

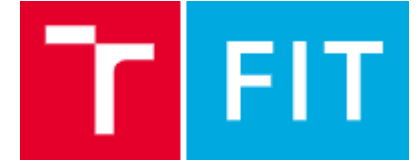
Evolutionary Optimization of Neural Network Accelerators

Vojtěch Mrázek

Faculty of Information Technology, Brno University of Technology, Czech Republic



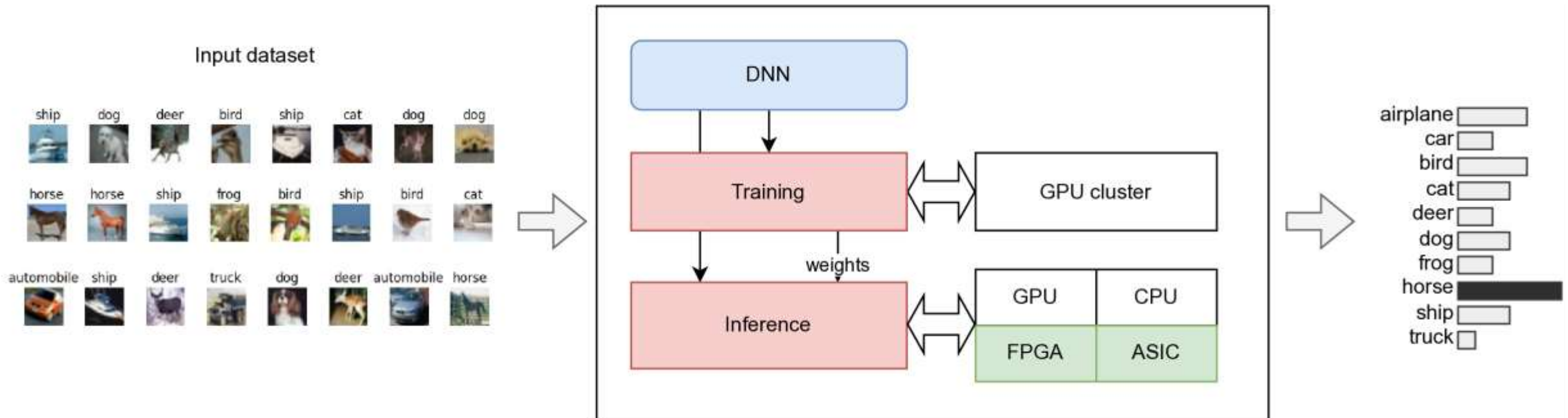
Outline



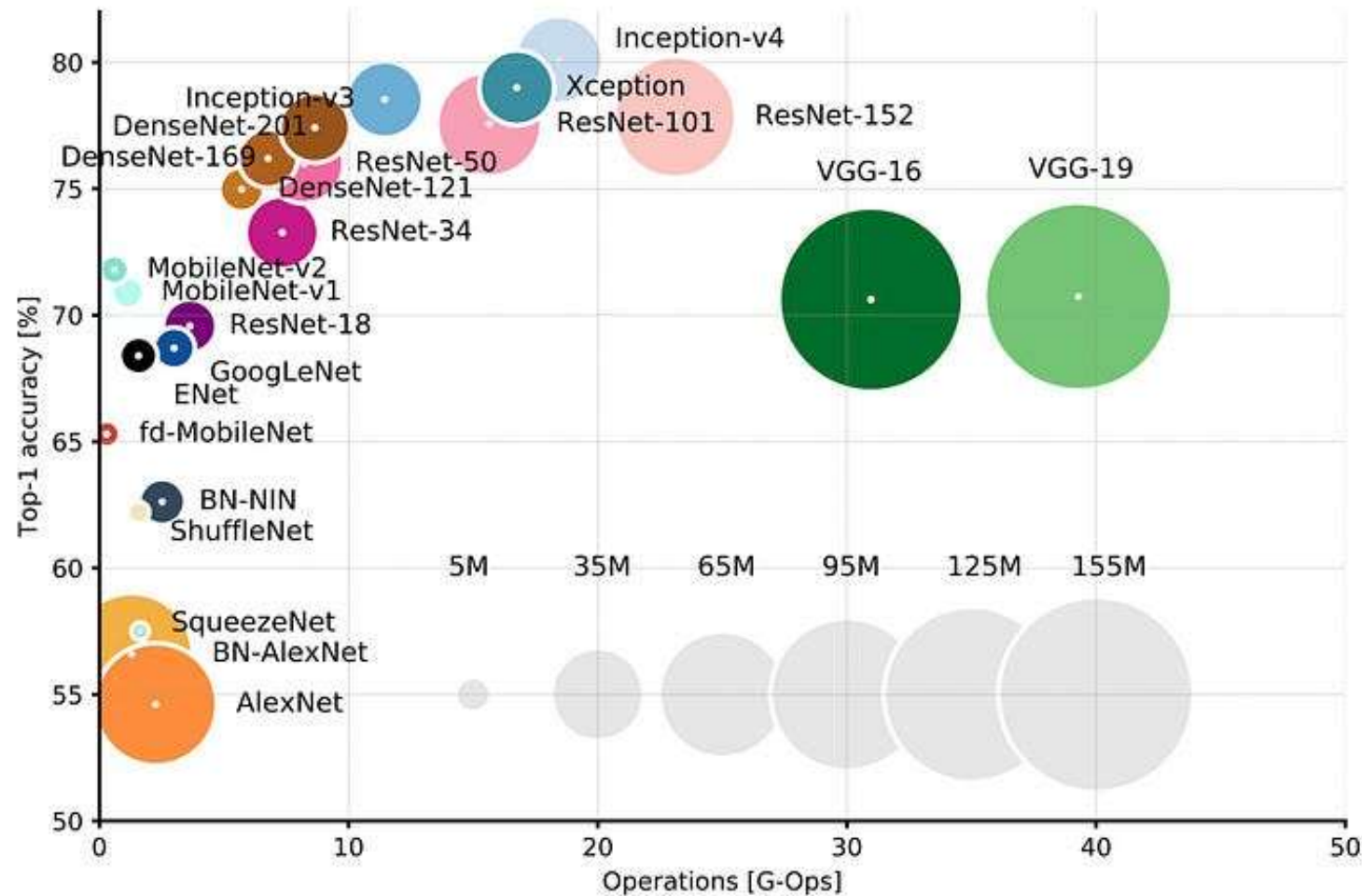
- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Deep Neural Networks

- The processing of NNs encompasses two primary phases: training and inference.
- This work will be mostly focused on the **inference** phase.



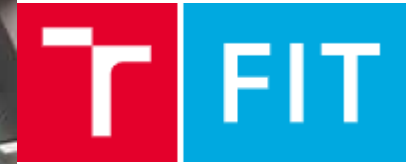
Parameters of commonly used NNs



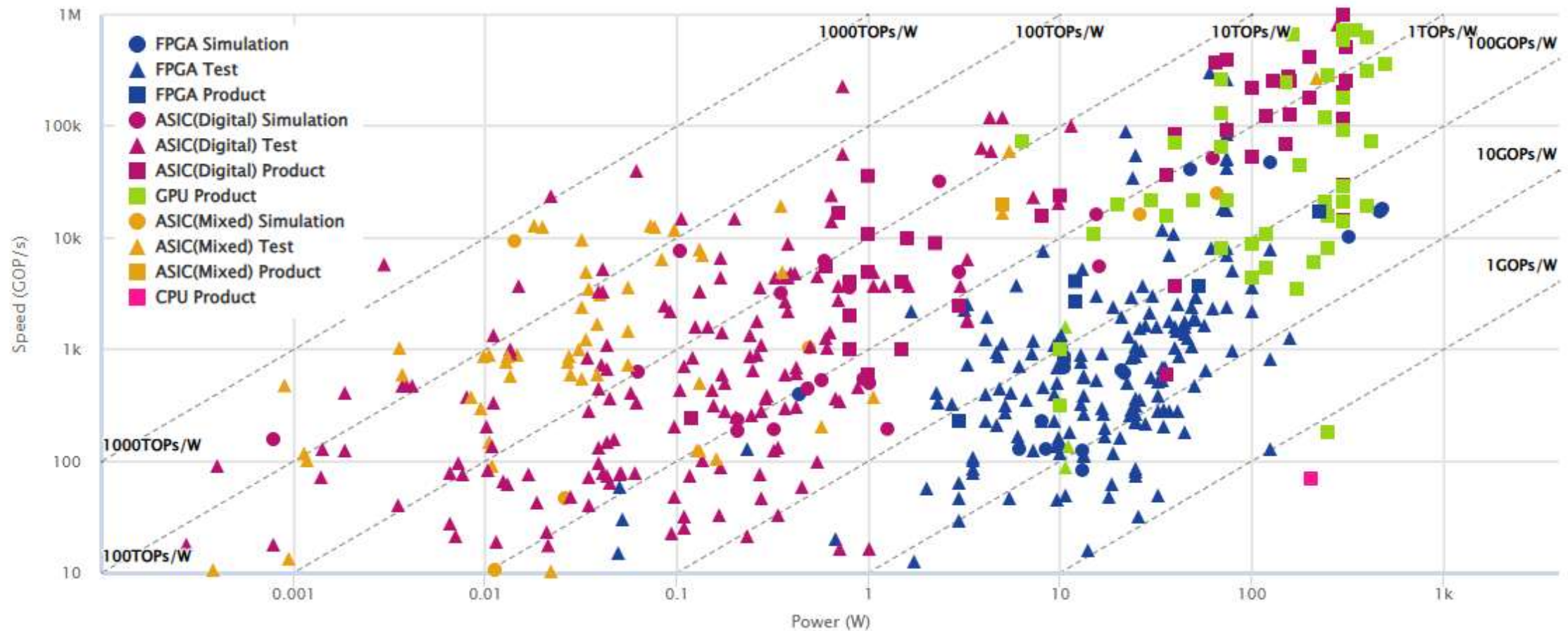
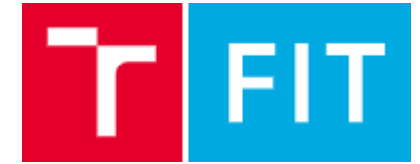
S. P. Samadhi and E. Izquierdo, "Deep-learned faces: a survey," *Eurasip Journal on Image and Video Processing*, vol. 2020, pp. 1–33, 12 2020.

Examples of platforms for DNNs

- IT4I Karolina – 800 kW
- NVidia DGX BasePOD – 14 kW
- NVIDIA Jetson – 5 – 10 W
- Google Edge TPU coprocessor – 2W



Neural Network Accelerator Comparison



K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, T. Xie, S. Han, Y. Xie, P. Debacker, M. Verhelst, Y. Wang. "Neural Network Accelerator Comparison" [Online]. Available: <https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/>

Accelerators for inference processing

Performance: the number of inferences per second

Energy-efficiency: the number of inferences per Watt/s

Platform	Chip	Freq. [MHz]	Precision	Perform. [infer./s]	Power [W]	Efficiency [infer./s/W]
ASIC	Eyeriss	200	FX16	34.7	0.3	124.8
FPGA	Kintex KU115	235	FX8	2252	22.9	98.3
FPGA	Kintex KU115	235	FX16	1126	22.9	49.2
FPGA	Zynq XC7Z045	200	FX8	340	7.2	47.2
FPGA	Zynq XC7Z045	200	FX16	170	7.2	23.6
GPU	Jetson TX2	1 300	FP16	250	10.7	23.3
GPU	Titan X	1 417	FP32	5120	227.0	22.6
CPU	Core-i7	3 500	FP32	162	73.0	2.2

(AlexNet on various platforms, according to [56], [68])

Unconventional platforms: in-memory computing, stochastic computing, memristive, RRAM, ...



