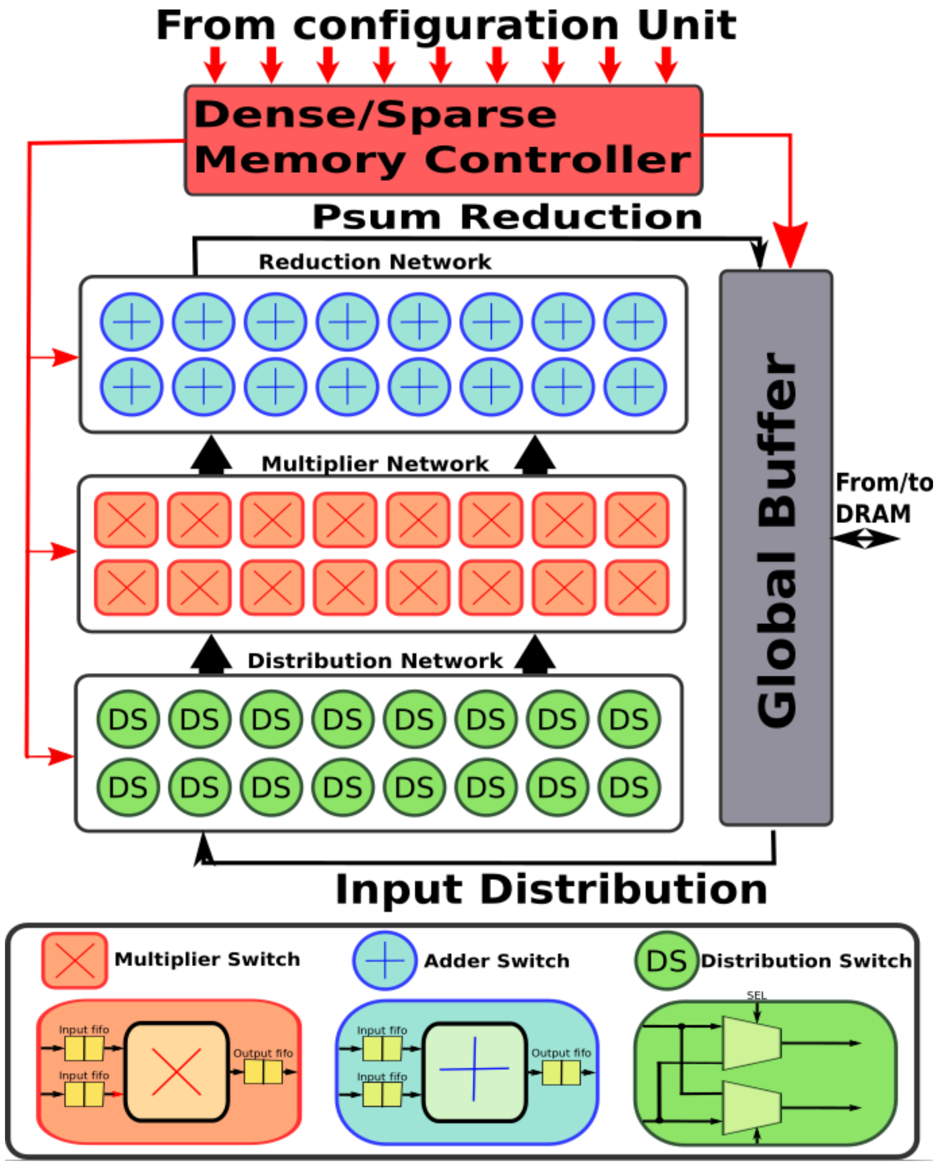
Instruction	Description
CreateInstance	Creates an instance of STONNE
ConfigureCONV	Configures the accelerator to run a *convolution operation*
ConfigureLinear	Configures the accelerator to run a *fully-connected layer*
ConfigureDMM	Configures the accelerator to run a *matrix multiplication*
ConfigureSpMM	Configures the accelerator to run a *sparse matrix multipl.*
ConfigureMaxPool	Configures the accelerator to run a *max pooling layer*
ConfigureData	Configures *weights*, *input* and *outputs addresses* from the CPU to the accelerator memory
RunOperation	*Launches the simulation* according to the current configuration of the architecture

- The **Mapper** generates a set of signals based on the tile and layer parameters (i.e., it determines the required signals to configure the virtual neurons).
- The **Configuration Unit** delivers the signals to the simulation engine, configuring the hardware.

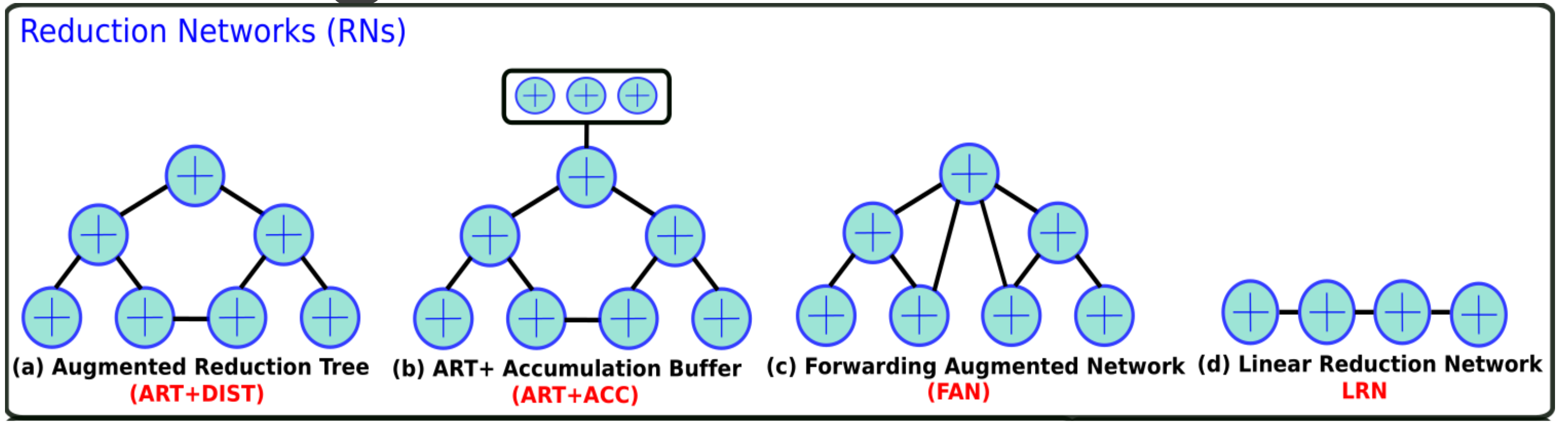


Simulation Engine

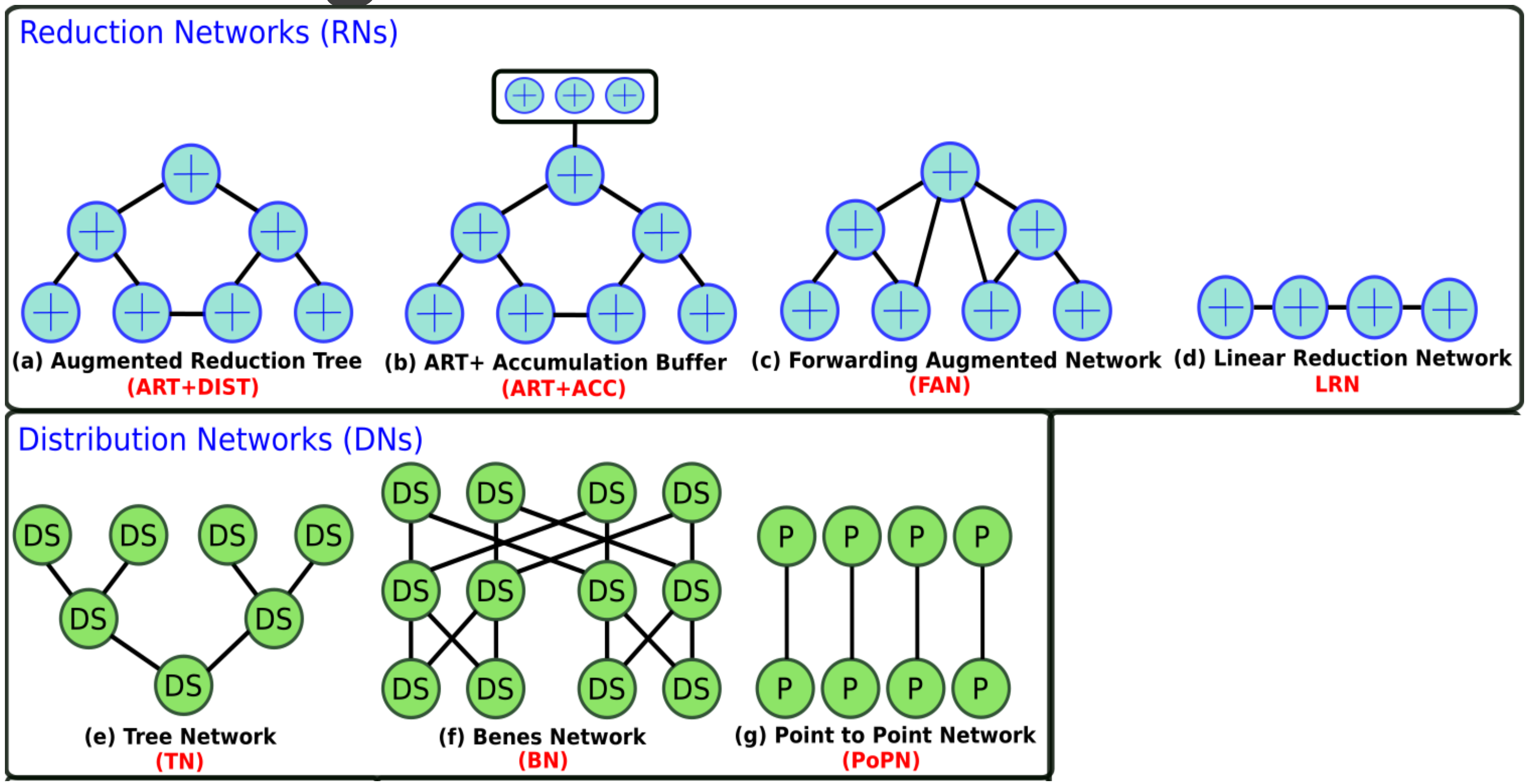
- Most current DNN accelerator architectures can be logically organized as three configurable network fabrics:
  - **Distribution Network**
  - **Multiplier Network**
  - **Reduction Network**