Tensor Processing Unit (TPU)

Inference only

Inference+Training

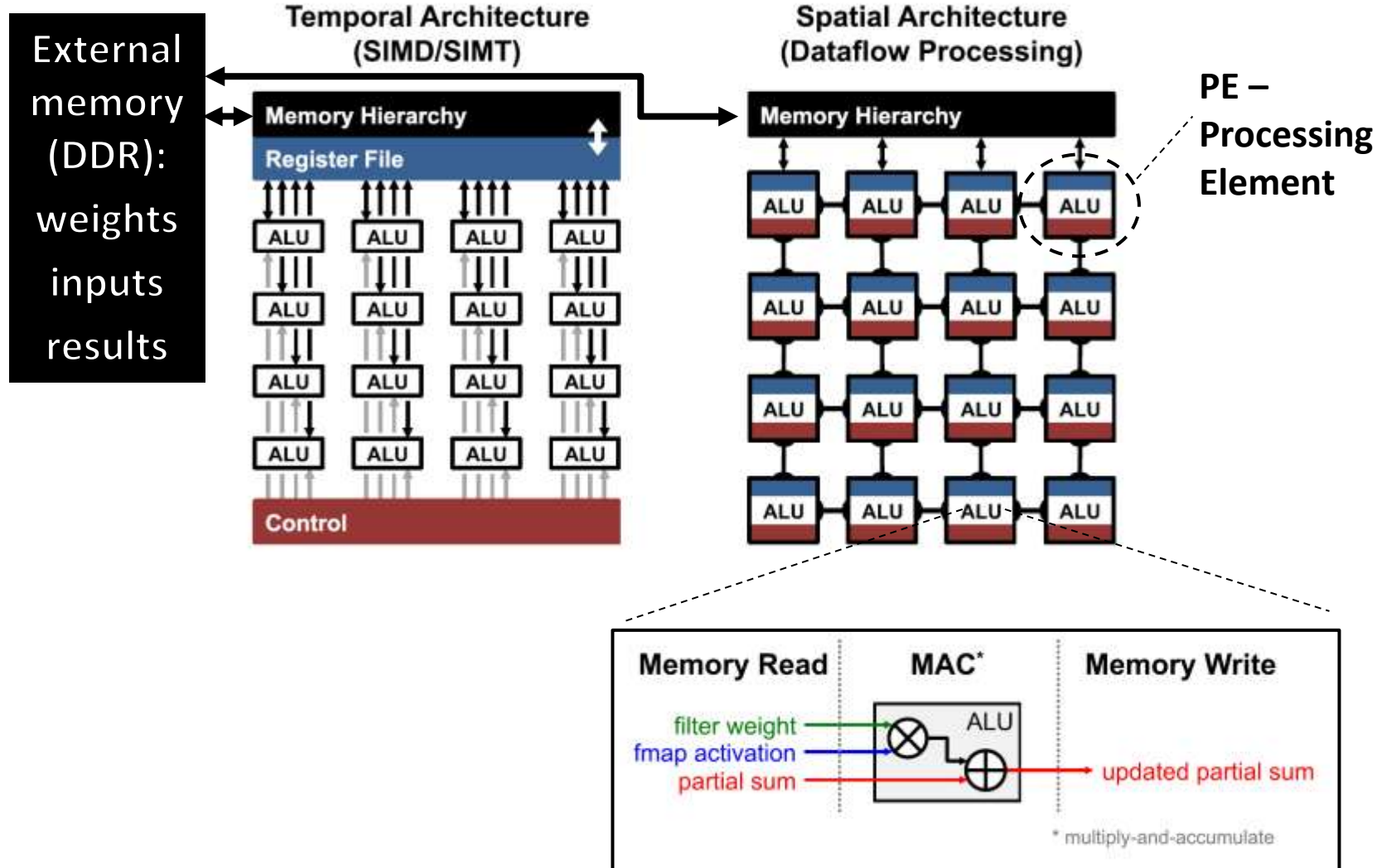
Feature	TPUv1	TPUv2	TPUv3
Peak TeraFLOPS/Chip	92 (8b int)	46 (16b) 3 (32b)	123 (16b) 4 (32b)
Network links x Gbits/s/Chip	--	4 x 496	4 x 656
Max chips/supercomputer	--	256	1024
Peak PetaFLOPS/supercomputer	--	11.8	126
Bisection Terabits/supercomputer	--	15.9	42.0
Clock Rate (MHz)	700	700	940
TDP (Watts)/Chip	75	280	450
TDP (Kwatts)/supercomputer	--	124	594
Die Size (mm ²)	<331	<611	<648
Chip Technology	28nm	>12nm	>12nm
Memory size (on/off-chip)	28MiB/8GiB	32MiB/16GiB	32MiB/32GiB
Memory GB/s/Chip	34	700	900
MXUs/Core, MXU Size	1 256x256	1 128x128	2 128x128
Cores/Chip	1	2	2
Chips/CPU Host	4	4	8

Joupi et al, Comm. of the ACM, 63(7), 2020

Two types of CNN accelerators

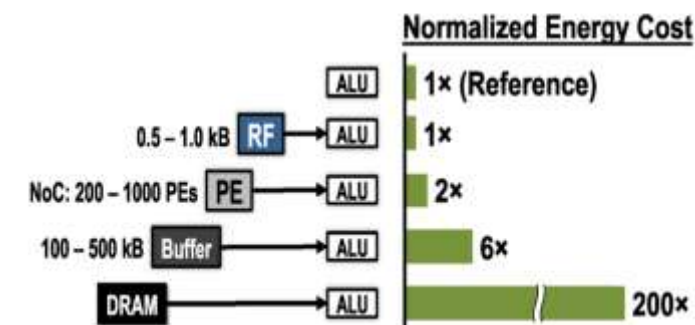
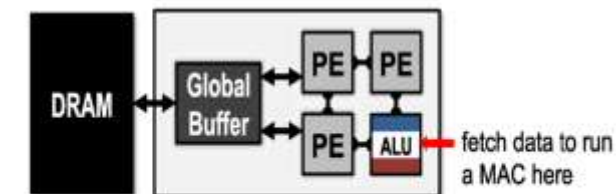
CPU and GPU
(for **training & inference**)

ASIC and FPGA
(usually for **inference** only)



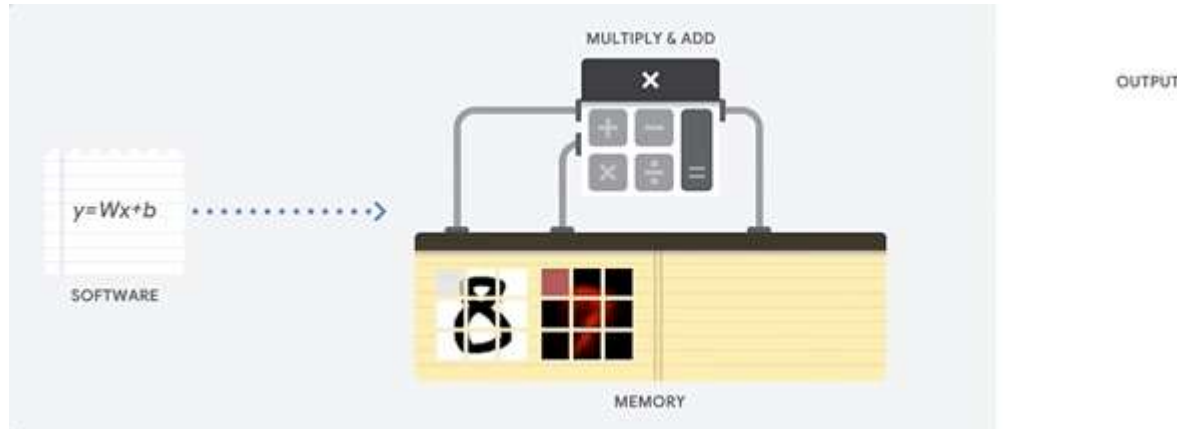
Design principles:

- Maximize the level of parallelization
- Maximize the reuse
- Minimize the communication (especially with external memory).
- Apply approximate computations



Architecture of NN accelerator: Temporal Architectures

- CPU implementation

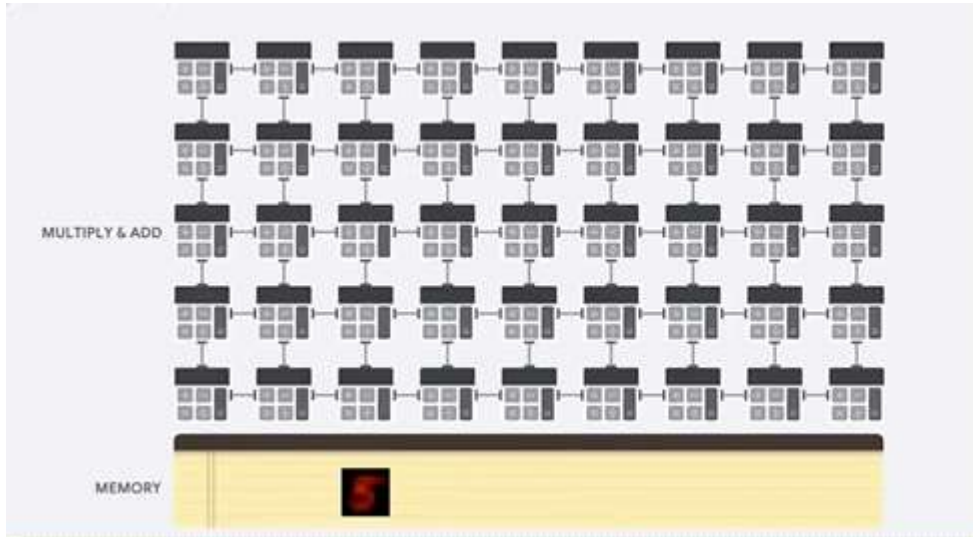


- GPU implementation – many warps in parallel

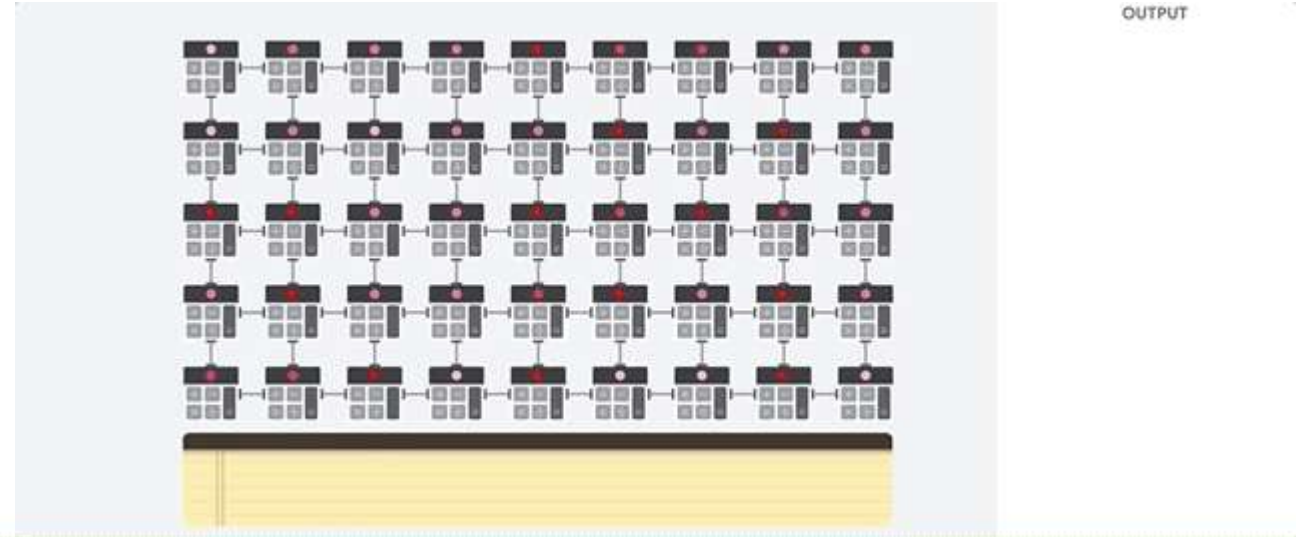


Architecture of NN accelerator: Spatial architectures

Step 1: loading kernel



Step 2: loading data and then passing



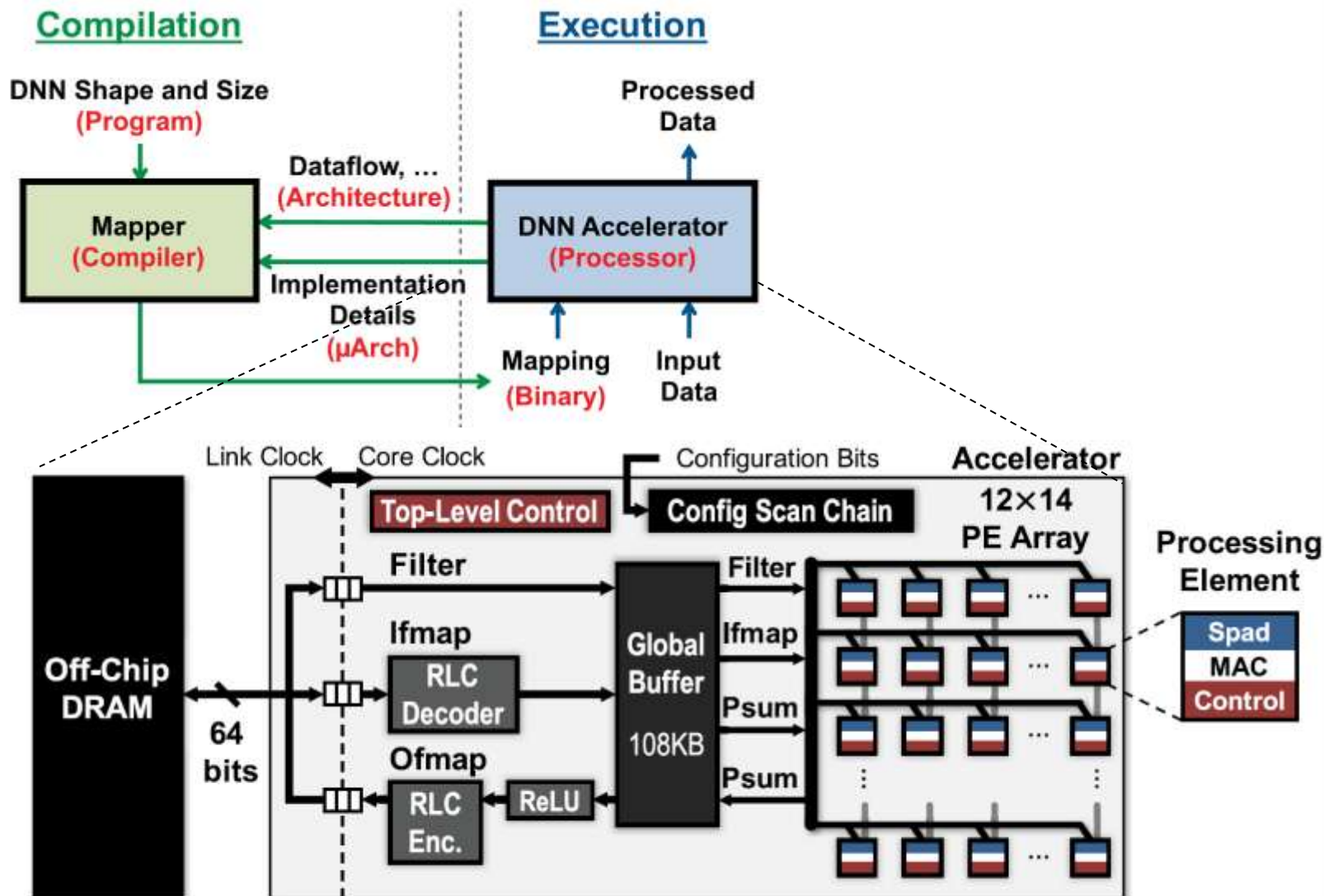
Pros

- fast
- reduced memory access
- low energy consumption

Cons

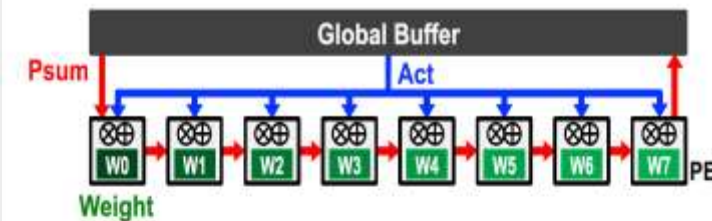
- not flexible (8-bit uint for MCUs and 16-bit float version for Cloud)
- ASIC – cost of design
- complicated planning

Example of an ASIC accelerator: Eyeriss (MIT, 2016)

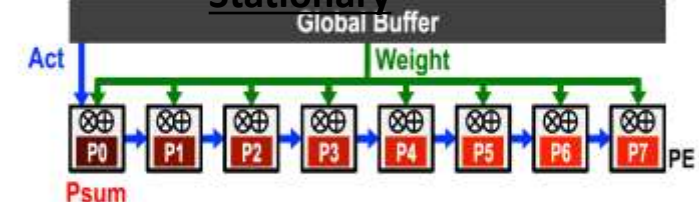


Data flow organization

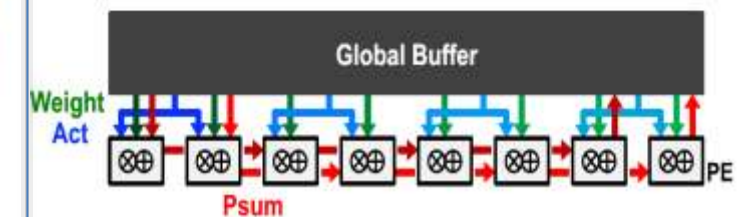
Weight Stationary



Output Stationary



No Local Reuse



Other: Row Stationary, etc.

Embedded neural networks



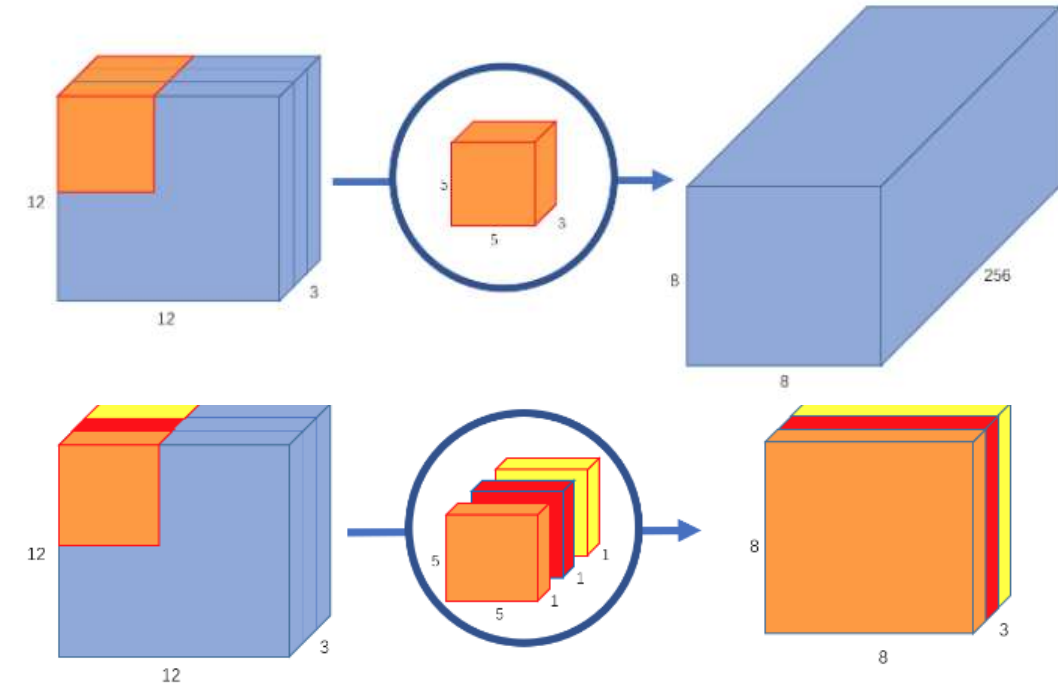
- **Reduced set of functions**

FULLY_CONNECTED, MAX_POOL_2D, SOFTMAX, LOGISTIC, SVDF, CONV_2D, CONCATENATION, DEPTHWISE_CONV_2D, AVERAGE_POOL_2D, ABS, SIN, COS, LOG, SQRT, RSQRT, SQUARE, PRELU, FLOOR, MAXIMUM, MINIMUM, ARG_MAX, ARG_MIN, LOGICAL_OR, LOGICAL_AND, LOGICAL_NOT, RESHAPE, EQUAL, NOT_EQUAL, GREATER, GREATER_EQUAL, LESS, LESS_EQUAL, CEIL, ROUND, STRIDED_SLICE, PACK, PAD, PADV2, SPLIT, UNPACK, NEG, ADD, MUL, QUANTIZE, DEQUANTIZE, RELU, RELU6, MEAN

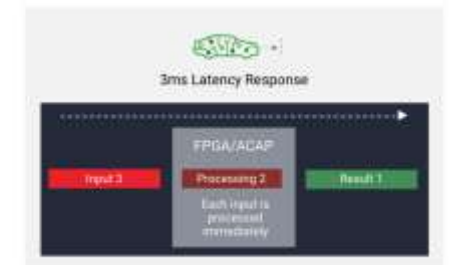
- Preferring of **Depth-wise convolutions**

- Do we really need to run inference at the edge?

- Cost of communication
- Privacy
- Latency



High Throughput **OR** Low Latency



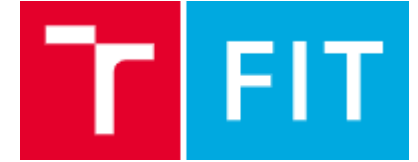
High Throughput **AND** Low Latency

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/micro/kernels/all_ops_resolver.cc

Challenges of embedded neural networks

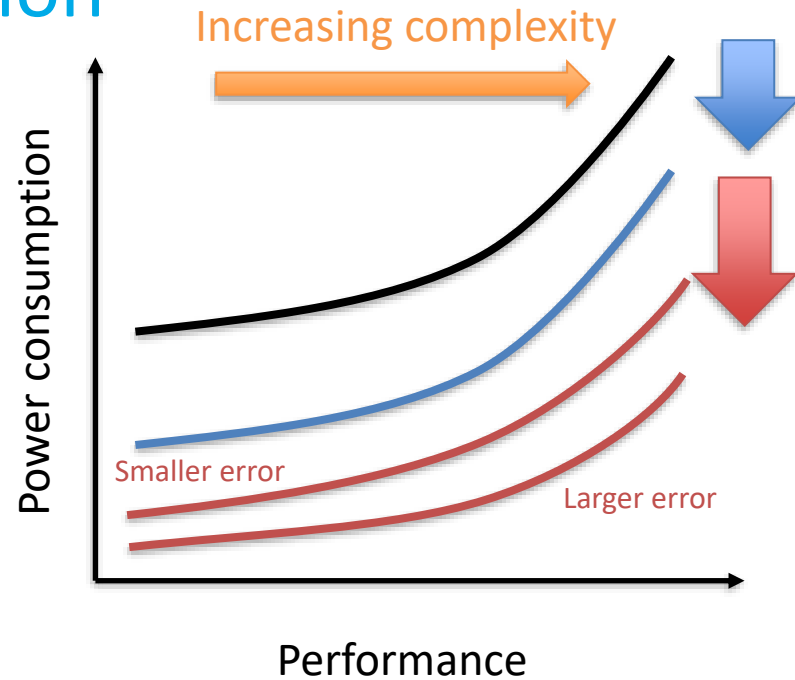
- Reduced set of layer types
- Integer representation
 - Low dynamic range of integer operations
 - Can cause many problems in the training => post-training **quantization**
- Very limited resources
 - Memory and memory-bandwidth
 - **Energy budget**

Outline



- Neural networks for embedded systems
- **Approximate computing**
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Motivation



Modern technologies & design techniques

Introducing some computational errors

Approximate computing



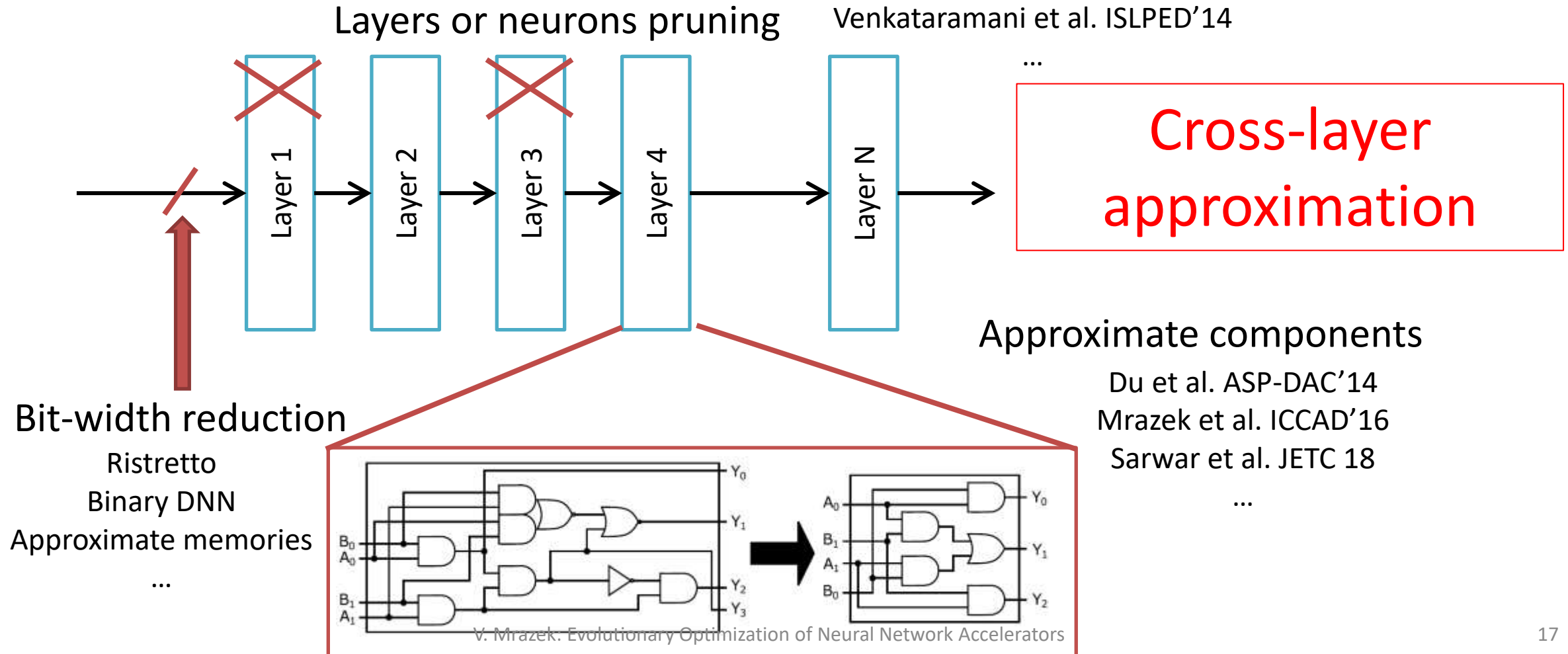
Image Courtesy: Institut für Technische Informatik - Universität Stuttgart

- Many computationally intensive applications feature an intrinsic property – the error resilience.
- Users are often willing to accept certain errors in some cases.

Approximate computing - a design paradigm for energy-efficient system.

Energy savings in neural network accelerators

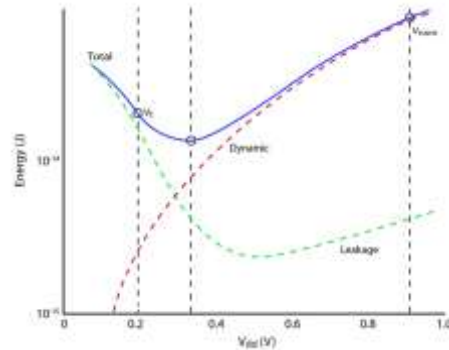
- Where can the approximations be introduced?