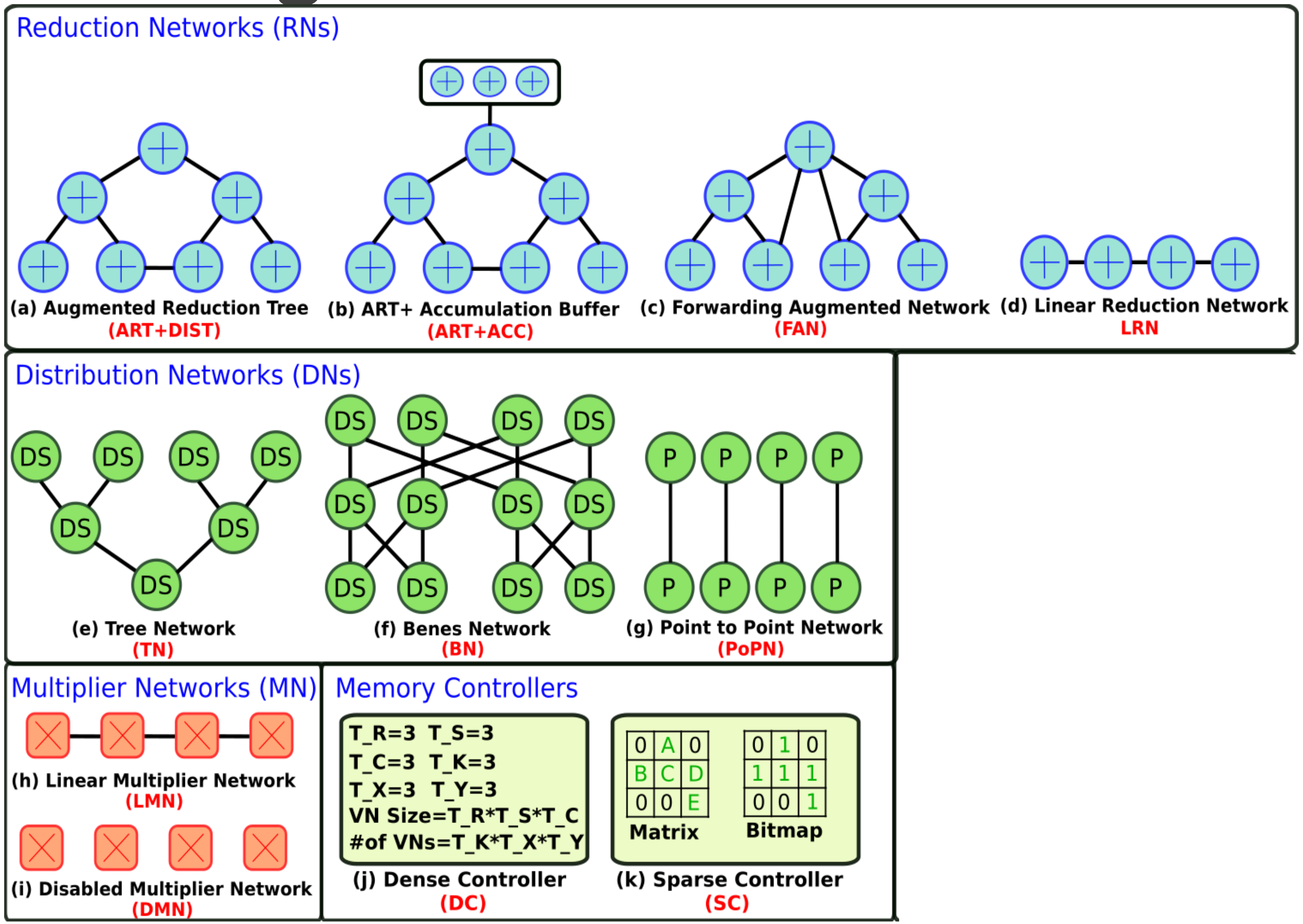
Simulation Engine



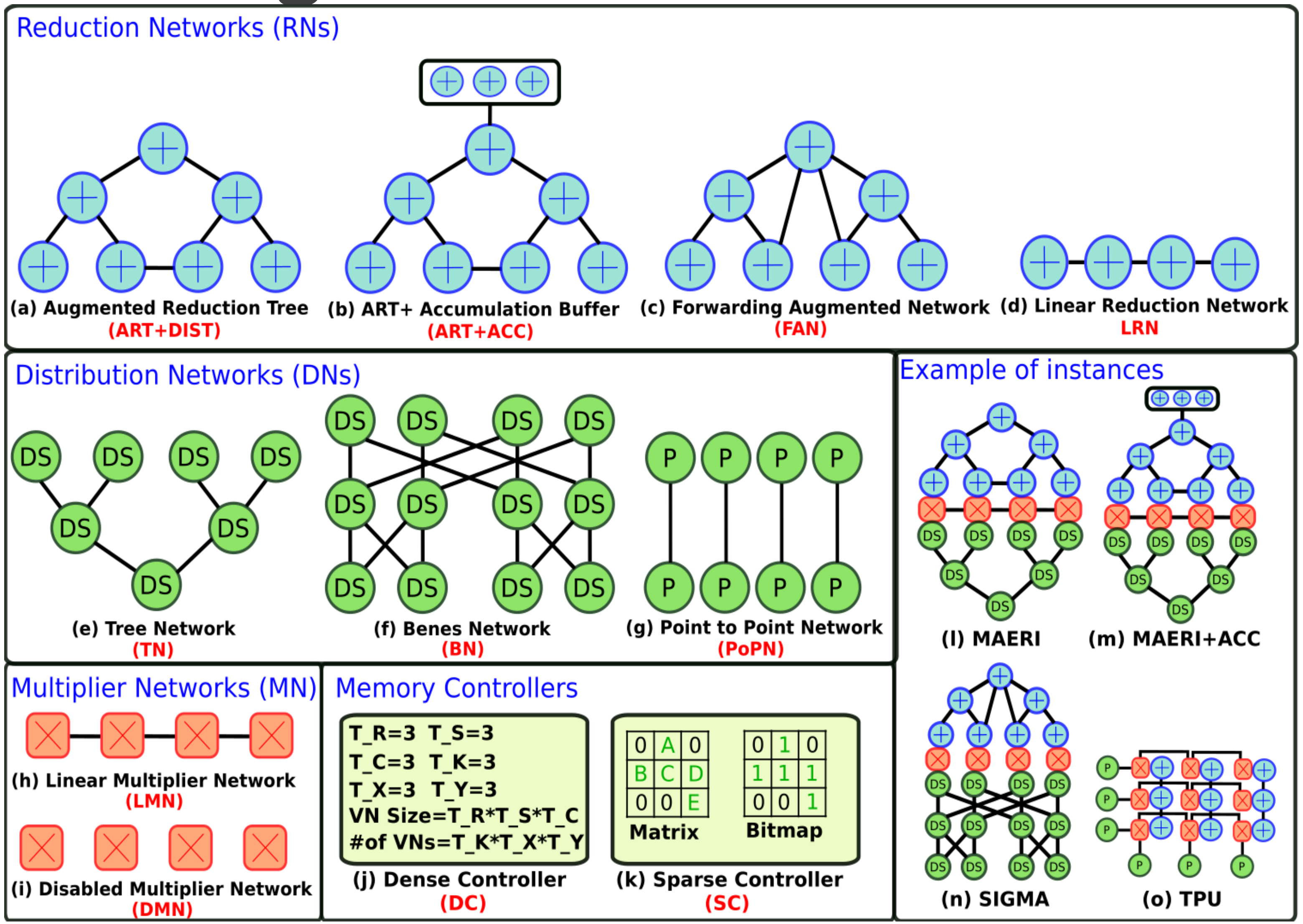
Simulation Engine



Simulation Engine



Simulation Engine



Outline

- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

Validation

- We have validated three state-of-the-art accelerators (i.e., MAERI, SIGMA and TPU) modeled with STONNE against their RTL implementations.

Design	Layer	M	N	K	RTL # cycles	STONNE # cycles	Error %
MAERI	MAERI-1	6	25	54	1338	1381	3.10%
	MAERI-2	20	25	180	16120	16081	0.24%
	MAERI-3	6	400	54	26178	26581	1.51%
SIGMA	SIGMA-1	64	128	32	2321	2304	0.73%
	SIGMA-2	256	64	64	8594	8448	1.72%
	SIGMA-3	256	128	64	17192	16896	1.75%
	SIGMA-4	128	1	64	139	138	0.72%
TPU	TPU-1	16	16	32	66	67	1.50%
	TPU-2	16	16	16	50	51	2.00%
	TPU-3	32	32	16	200	204	2.00%
	TPU-4	64	64	32	1056	1072	1.50%

- **Max 3.10% of difference in terms of number of cycles (1.5% on average)**

Outline

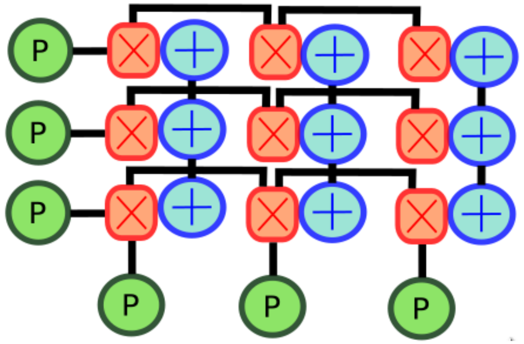
- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio and T. Krishna, "STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators". Proc. of **IISWC 2021**.

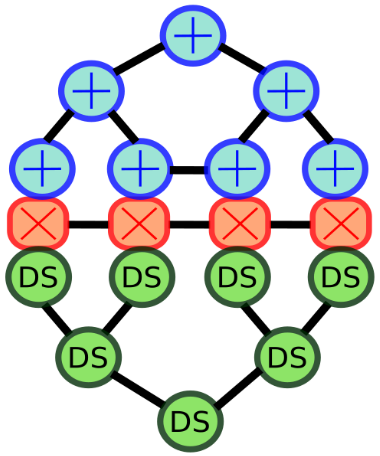
UC#1: DNN inference in TPU, MAERI and SIGMA

Aim: Demonstrate how STONNE can be used to conduct comprehensive evaluations of several DNN accelerators running complete DNN models

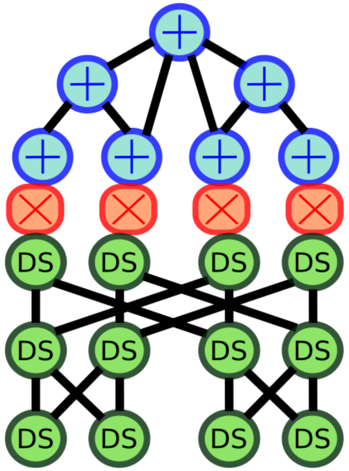
Simulated Accelerators: TPU, MAERI, SIGMA with 256 MSs and 128 elem/cycle BW



TPU-like



MAERI-like



SIGMA-like

	TPU-like	MAERI-like	SIGMA-like
Memory Controller	Dense	Dense	Sparse
Distribution Network	PoPN	TN	BN
Multiplier Network	LMN	LMN	DMN
Reduce Network	LRN	ART	FAN

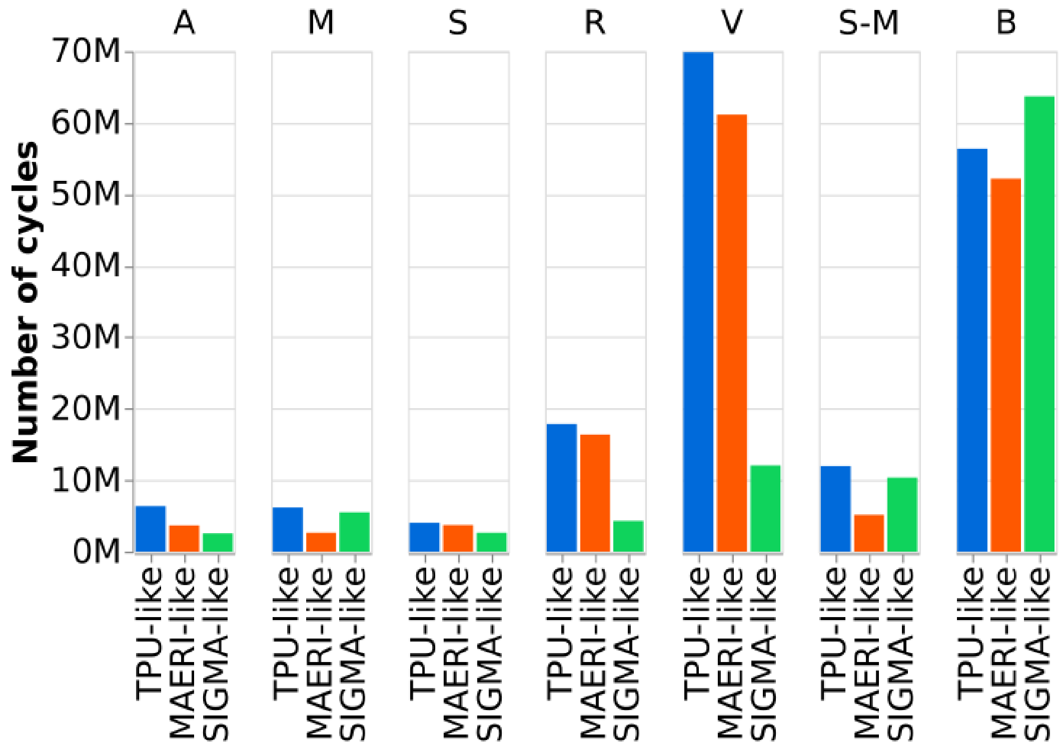
UC#1: DNN inference in TPU, MAERI and SIGMA

Benchmarks:

Domain	DNN Model	Sparsity	Dominant Layer Types
Image Classification	Mobilenets-V1 (M)	75%	Factorized Convolution (FC)
			Linear (L)
	Squeezenet (S)	70%	Squeeze Convolution (SC)
			Expand Convolution (EC)
	Alexnet (A)	78%	Convolution (C)
			Linear (L)
	Resnets-50 (R)	89%	Residual Function (RF)
			Convolution (C)
	VGG-16 (V)	90%	Convolution (C)
			Linear (L)
Object Detection	SSD-Mobilenets (S-M)	75%	Factorized Convolution (FC)
			Linear (L)
Language Processing	BERT (B)	60%	Transformer (TR)
			Linear (L)

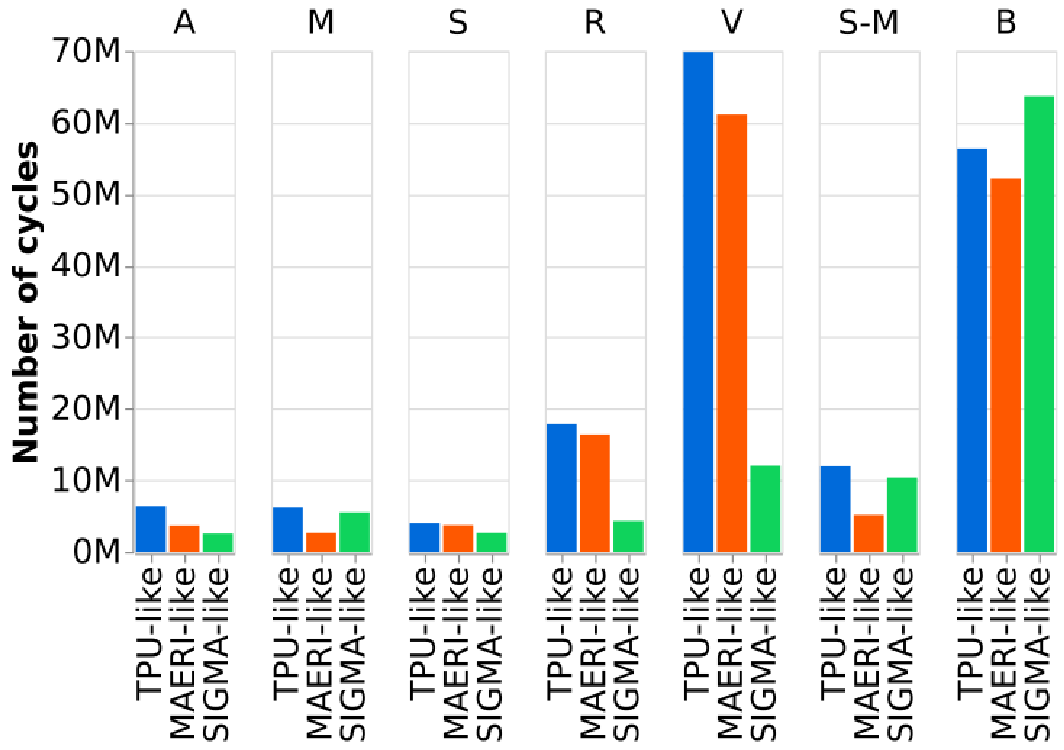
UC#1: DNN inference in TPU, MAERI and SIGMA

Results: Number of cycles



UC#1: DNN inference in TPU, MAERI and SIGMA

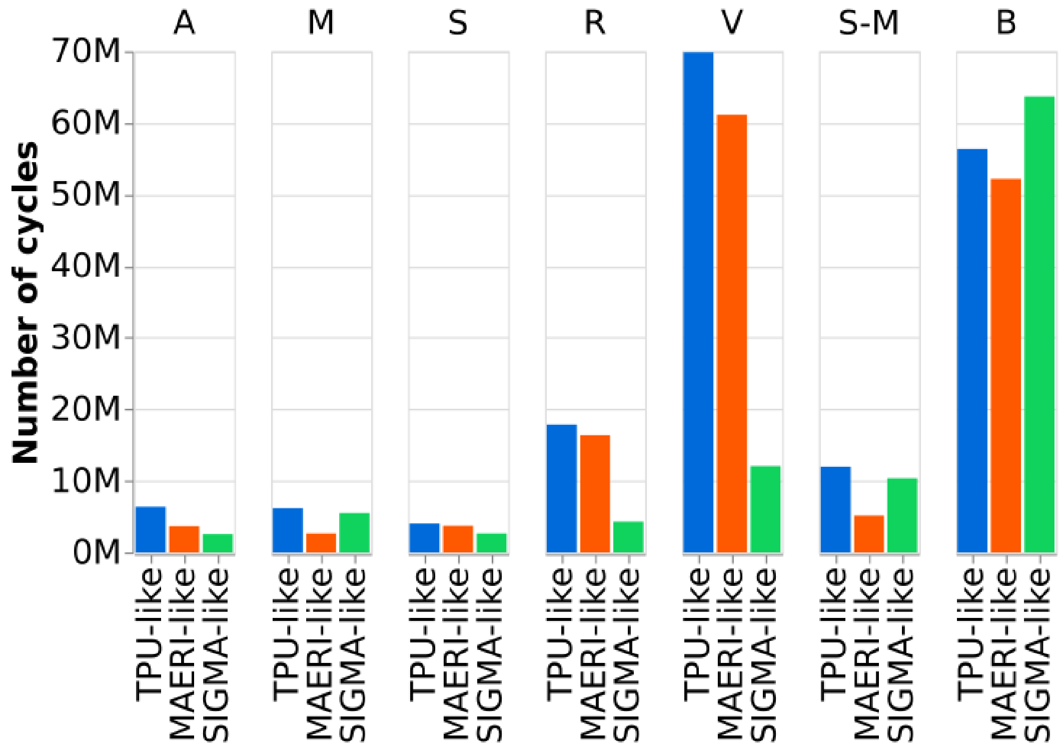
Results: Number of cycles



MAERI-like architecture 20% faster than the TPU-like architecture

UC#1: DNN inference in TPU, MAERI and SIGMA

Results: Number of cycles

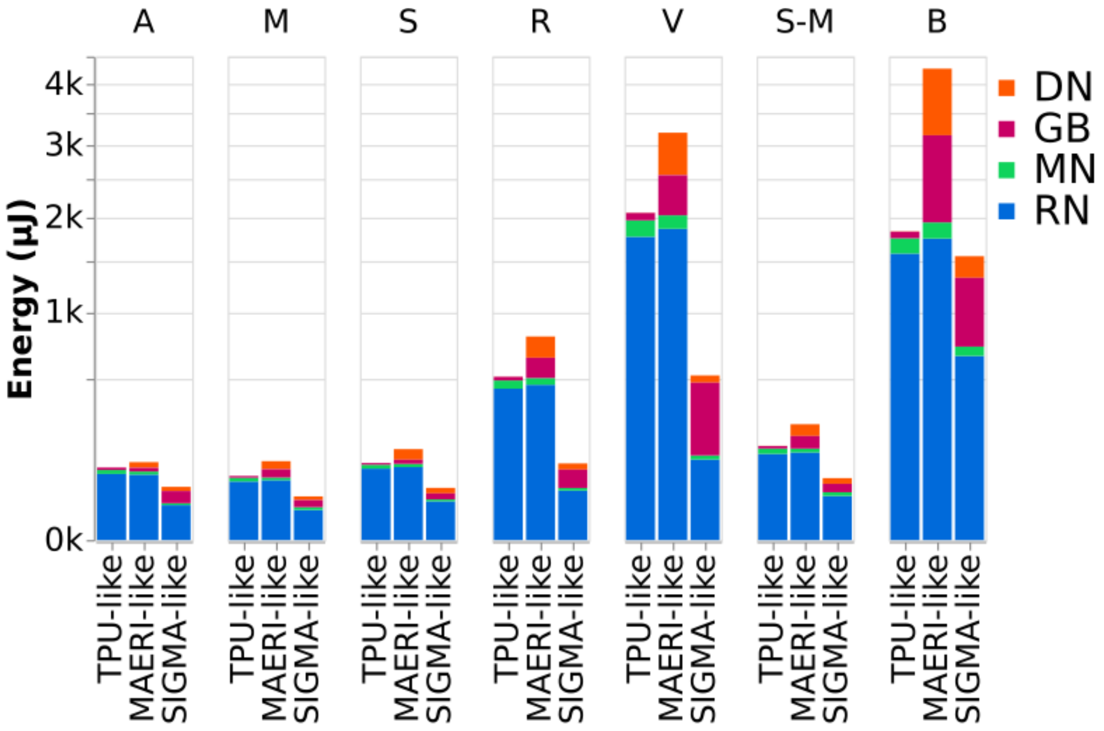


MAERI-like architecture 20% faster than the TPU-like architecture

SIGMA-like architecture is 91% faster than a MAERI-like one

UC#1: DNN inference in TPU, MAERI and SIGMA

Results: Energy

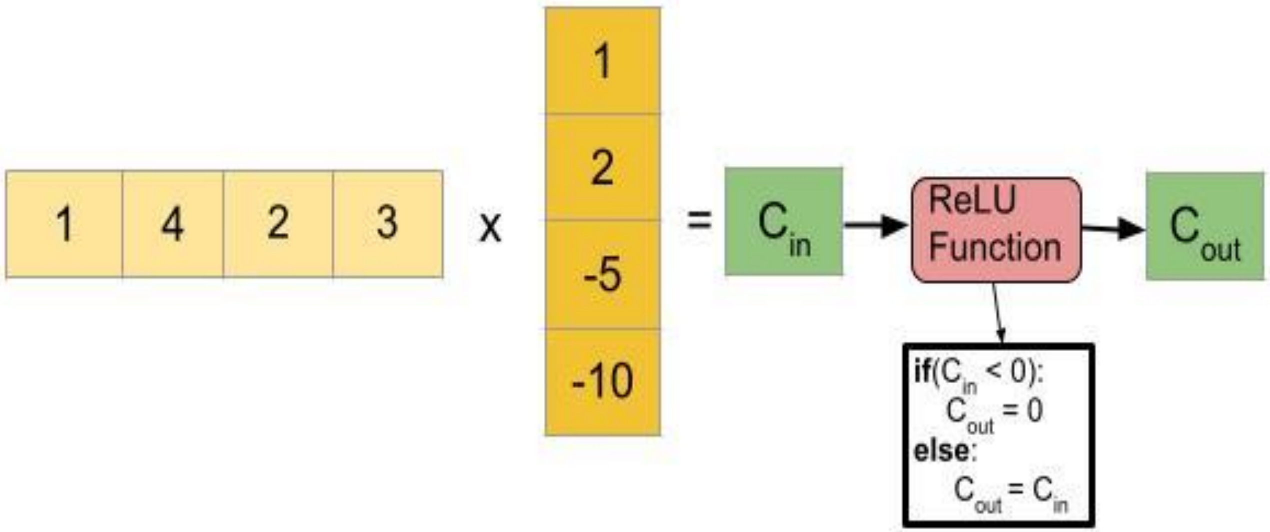


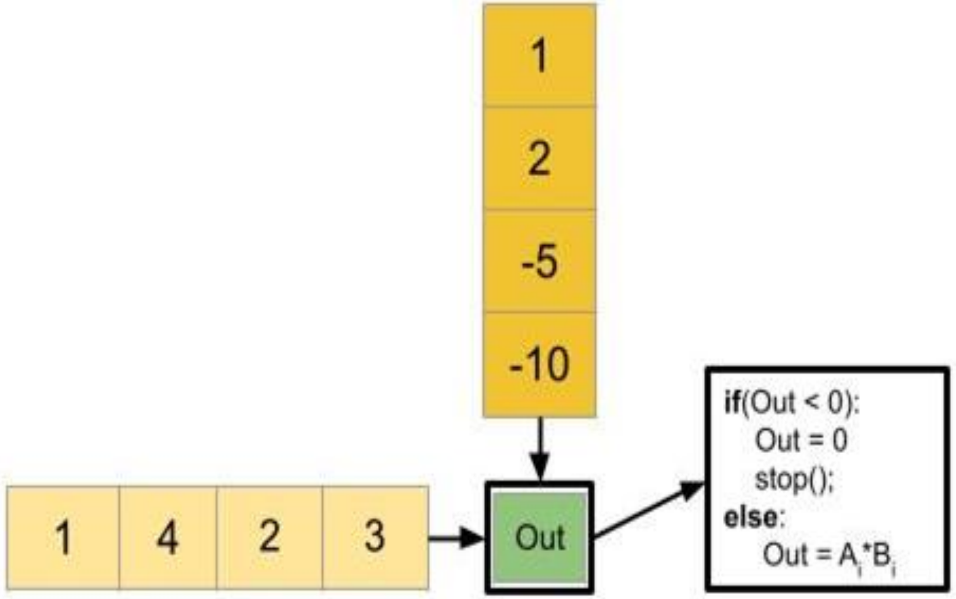
SIGMA-like architecture is 70% and 54% more energy efficient than the MAERI-like and TPU-like architectures