Hardware approaches for the approximate computing

Physical approximation

- voltage scaling
- near threshold computing
- inexact memories



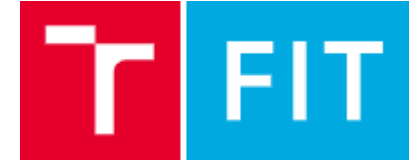
Architectural approximation

- function skipping
- data precision reduction
- inexact models (regression)

Functional approximation

- modification of the Boolean function
- the rest of the tool flow (synthesis & implementation keep same)
- two major task
 - **design of approximate components**, in particular adders and multipliers
 - **high-level approximate synthesis** of complex hardware accelerators.

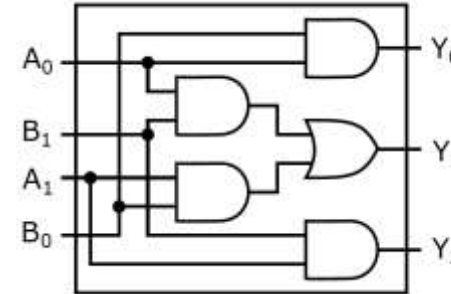
Outline



- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

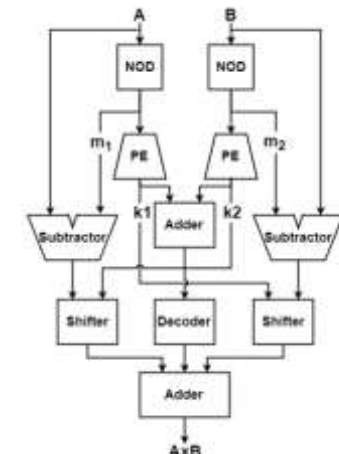
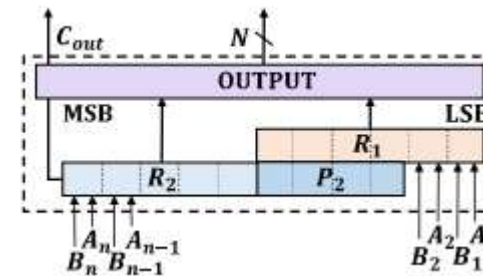
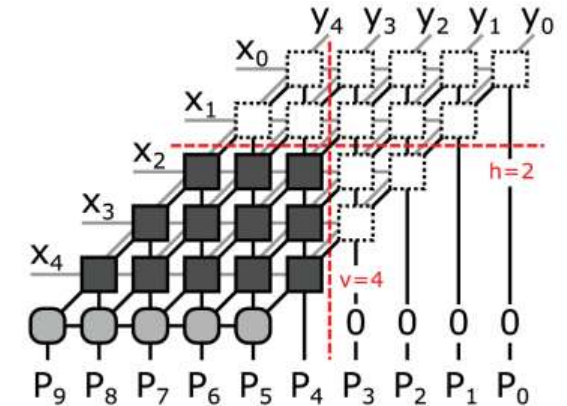
Manual functional approximation

- Elementary units such as full-adders or 2x2 multipliers were replaced by approximate implementation [Kulkarni 2011]
- The structure of circuits was modified [Mahdiani 2010]
- The longest computational paths of the circuit were cut [Hanif 2017]
- Mathematical properties of circuits were exploited [Ansari 2019]

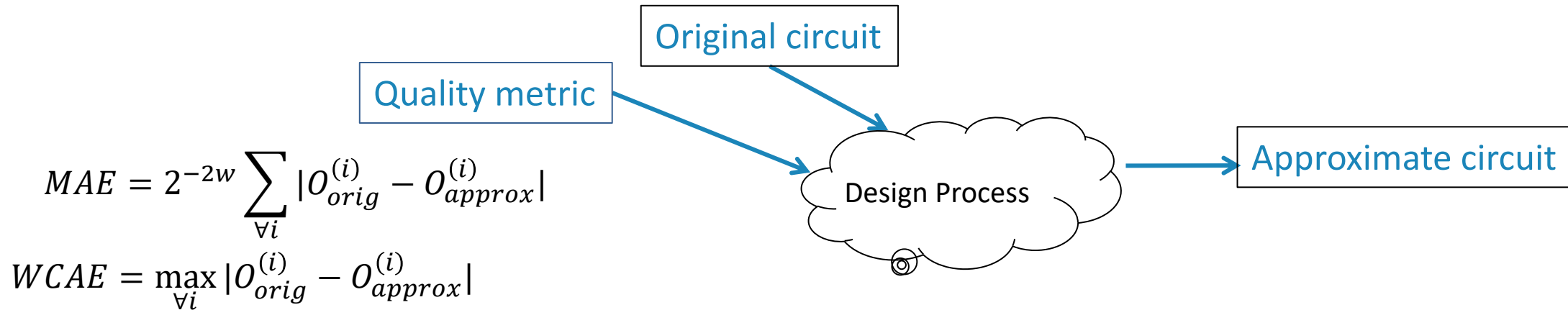
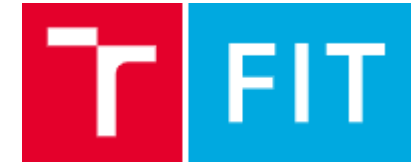


$\backslash B$	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	7

Power savings
37%



Design methodology for functional approximation

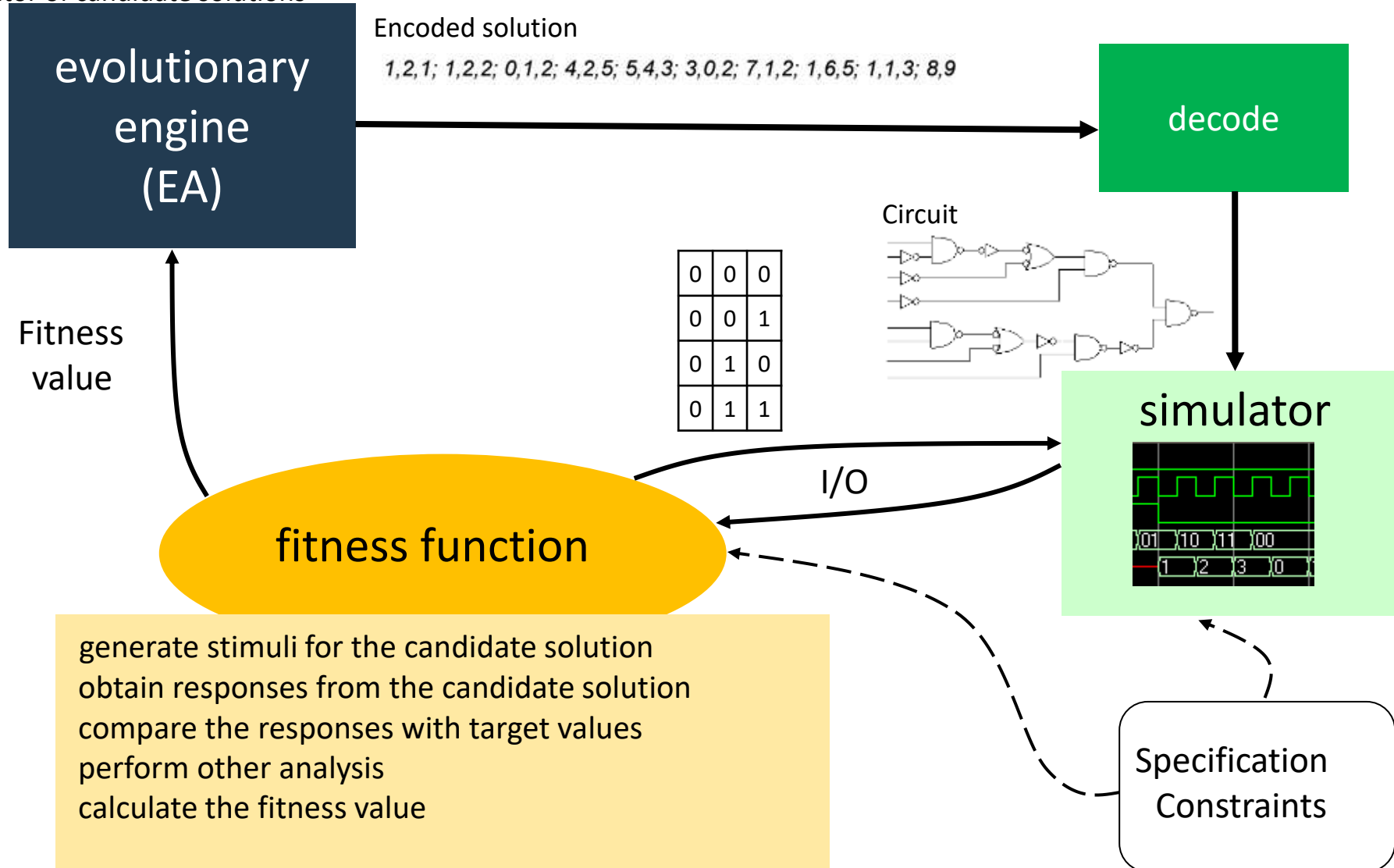


Automated CAD

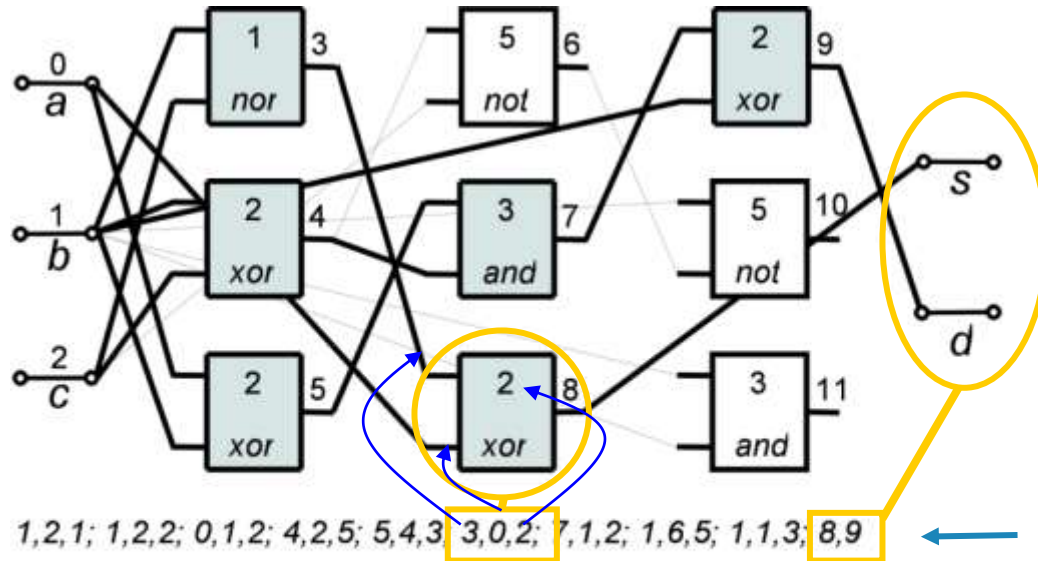
SALSA: Systematic logic synthesis using Quality Constraint Circuit [Venkataramani et al, DAC 2012],
SASIMI: Substitute-and-simplify [Venkataramani et al, DATE 2013],
Search-based (evolutionary) synthesis [Sekanina, Vasicek, ICES 2013, Mrazek et.al, ICCAD'16, Ceska et al. ICCAD'17],
ABACUS: AST-based approach [Nepal et al., DATE 2014],
ASLAN: [DATE 2014]
Approximation-aware Rewriting of AIGs [Chandrasekharan et al., ICCAD 2016]
BLASYS [DAC 2018]

Evolutionary algorithm

A generator of candidate solutions



Cartesian Genetic Programming

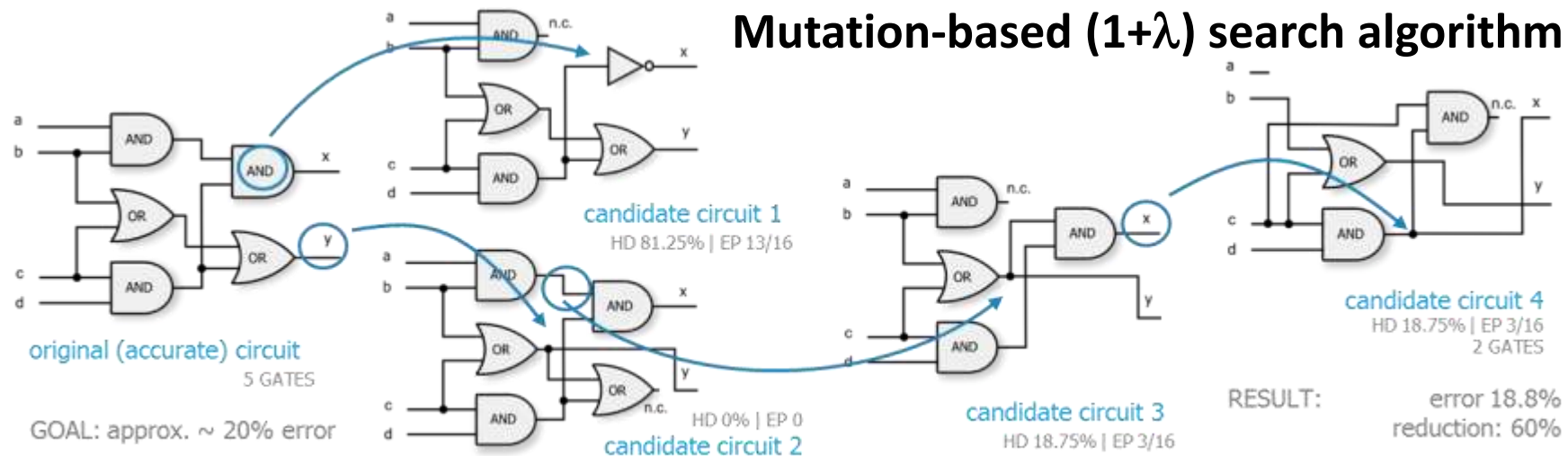


Example: CGP parameters

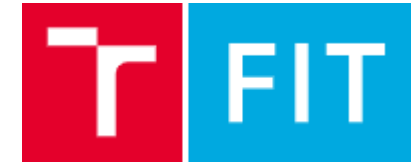
- $n_r=3$ (# rows)
- $n_c=3$ (# columns)
- $n_i=3$ (# inputs)
- $n_o=2$ (# outputs)
- $\Gamma=\{\text{NAND}^{(0)}, \text{NOR}^{(1)}, \text{XOR}^{(2)}, \text{AND}^{(3)}, \text{OR}^{(4)}, \text{NOT}^{(5)}\}$

NETLIST

Mutation-based ($1+\lambda$) search algorithm



Fitness calculation



- Fitness function evaluates candidate solutions.
- Better solutions obtain better scores.
- In our case:
- $$f(C) = \begin{cases} size(C) & \text{if } WCAE(C) < \tau \\ \infty & \text{else} \end{cases}$$
- where C is a candidate circuit and WCAE is the worst case absolute error, τ is target error.

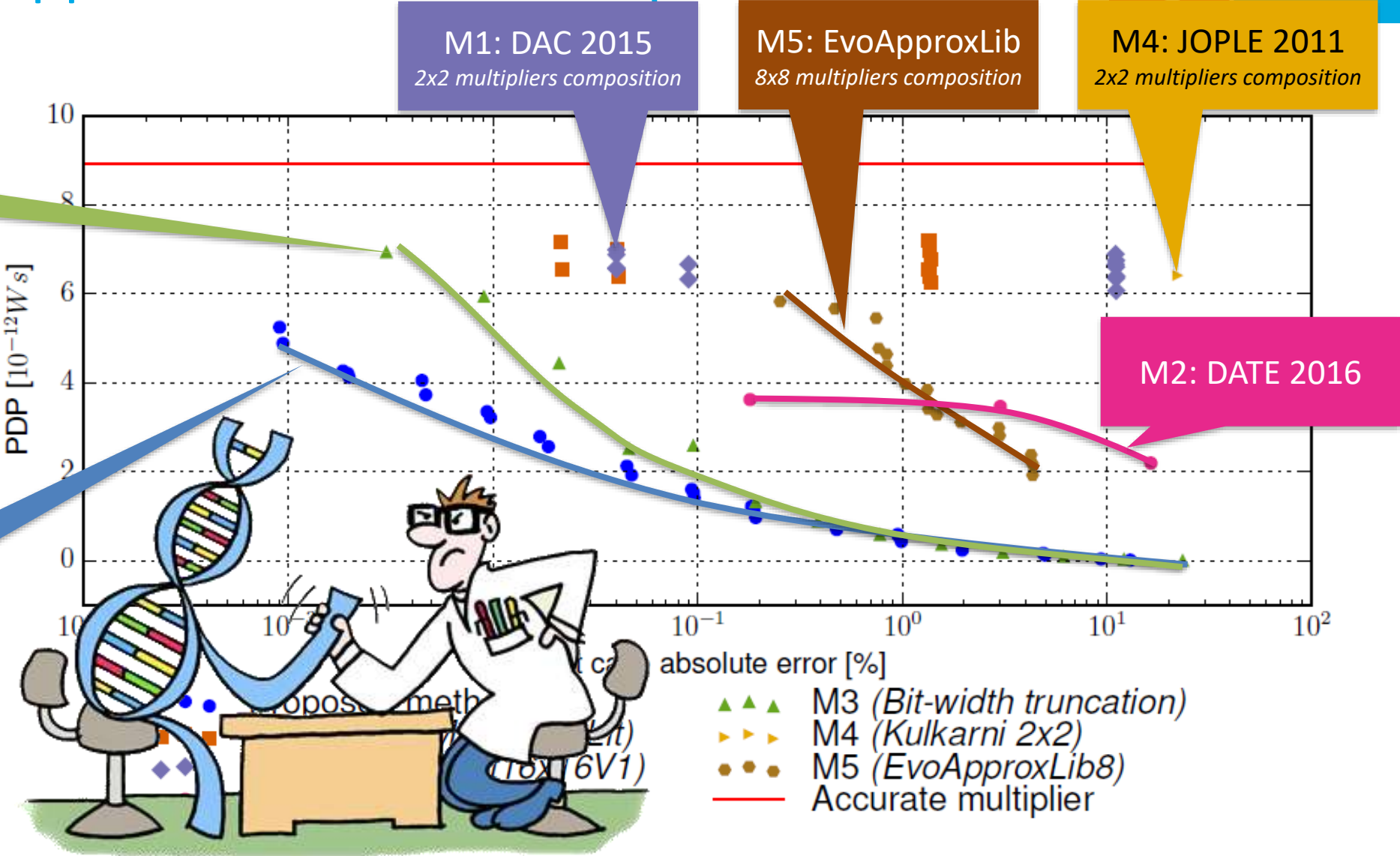
Gate	Size
INV	1.4079
AND	2.3465
OR	2.3465
XOR	4.693
NAND	1.8772
NOR	2.3465
XNOR	4.693
BUF	2.3465

Comparison of approximate 16-bit multipliers



Truncated mult.
15x15, 14x14 etc.

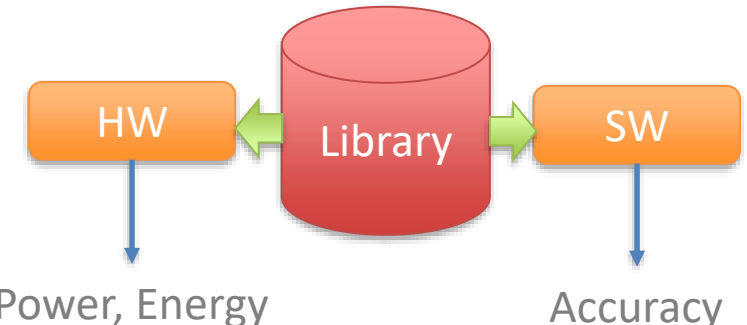
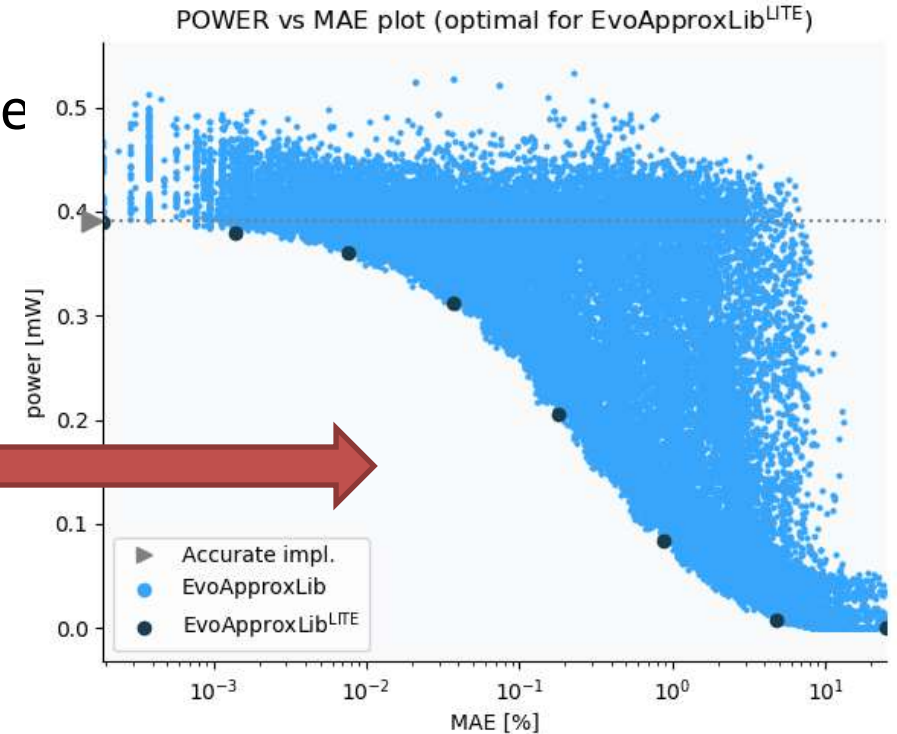
Designed multipliers



Approximate computing: EvoApproxLib

- Approximate circuits available in component libraries are electrical parameters and various error metrics.
- The circuits are Pareto-optimal in all metrics.

Circuit	Bit-width	# components
multiplier	8	29,911
	12	3,495
	16	35,406
	32	349

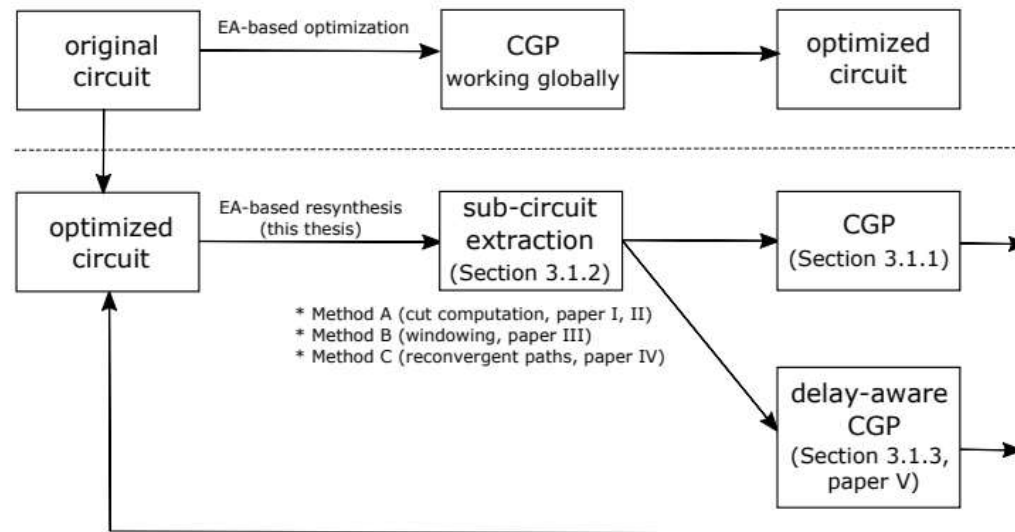


<https://github.com/ehw-fit/evoapproxlib>

MRÁZEK, V.; SEKANINA, L.; VAŠÍČEK, Z. Libraries of Approximate Circuits: Automated Design and Application in CNN Accelerators. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2020, vol. 10, no. 4, p. 406-418. ISSN: 2156-3357.

Research questions and possible solutions

- Time of design & non-deterministic approach
- Scalability of evaluation
 - Using advanced datasets
 - Employing formal verification techniques (BDDs for adders, SAT with limit for multipliers)
- Scalability of representation
 - Extracting small subcircuits of the circuits and optimize them separately => how to do it for approximate circuits?



Outline

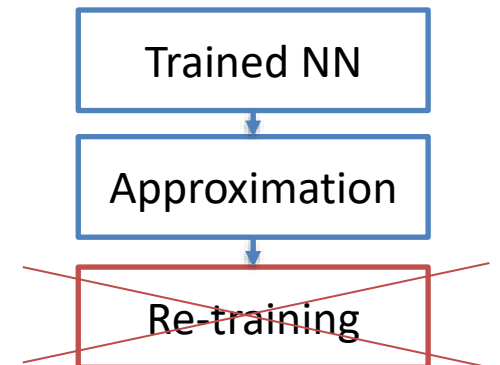


- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - **Advanced approximation of NN**
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Approximate components in neural networks



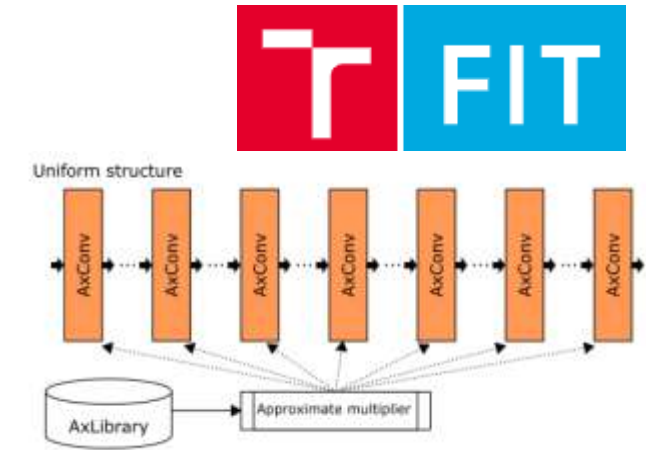
- The bit-width reduction can be generalized using approximate arithmetic components.
- Approximate arithmetic component = Boolean function with two n -bit operands and m -bit output; in our case realizing multiplication function.
- As there is no support in current NN frameworks, we implemented our own layers (convolutional)
 - Approximate multiplication simulated using a lookup table, but there is a bottleneck for speed
- In the previous approaches, researchers performed additional retraining to adapt the NN to the approximation => time consuming and typically results in
 - approximation of significantly smaller networks (limited scalability)
 - limited set of considered approximate components.