UC#2: Data-Dependent HW Optimizations

Aim: Prove how the **back-end of STONNE** can be easily extended to model data-dependent accelerators

Simulated Accelerator: SnaPea

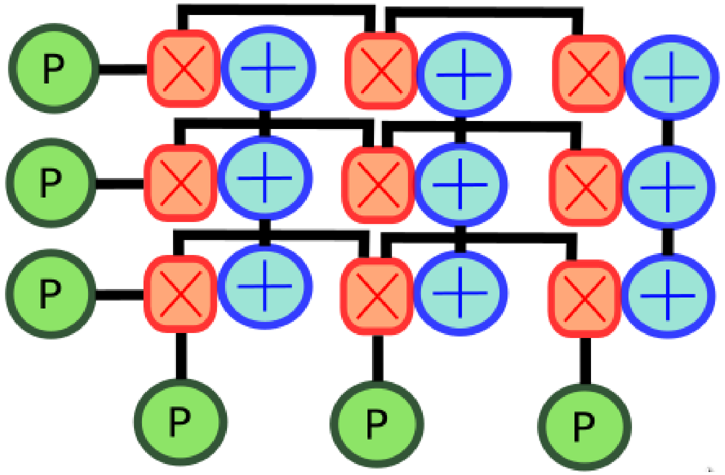




1x1 SnaPea Architecture

UC#2: Data-Dependent HW Optimizations

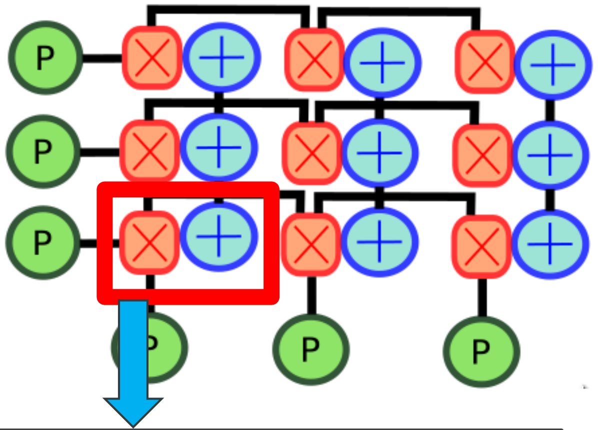
Implementation:



TPU-like

UC#2: Data-Dependent HW Optimizations

Implementation:

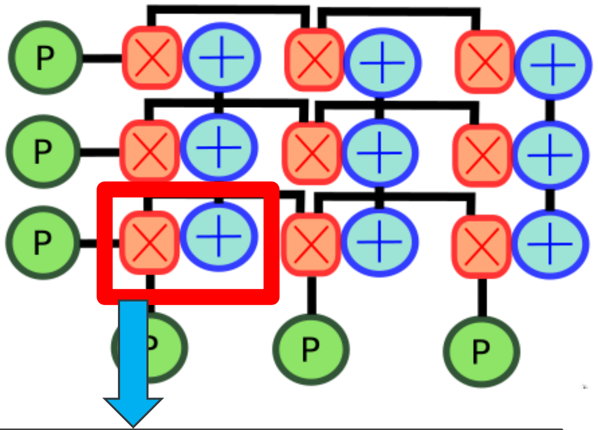


Negative detection Logic

```
if(current_output < 0):
    result = 0;
else:
    continue();
```

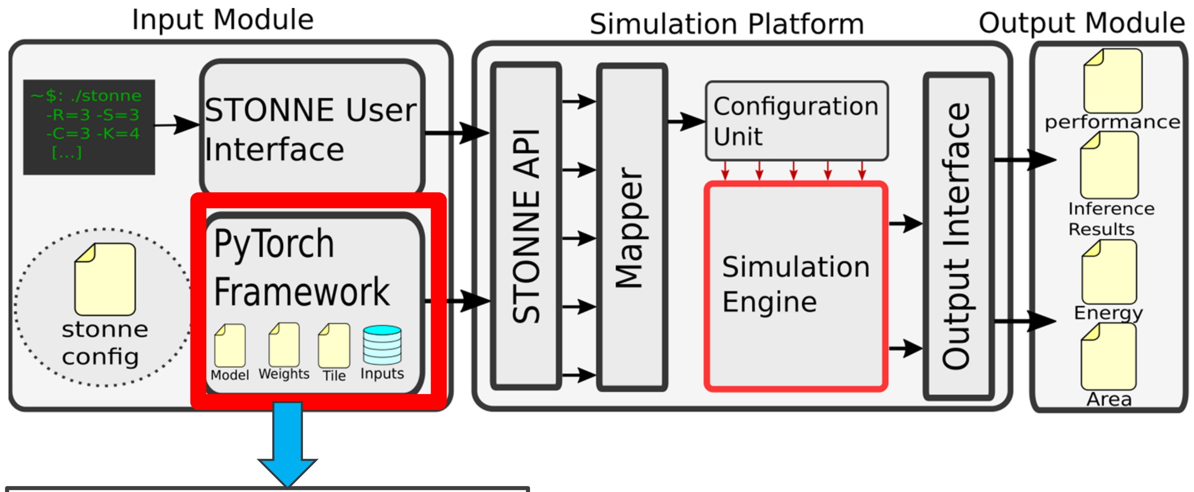
UC#2: Data-Dependent HW Optimizations

Implementation:



Negative detection Logic

```
if(current_output < 0):
    result = 0;
else:
    continue();
```



Order weights

```
order_weights(weights)
off_load_simulation();
```

UC#2: Data-Dependent HW Optimizations

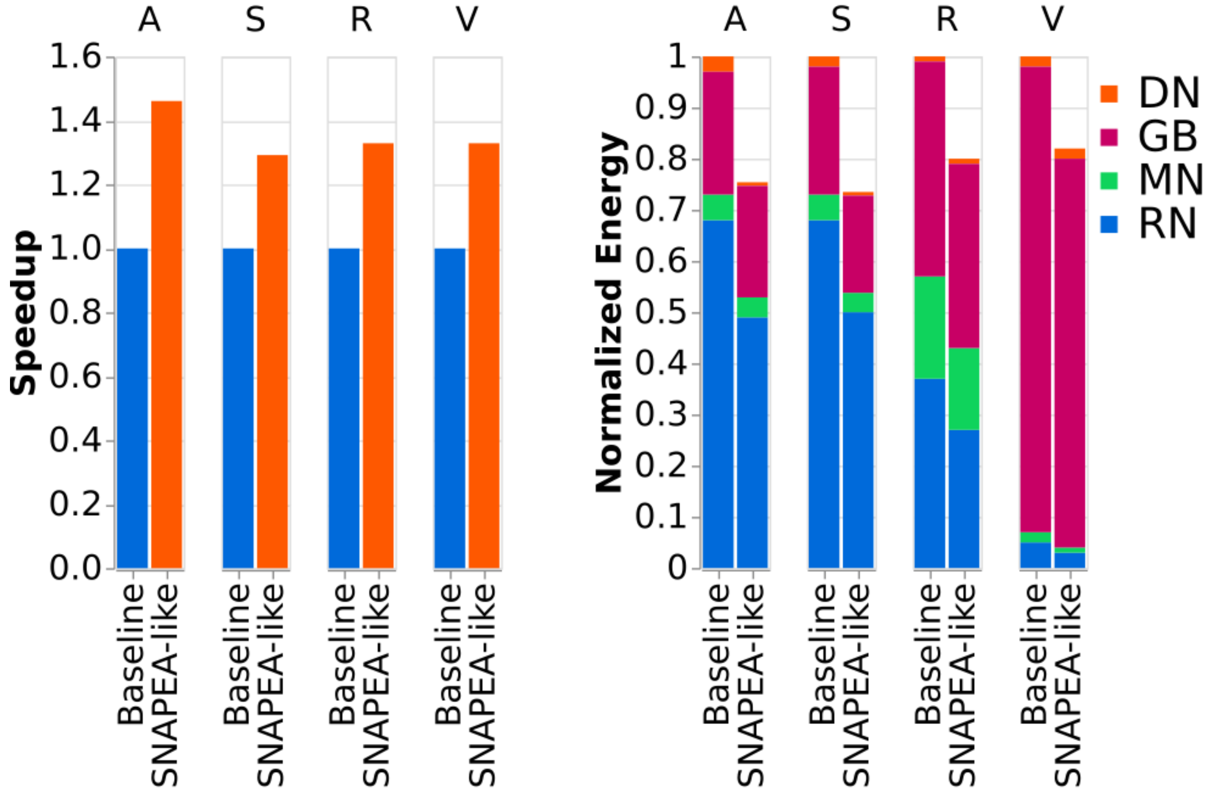
Parameters: 64 MSs and ASs, and 64 elements/cycle GB read/write BW

Benchmarks: Pure CNN models

Domain	DNN Model	Sparsity	Dominant Layer Types
Image Classification	Mobilenets-V1 (M) [15]	75%	Factorized Convolution (FC)
			Linear (L)
	Squeezenet (S) [16]	70%	Squeeze Convolution (SC)
			Expand Convolution (EC)
	Alexnet (A) [17]	78%	Convolution (C)
			Linear (L)
	Resnets-50 (R) [18]	89%	Residual Function (RF)
			Convolution (C)
	VGG-16 (V) [19]	90%	Convolution (C)
			Linear (L)
Object Detection	SSD-Mobilenets (S-M) [20]	75%	Factorized Convolution (FC)
			Linear (L)
Language Processing	BERT (B) [21]	60%	Transformer (TR)
			Linear (L)

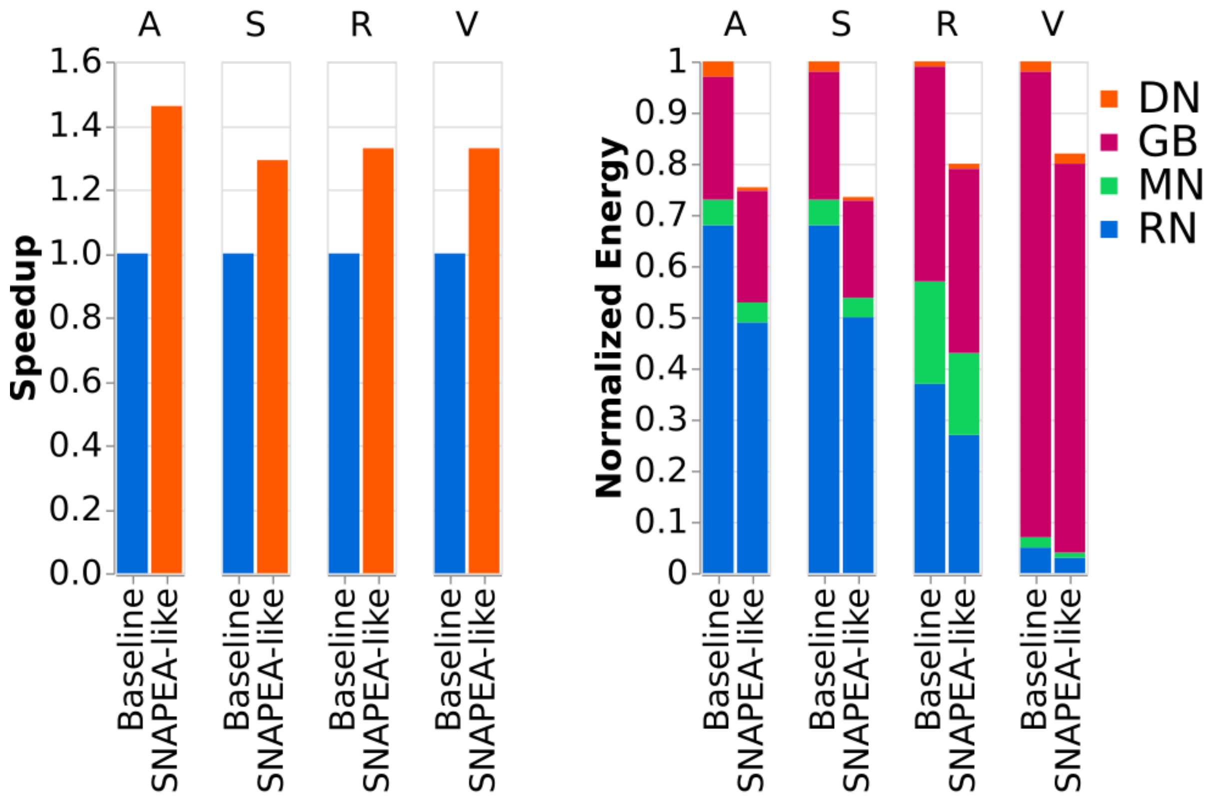
UC#2: Data-Dependent HW Optimizations

Results:



UC#2: Data-Dependent HW Optimizations

Results:

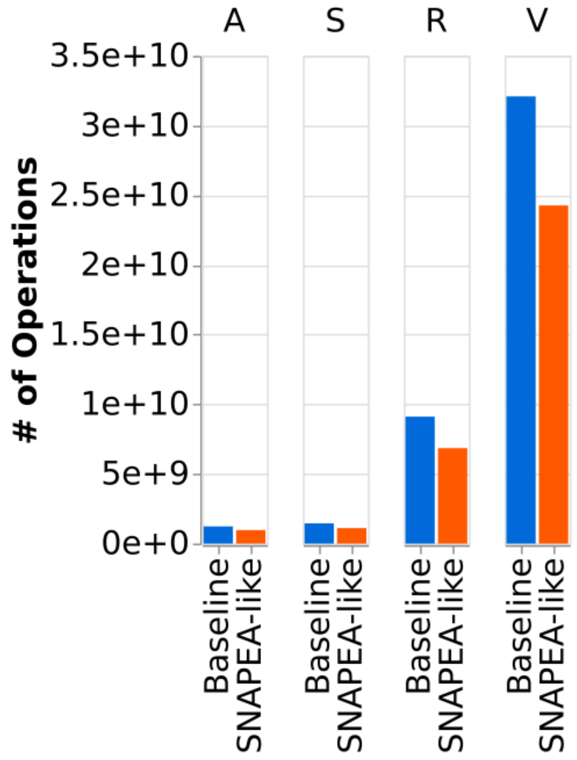


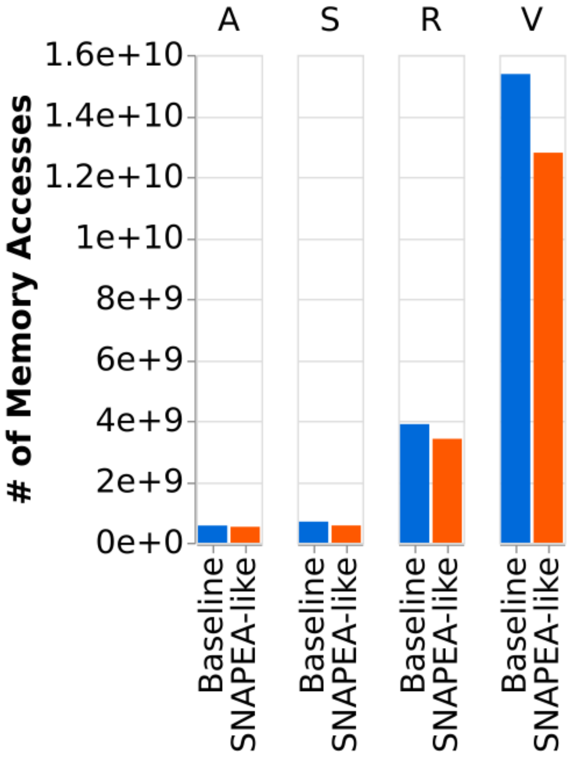
Average SpeedUp of 35%

Average Energy Savings of 21%

UC#2: Data-Dependent HW Optimizations

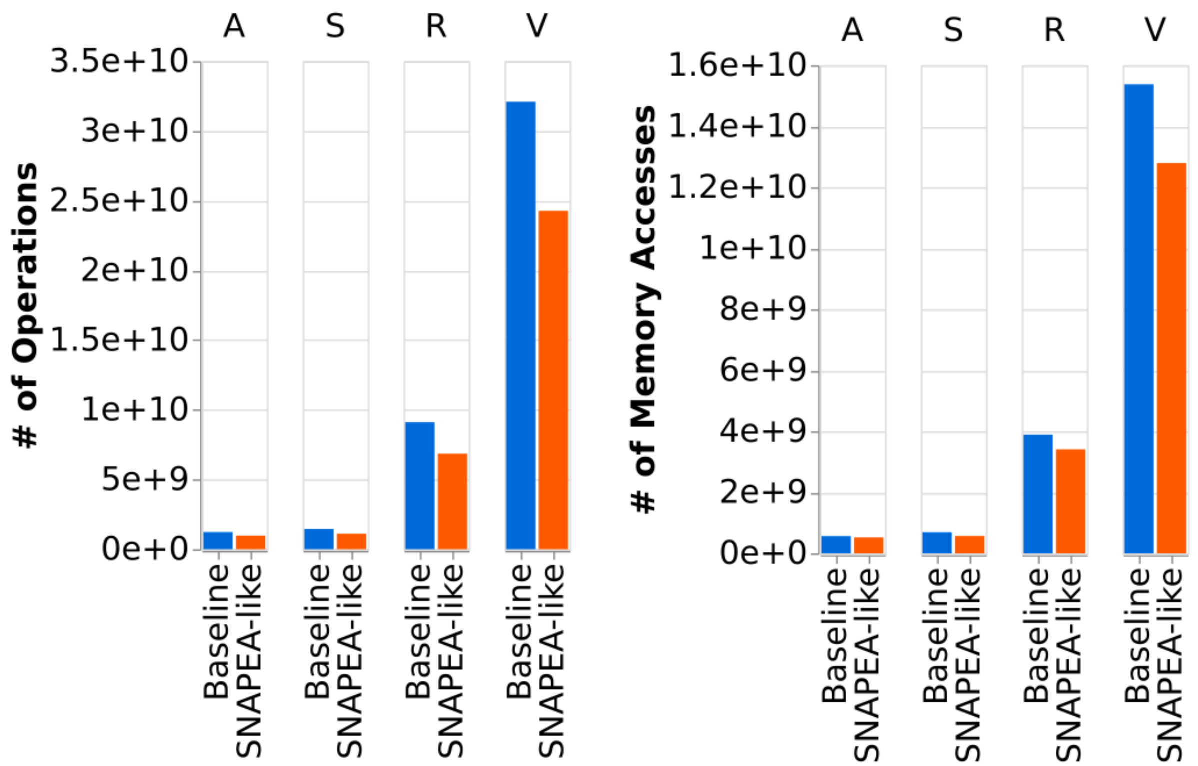
Results:





UC#2: Data-Dependent HW Optimizations

Results:



30% reduction of # of operations

16 % reduction of # of Memory Accesses

UC#3: Filter Scheduling in Sparse Accelerators

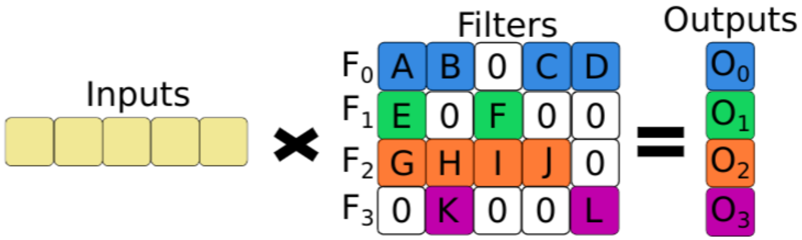
Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance

UC#3: Filter Scheduling in Sparse Accelerators

Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

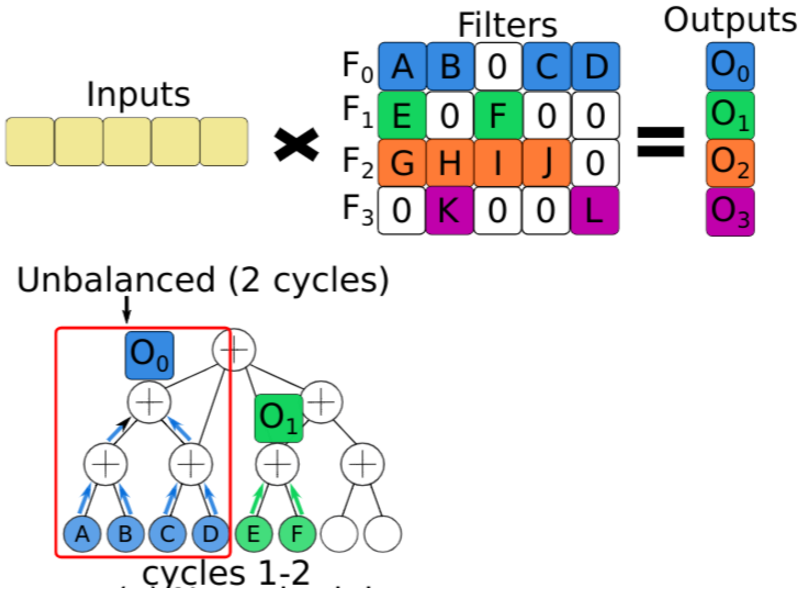
Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance



UC#3: Filter Scheduling in Sparse Accelerators

Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

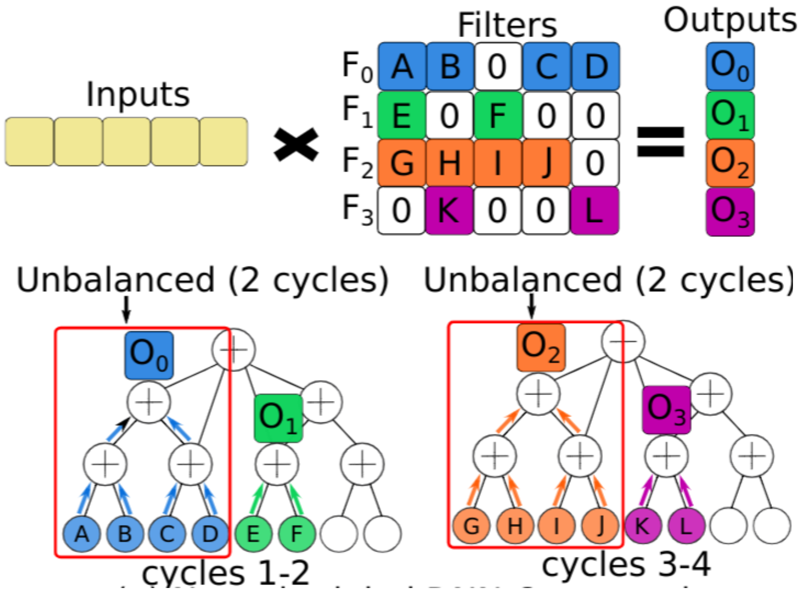
Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance



UC#3: Filter Scheduling in Sparse Accelerators

Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

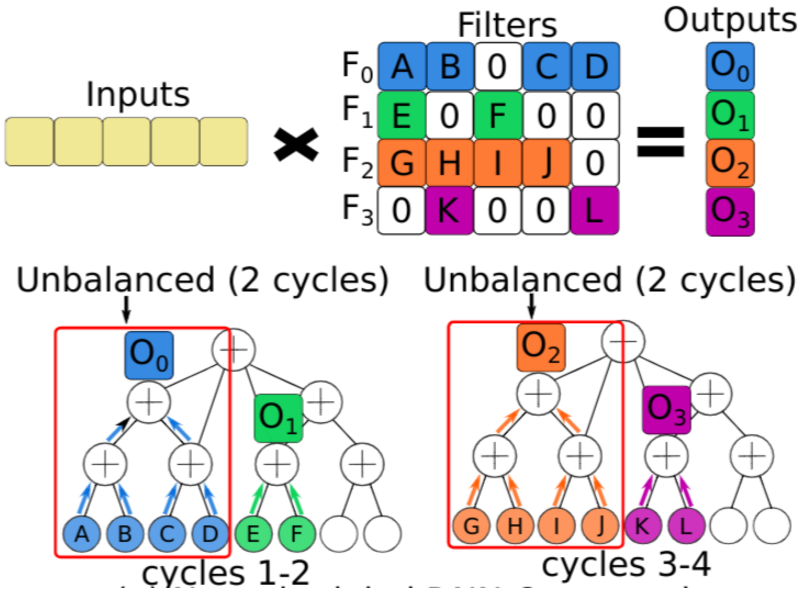
Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance

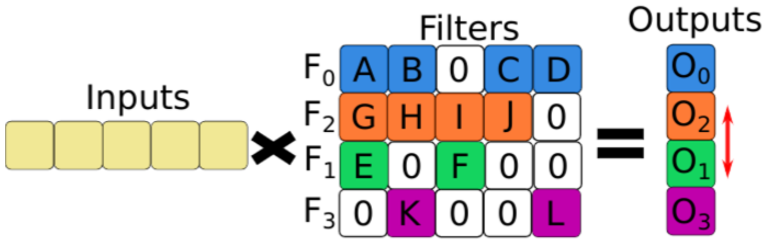


UC#3: Filter Scheduling in Sparse Accelerators

Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance

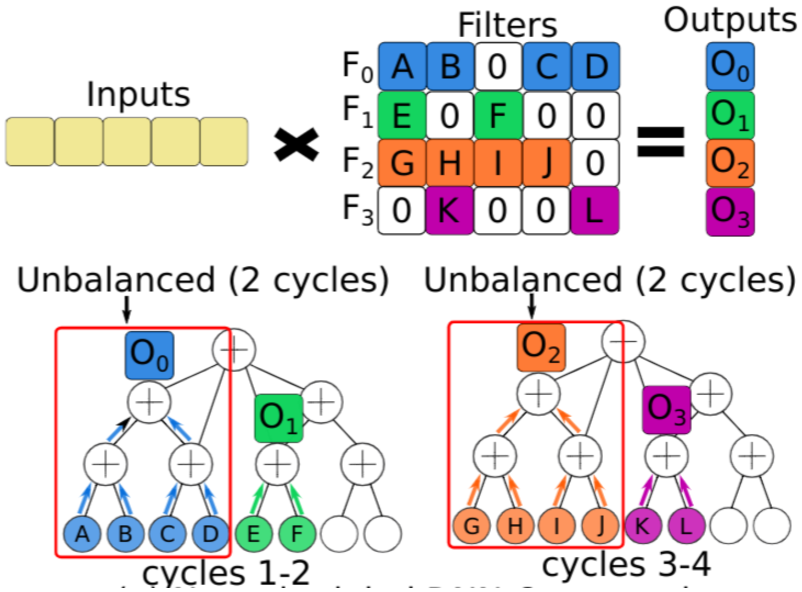


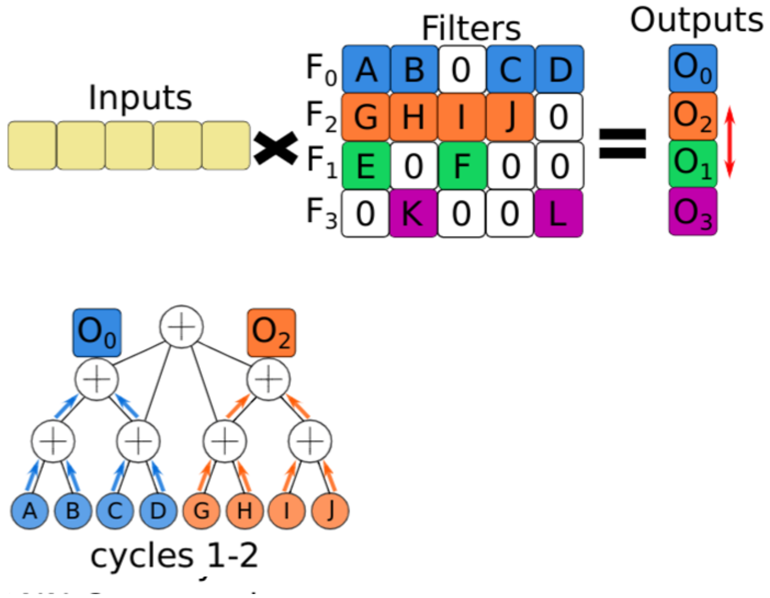


UC#3: Filter Scheduling in Sparse Accelerators

Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance

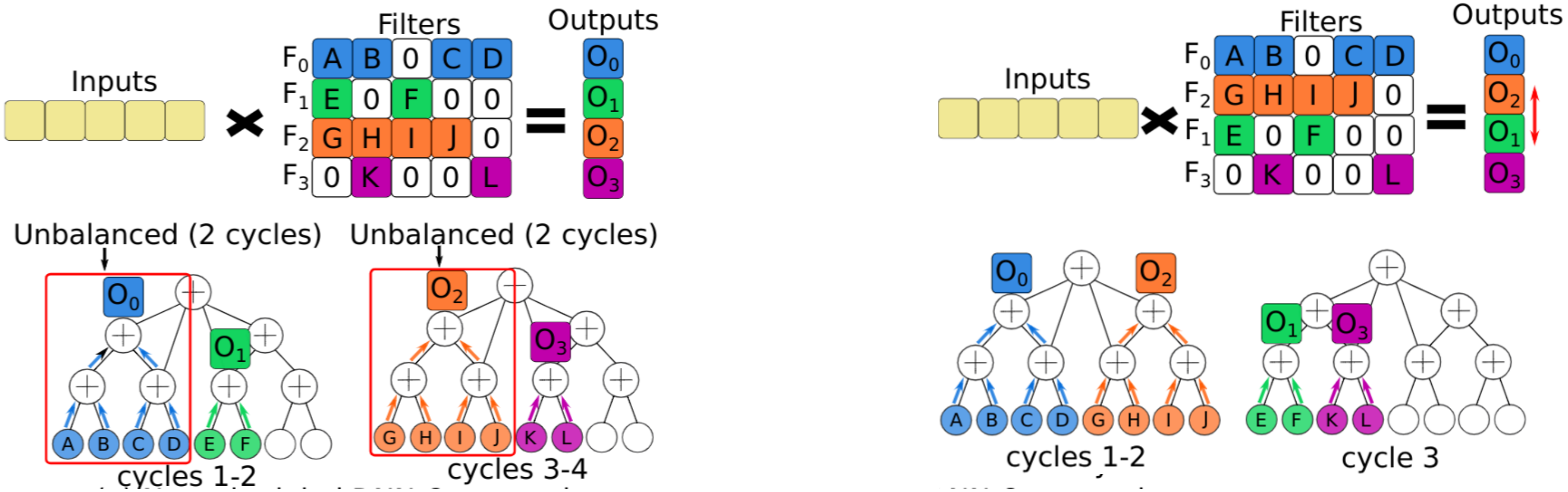




UC#3: Filter Scheduling in Sparse Accelerators

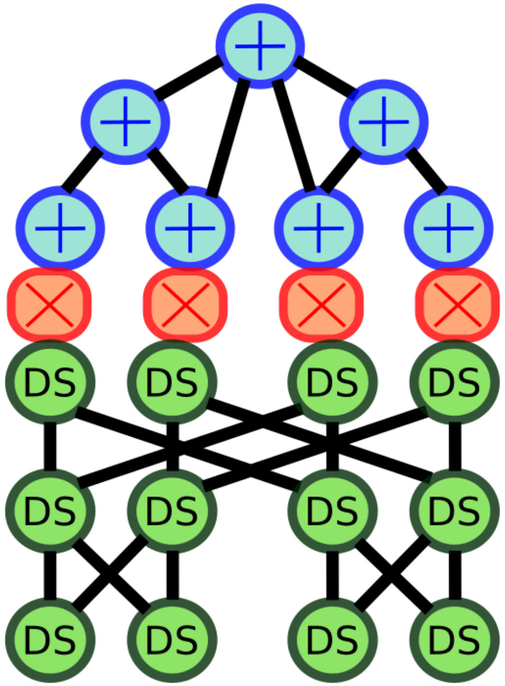
Aim: Demonstrate that precise, full-model evaluation is required to expose the particular values used during inference

Motivation and Idea: The way in which the filters of a sparse DNN model are scheduled onto a DNN inference accelerator might have significant impact on performance



UC#3: Filter Scheduling in Sparse Accelerators

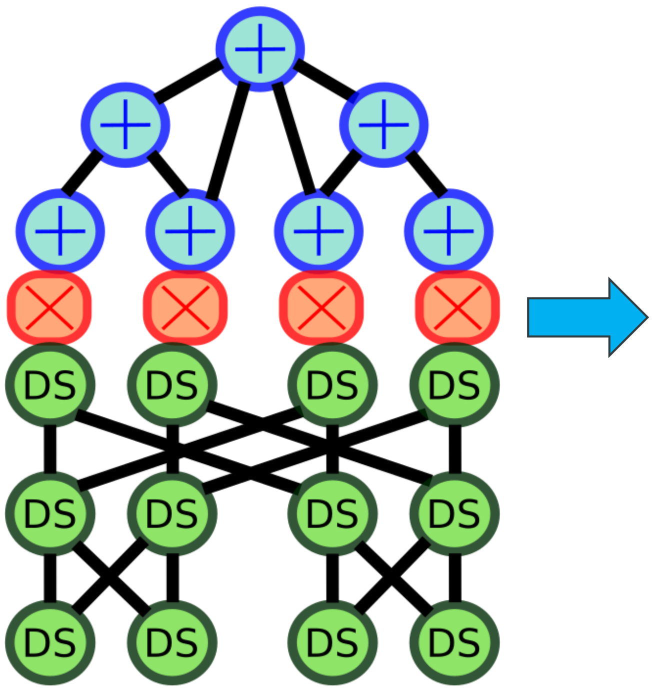
Implementation:



SIGMA-like

UC#3: Filter Scheduling in Sparse Accelerators

Implementation:

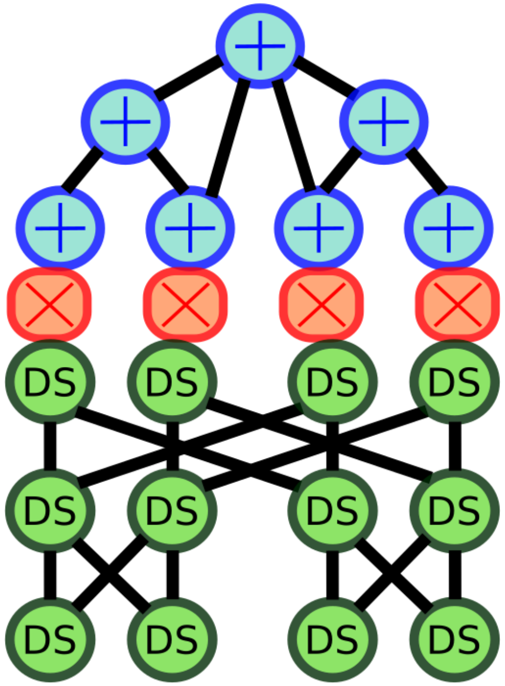


SIGMA-like

We do not require modifications in the accelerator

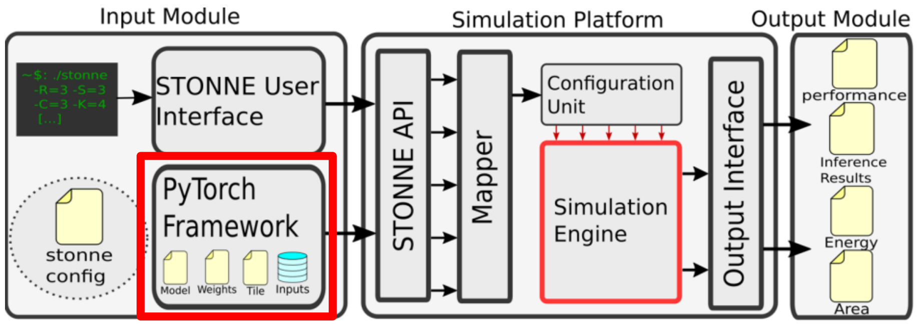
UC#3: Filter Scheduling in Sparse Accelerators

Implementation:



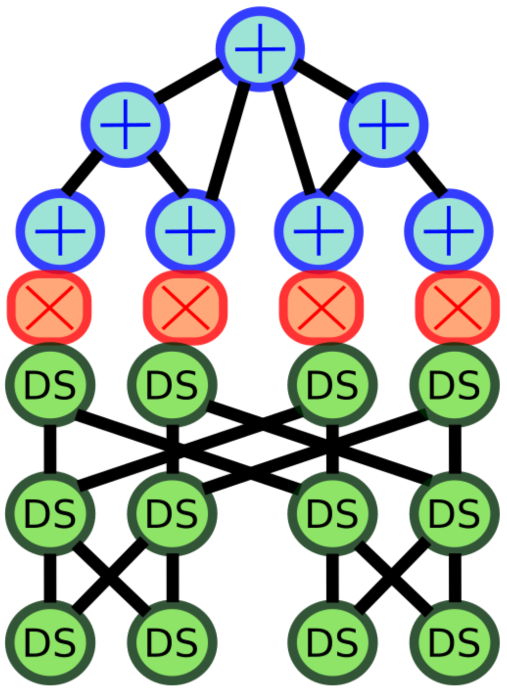
SIGMA-like

We do not require modifications in the accelerator



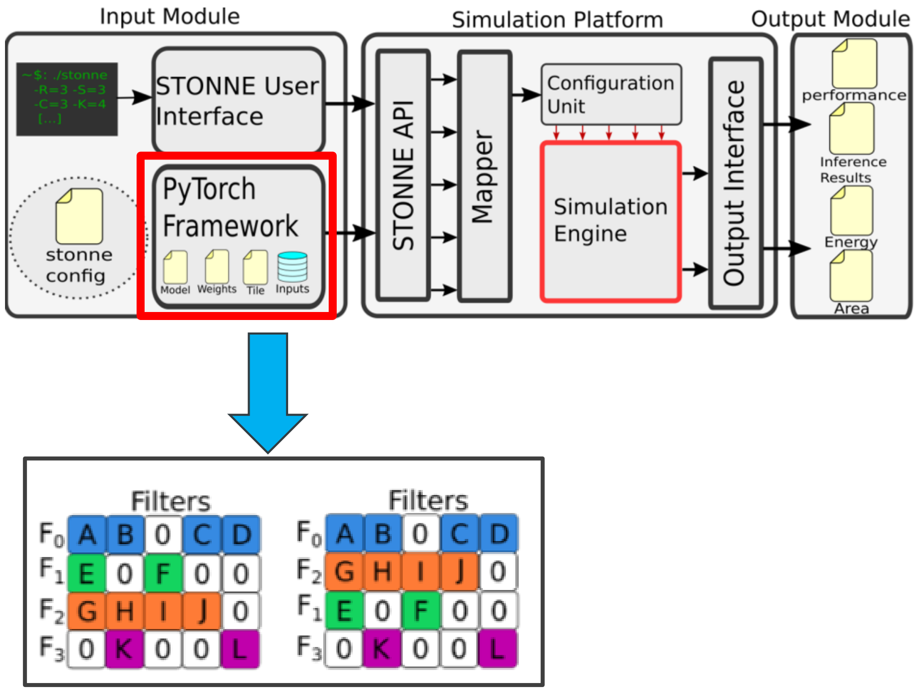
UC#3: Filter Scheduling in Sparse Accelerators

Implementation:



SIGMA-like

We do not require modifications in the accelerator

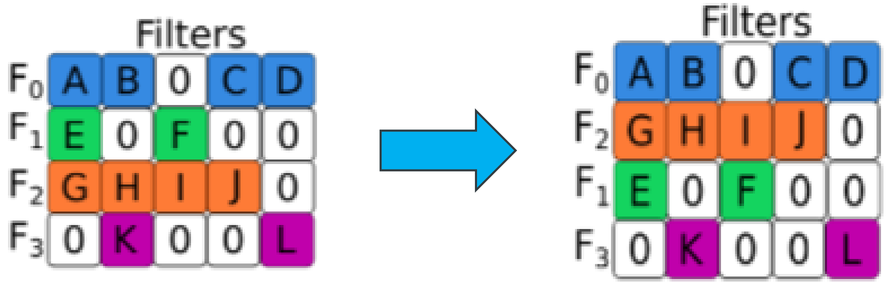


Pre-simulation function to order the filters based on a certain heuristic

UC#3: Filter Scheduling in Sparse Accelerators

Implementation: We implement two heuristic algorithms:

- ***Largest Filter First (LFF)***: The filters are reordered so that the sparse controller always selects the largest available filter.



- ***Random (RDM):*** The filters are reordered randomly.
- ***No Schedule (NS):*** The filters are selected as they are in the model.

UC#3: Filter Scheduling in Sparse Accelerators

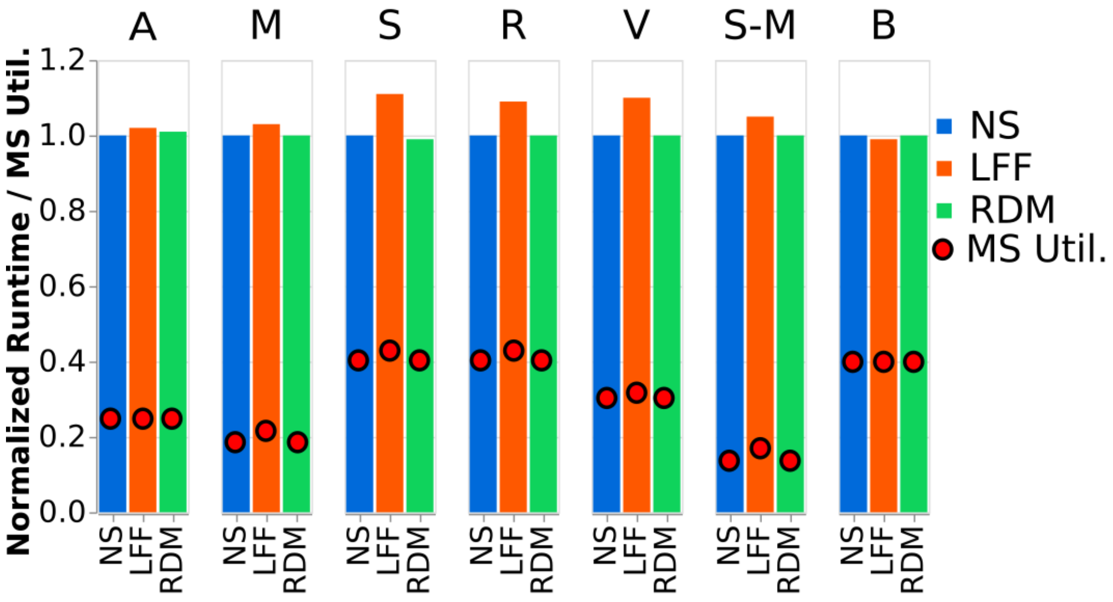
Simulated Parameters: SIGMA-like architecture with 256 multipliers and adders and 128 elements/cycle Global Buffer(GB) read/write bandwidth.

Benchmarks:

Domain	DNN Model	Sparsity	Dominant Layer Types
Image Classification	Mobilenets-V1 (M) [15]	75%	Factorized Convolution (FC)
			Linear (L)
	Squeezenet (S) [16]	70%	Squeeze Convolution (SC)
			Expand Convolution (EC)
	Alexnet (A) [17]	78%	Convolution (C)
			Linear (L)
	Resnets-50 (R) [18]	89%	Residual Function (RF)
			Convolution (C)
	VGG-16 (V) [19]	90%	Convolution (C)
			Linear (L)
Object Detection	SSD-Mobilenets (S-M) [20]	75%	Factorized Convolution (FC)
			Linear (L)
Language Processing	BERT (B) [21]	60%	Transformer (TR)
			Linear (L)

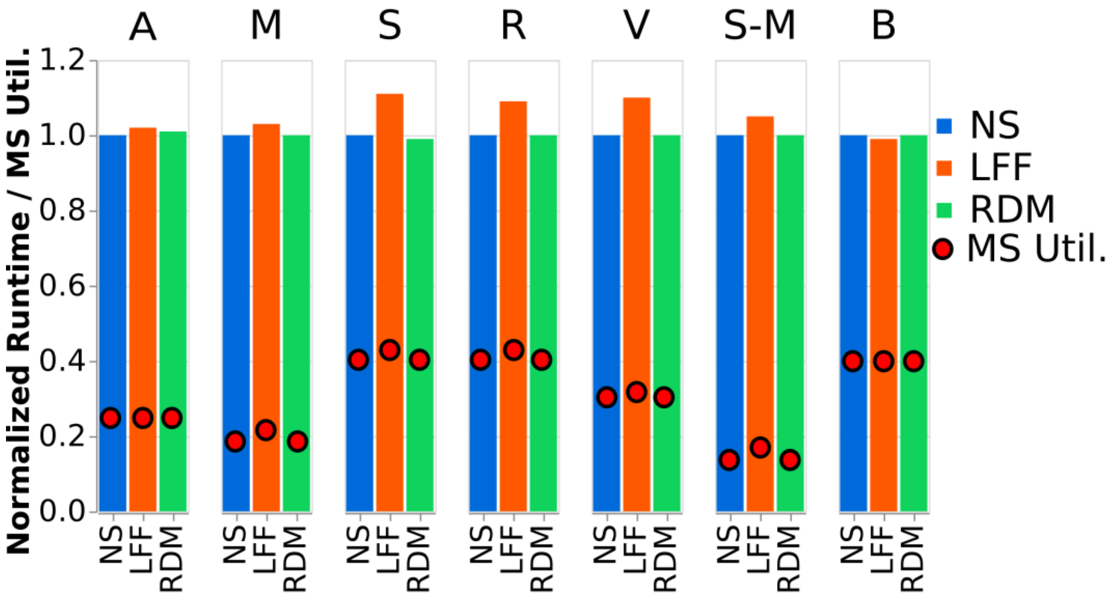
UC#3: Filter Scheduling in Sparse Accelerators

Results:



UC#3: Filter Scheduling in Sparse Accelerators

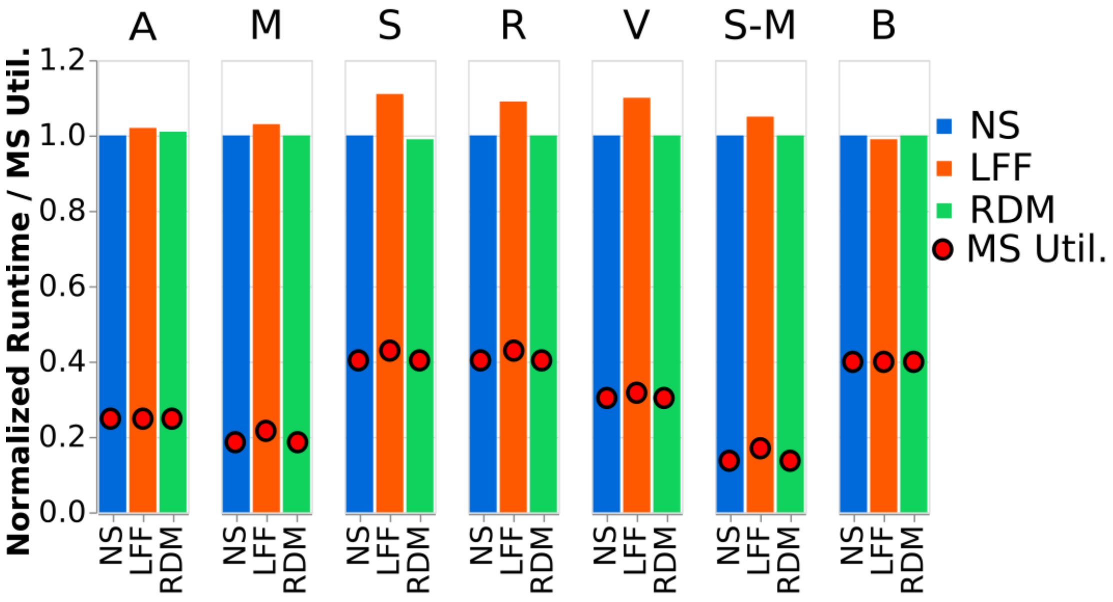
Results:



LFF obtains performance benefits of up to 11% (7% on average).