Approximate components in NN: challenges

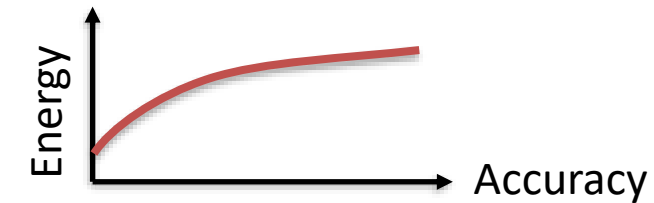
The layers have different error resilience, however the approximate NNs employ the **same component in all layers** (uniform structure)

We propose an algorithm assigning the approximate components to the layers.



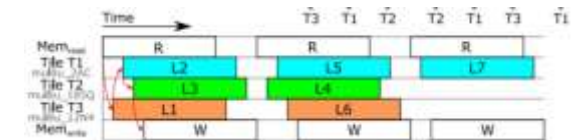
How to obtain a good **tradeoff** between accuracy and energy consumption

A multi-objective optimization is performed.



Different **architectures of accelerators** : pipelined or power-gated

The proposed methodology can handle different accelerator architectures.



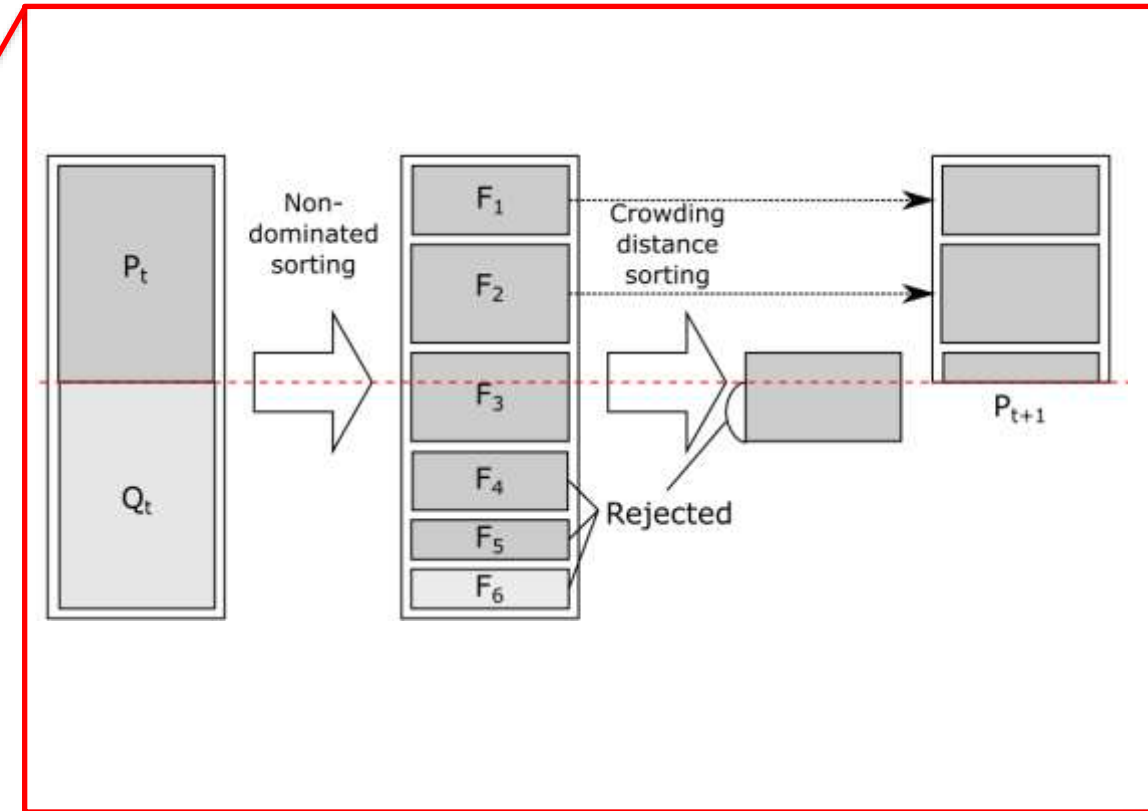
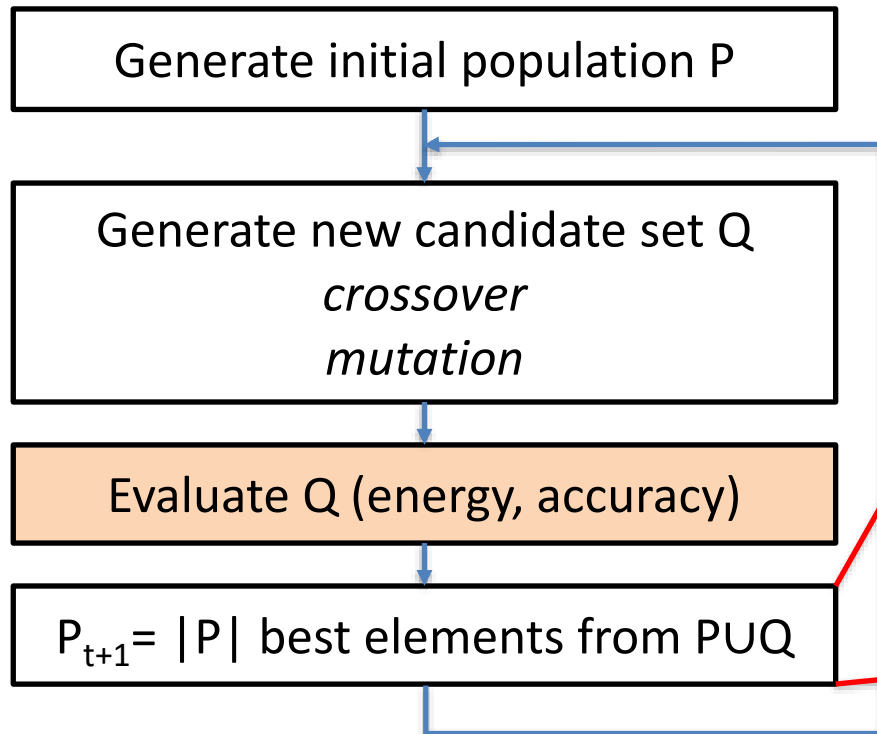
NSGA-II as optimization algorithm

Design space:

Assigning one of 20 approximate multipliers to one of 50 layers: 10^{65} combinations.

Heuristic space search exploration:

Multi-objective NSGA-II



Problem representation

The target accelerator based on systolic array is composed of $|T|$ **tiles** with the same approximation unit (multiplier)

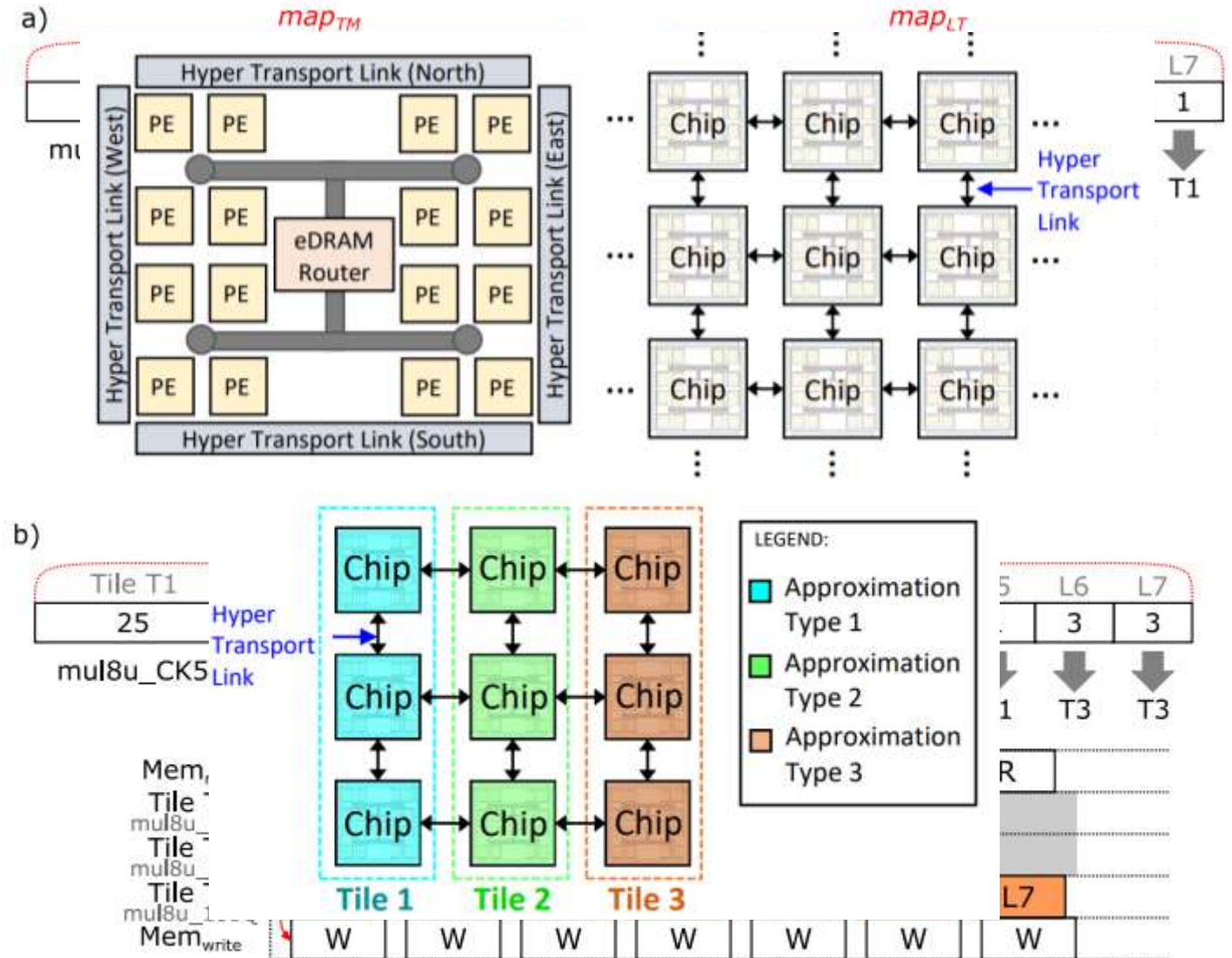
The optimization algorithm is searching for **mappings**:

$map_{TM}: Tiles \rightarrow Multipliers$

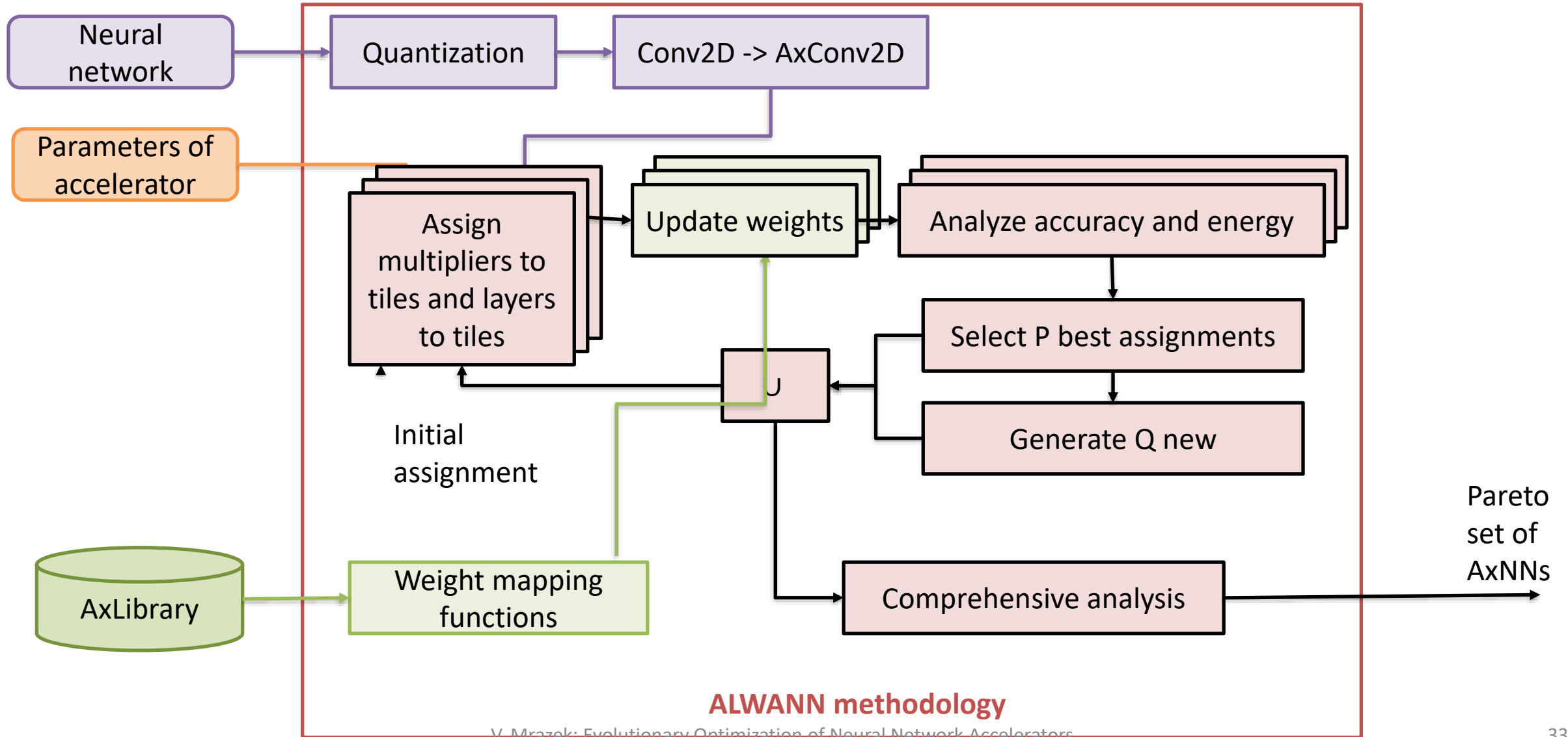
$map_{LT}: Layers \rightarrow Tiles$

Two architectures are modelled

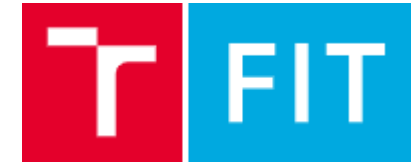
- **Pipelined**: $|T|$ -tuples are distributed to the tiles
- **Power-gated**: arbitrary mapping of layers to tiles is allowed



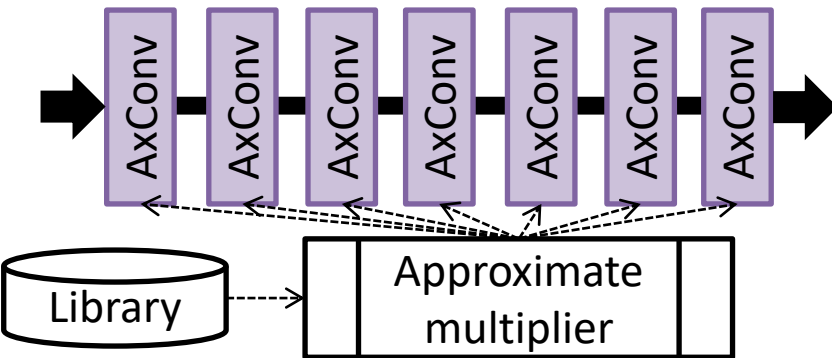
ALWANN methodology overview



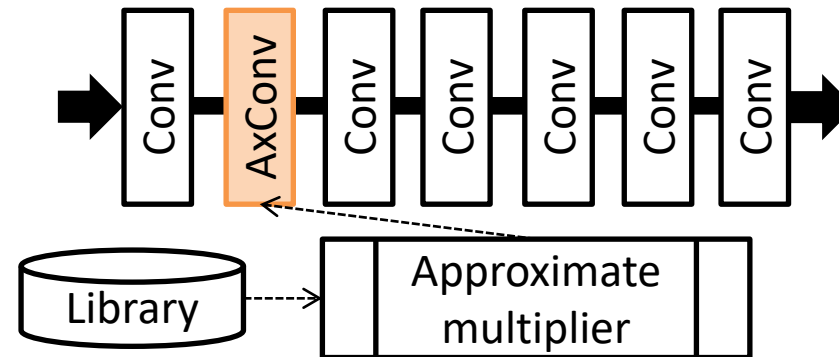
Structures of NNs: Do we need non-uniform structure?



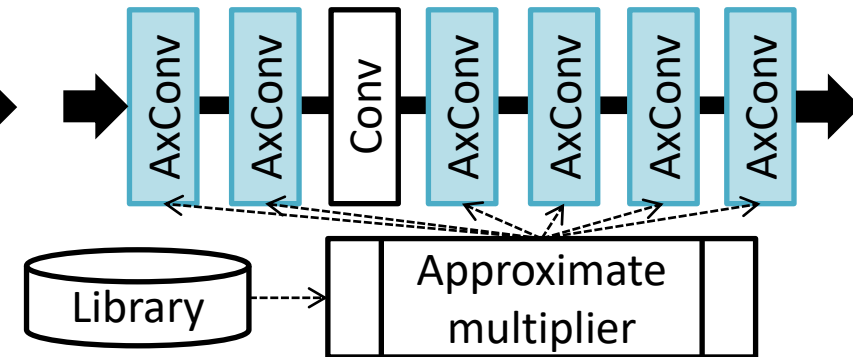
Uniform structure



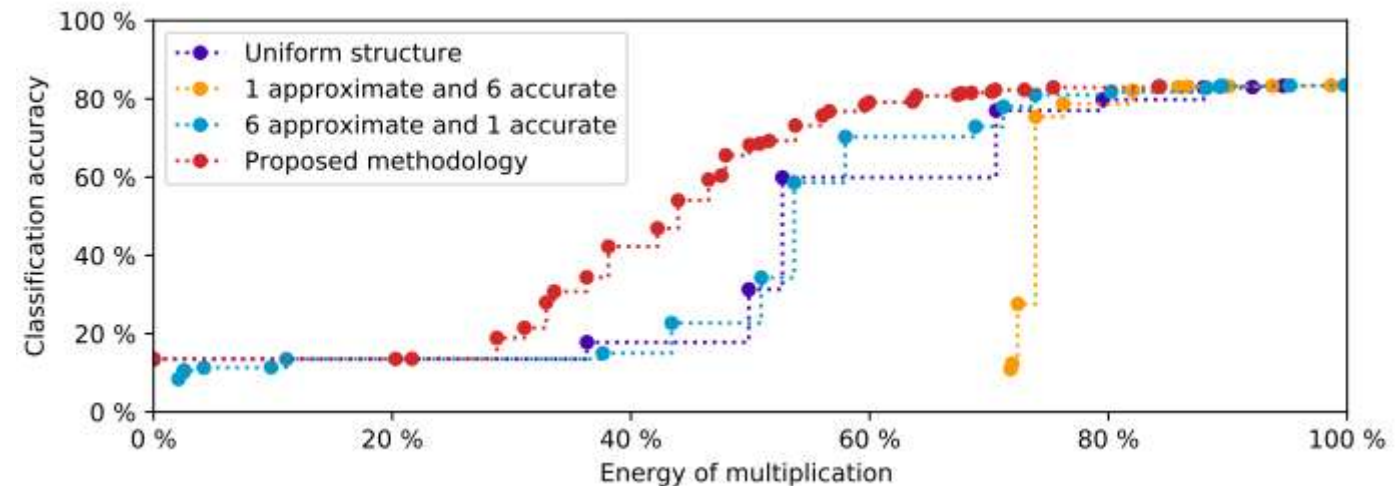
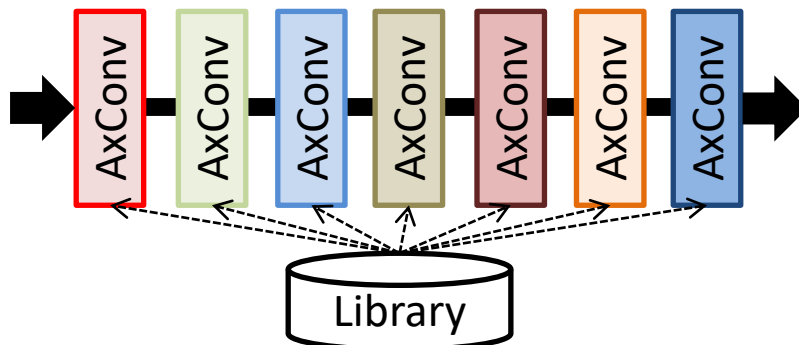
1 approximate



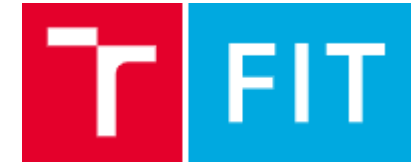
N-1 approximate



Proposed methodology



Experimental setup



ResNet networks trained for CIFAR-10 dataset

ResNet instance	# conv. layers	# mults.	accuracy (floating-point)	accuracy (qint-8)
ResNet-8	7	21.1M	83.42%	83.26%
ResNet-14	13	35.3M	85.93%	85.55%
ResNet-50	49	120.3M	89.38%	89.15%

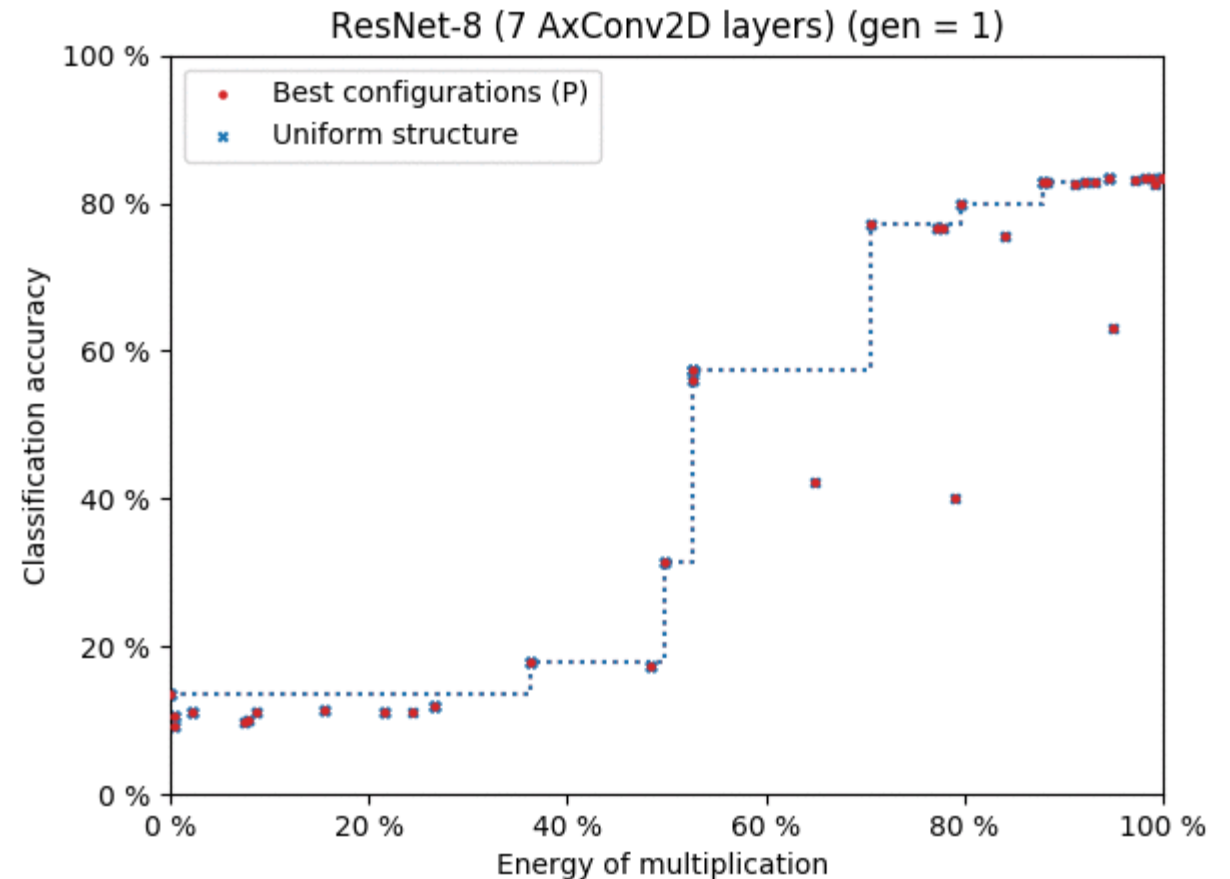
New AxConv2D layers implemented in TensorFlow framework.

A set of 35 approximate multipliers and 1 accurate from EvoApproxLib^{LITE} library.

NSGA-II: $|P|=50$, $|Q|=50$, in total 30 generations (1550 evaluations).

Energy estimation:

$$E(N) = \sum_{l \in \text{Layers}(l)} \#mults(l) \cdot \frac{E(\text{mult}(l))}{E(\text{accurate})}$$



Complexity analysis



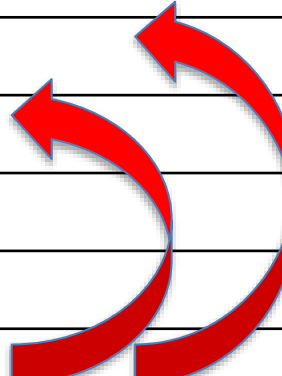
	# Layers	# Ax. components ^{*)}	Retraining	Uniform struct.
AxNN (ISLPED'14)	2-6	-	yes	no
ApproxNN (DATE'15)	2-6	8	yes	no
Mrazek et al. (ICCAD'16)	2-6	8 (from 420)	yes (2 hours for 10 steps)	yes
Sarwar et al. (JETCS'18)	2-164	4	yes (<u>limited</u> set of ax. comp.)	yes
ALWAN (proposed)	8-49	36	no (fast weight-tuning alg.)	no

AxNN	Searching loop (1k images)		Final validation (10k images)		Total
	One evaluation	Total	One evaluation	Total	
AxResNet-8	1.8 sec	0.75 h	3.2 s	2.6 min	0.79 h
AxResNet-14	2.1 sec	0.87 h	4.9 s	4.1 min	0.95 h
AxResNet-50	3.6 sec	1.5 h	14.6 s	12.2 min	1.70 h

Vaverka, Mrazek, Vasicek, Sekanina. *TFAprox: Towards a Fast Emulation of DNN Approximate Hardware Accelerators on GPU*. DATE'20.