UC#3: Filter Scheduling in Sparse Accelerators

Results:

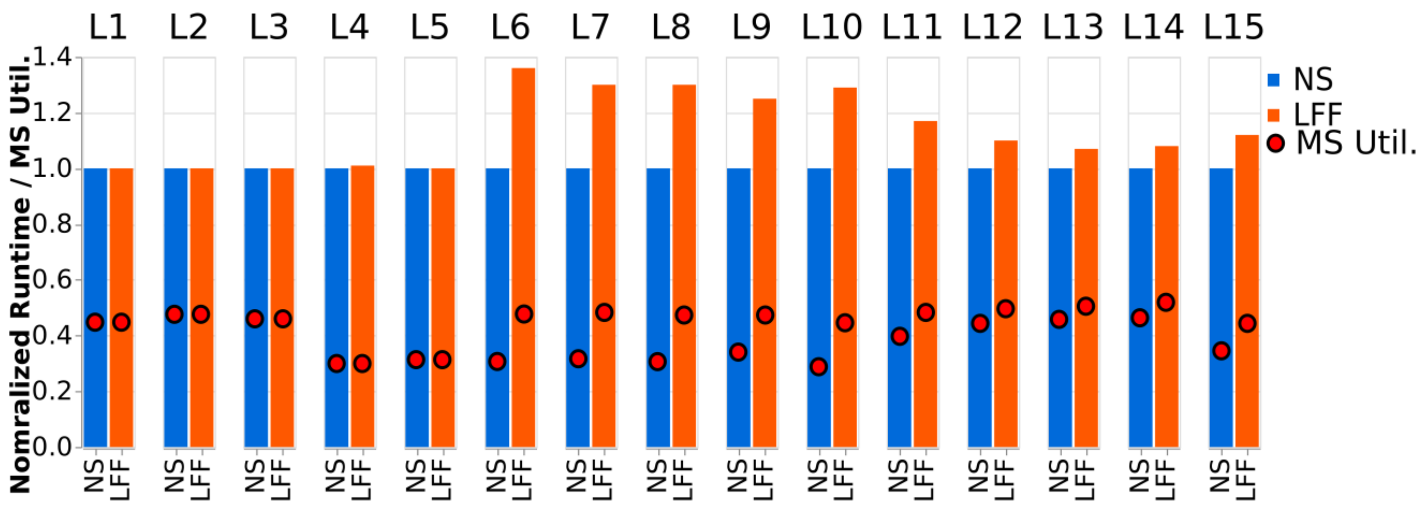


LFF obtains performance benefits of up to 11% (7% on average).

RDM does not translate into better performance.

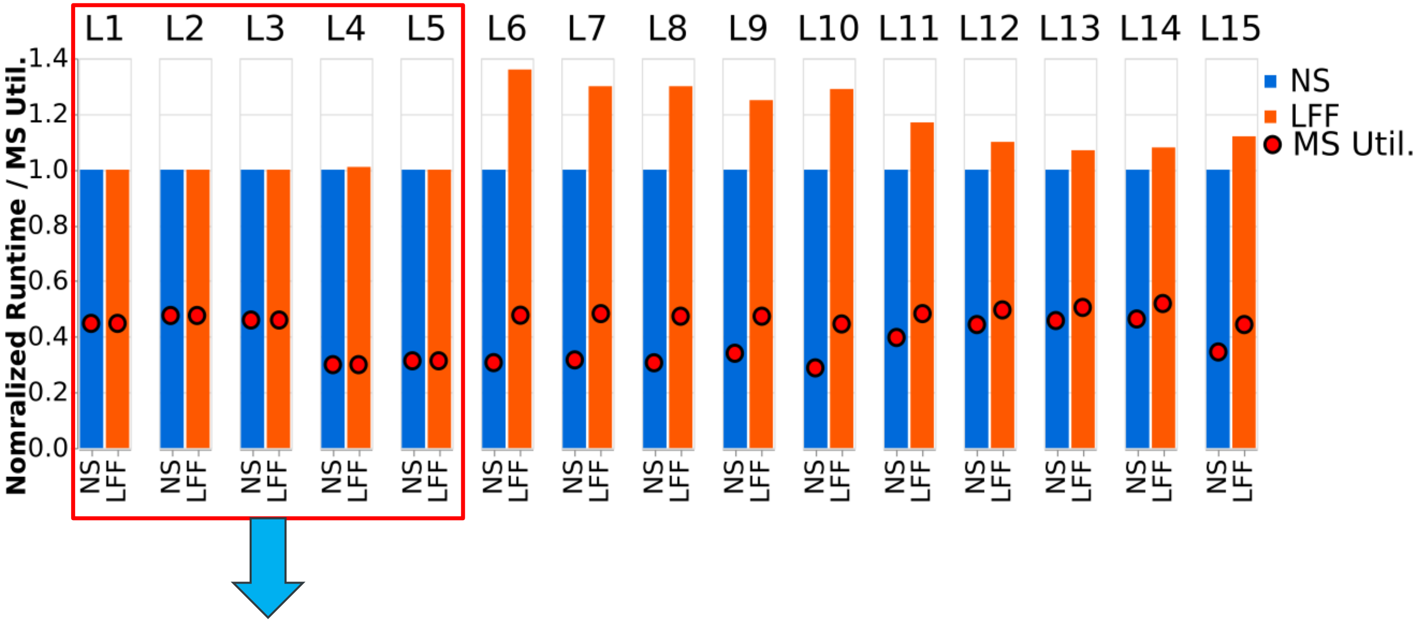
UC#3: Filter Scheduling in Sparse Accelerators

Results:



UC#3: Filter Scheduling in Sparse Accelerators

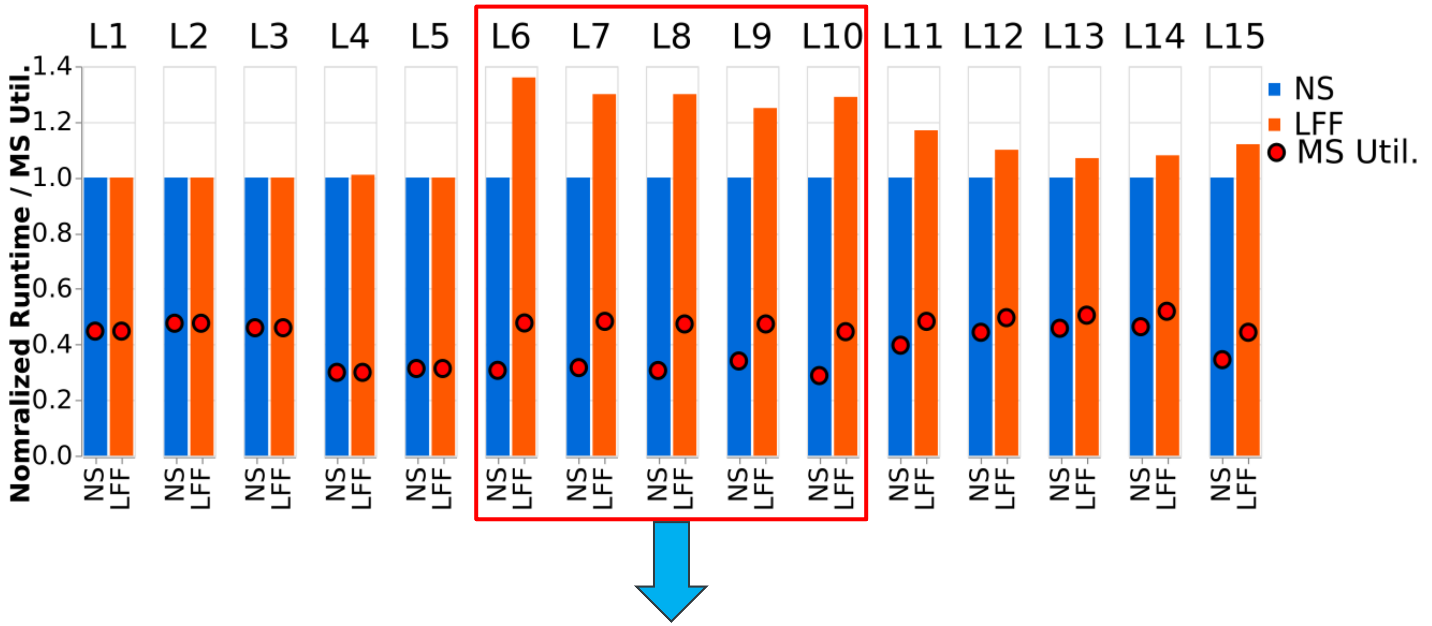
Results:



Low-Sensitive layers: Do not represent performance benefit at all

UC#3: Filter Scheduling in Sparse Accelerators

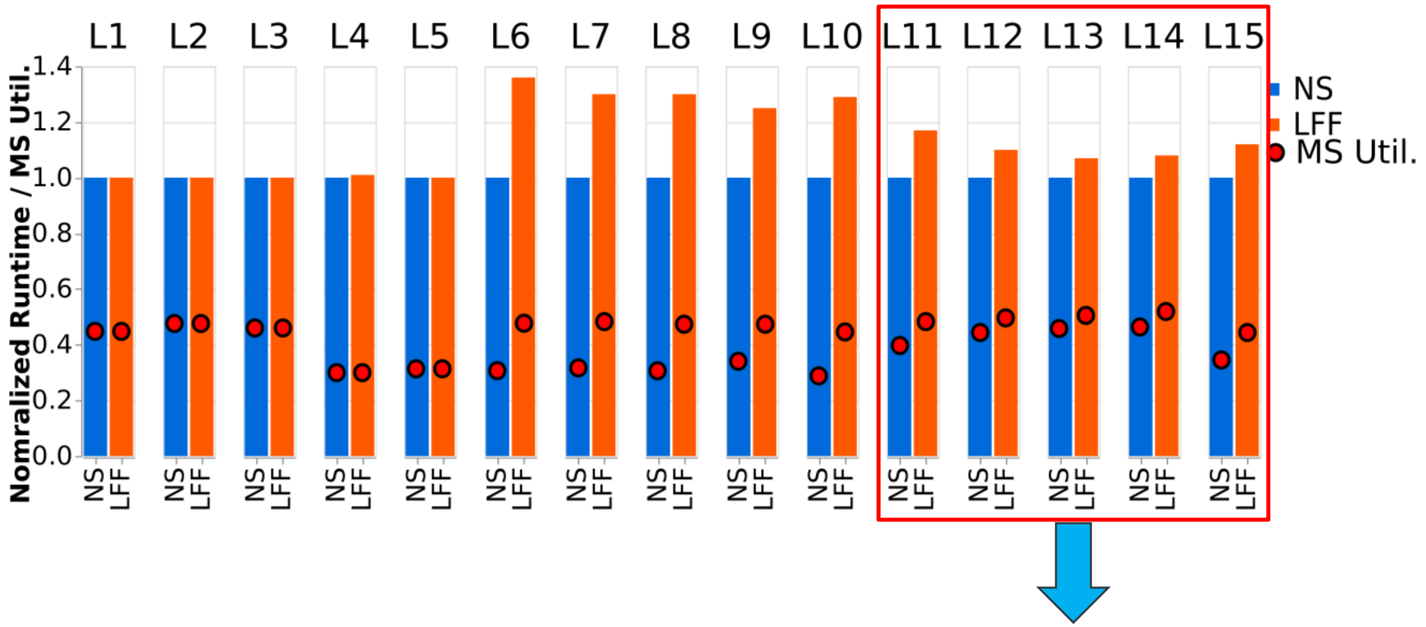
Results:



High-Sensitive layers: Up to 36% performance gain

UC#3: Filter Scheduling in Sparse Accelerators

Results:



Medium-Sensitive layers: Up to 17% performance gain

Outline

- Motivation
- STONNE Framework
- Validation
- Uses Cases of STONNE
- Conclusions

Conclusions

- As the complexity of the microarchitecture of DNN accelerators grows, the analytical models are not able to capture many important subtleties that simulation at cycle level does.
- **STONNE** is an accurate cycle-level simulator for next-generation DNN accelerator architectures.
- **STONNE** can model rigid, flexible and data-dependent accelerators performing actual computation.

https://github.com/stonne-simulator/stonne

F. Muñoz-Martínez, J. L. Abellán, M. E. Acacio and T. Krishna, "STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators". Proc. of **IISWC 2021**.

thank you

STONNE

A **S**imulation **TO**ol for **N**eural **N**etwork **E**ngines

José L. Abellán, PhD

jlabellan@ucam.edu

Universidad Católica de Murcia (UCAM)