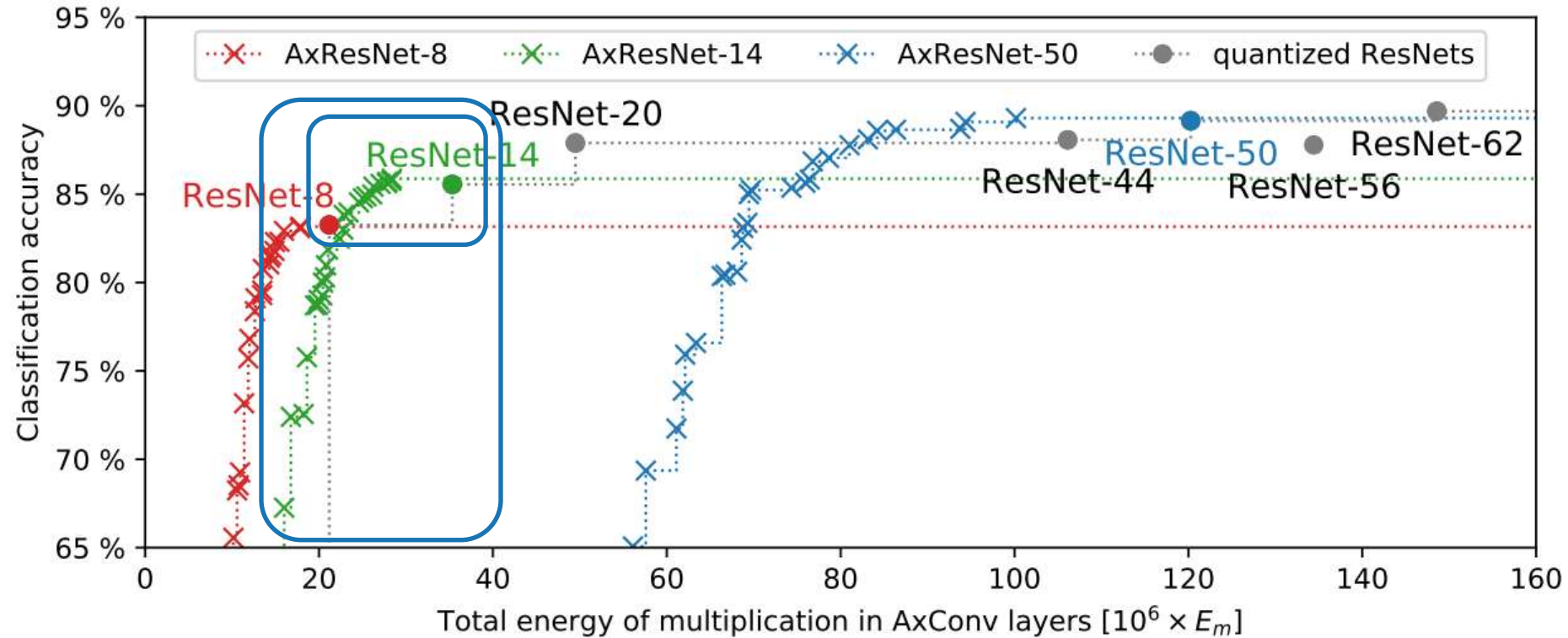
Comparison with the state of the art – CIFAR-10

Approach	Retraining	Energy	Accuracy	
Venkataramani <i>AxNN</i> [ISLPED'14]	Yes	-22%	-0.5%	
		-26%	-2.5%	
Sarwar [JETC 2018]	Yes	-33%	-1.8%	
He <i>ResNet</i> [arXiv]	Yes	-12%	-1.2%	ResNet-50 -> 44
		-71%	-4.0%	ResNet-50 -> 14
		-48%	-2.7%	ResNet-14 -> 8
Proposed methodology	No	-30%	-0.6%	AxResNet-50
		-30%	-0.9%	AxResNet-14
		-30%	-1.7%	AxResNet-8



- The power savings depends on the considered **starting point**.
- The proposed methodology saves the energy of large NNs more than SoA approaches.
- It is necessary to make a comparison with architecture scaling.

Comparison with NN scaling



Why should developers use ALWANN algorithm instead of the training smaller network

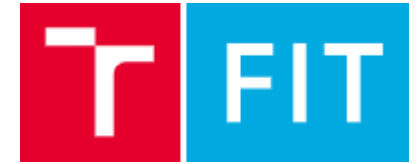
- No training data are available
- The architecture cannot be scaled
- The training time is larger than ALWANN time

Outline



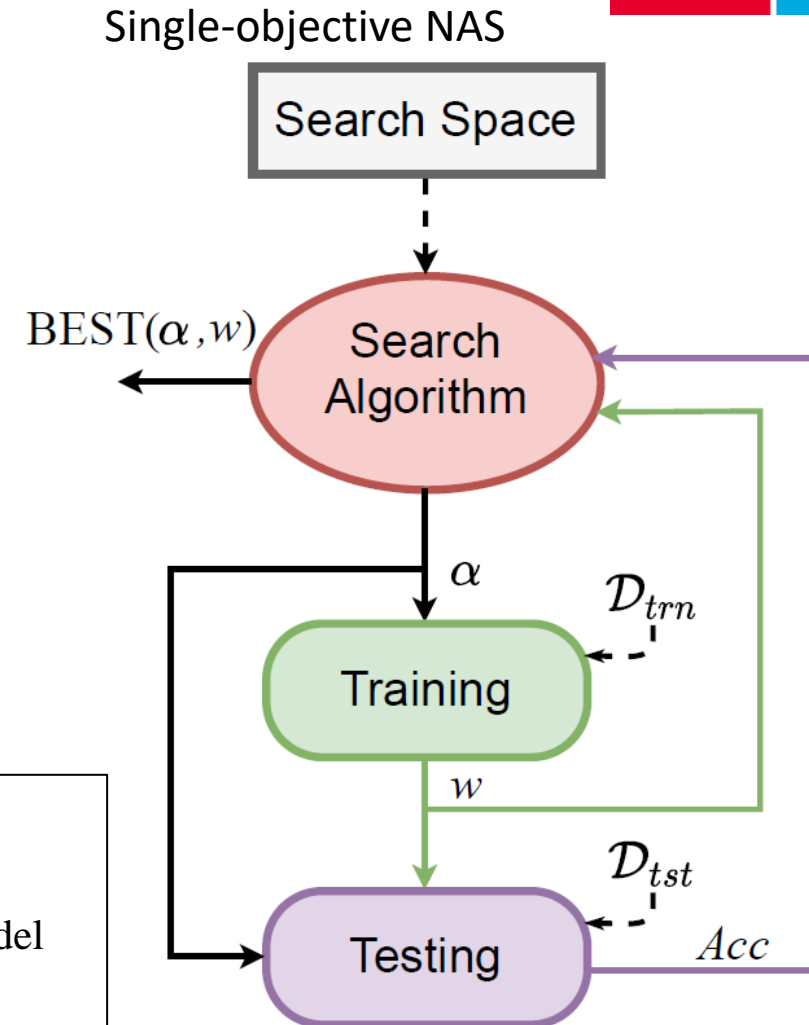
- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Single-objective NAS



- The aim of NAS is to **automate** the process of **finding the most suitable NN architecture** for a **given dataset**. The single-objective NAS has one objective - maximizing the **Accuracy**.
 - Neuro-evolution has been performed in the Evolutionary Algorithms community since the mid-1980.
 - NAS has been connected with DNNs since 2016.
- Key components of NAS methods
 - Search space
 - Search algorithm
 - Performance estimation
- Target hardware: usually GPU

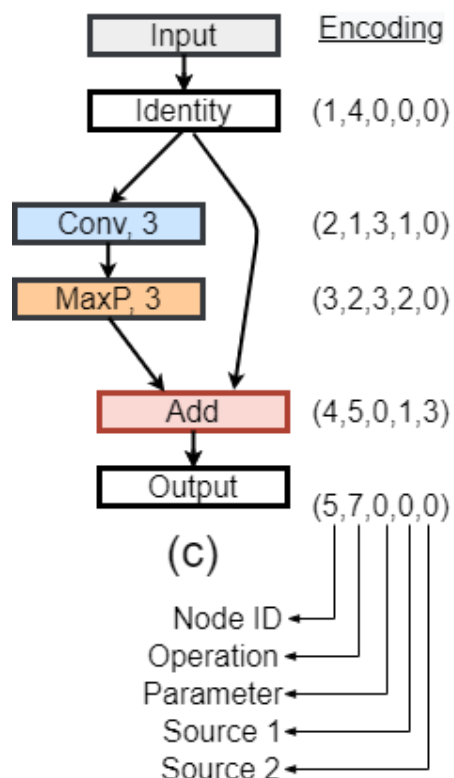
D_{trn} – training data
 D_{tst} – test data
 α – a candidate NN model
 w – weights
 Acc – Accuracy



Search Spaces and CNN encoding

Candidate CNN ~ string of integers

Search space ~ all feasible strings

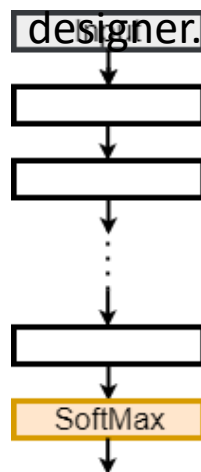


(Node ID; Operation; Parameter; Source ID 1; Source ID 2).

Set of operations: (1) convolution, (2) max. pooling, (3) average pooling, (4) identity, (5) add, (6) concatenation, (7) terminal node [87].

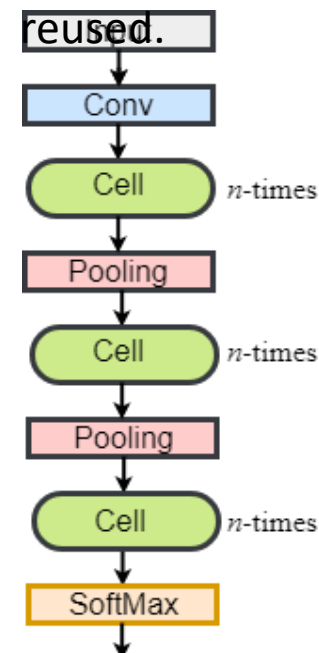
Macro search space

- The entire CNN is encoded.
- Some parts can be fixed by the designer.



Micro search space

- A subgraph (cell, block) or subgraphs is/are encoded and reused.



Hierarchical search space

- Recursive construction using a set of small graphs.

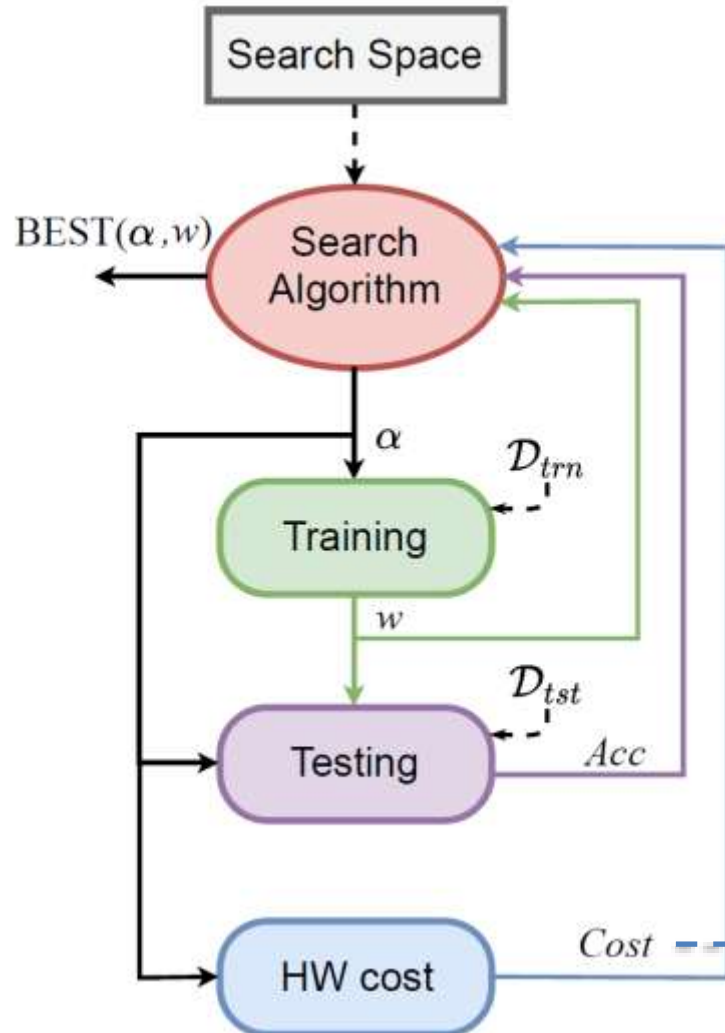
Indirect encoding

- A construction program is encoded.
- The program is executed to build a NN.

Supernets

- A large NN is pretrained and then pruned.

Multi-objective NAS for a particular (fixed) hardware

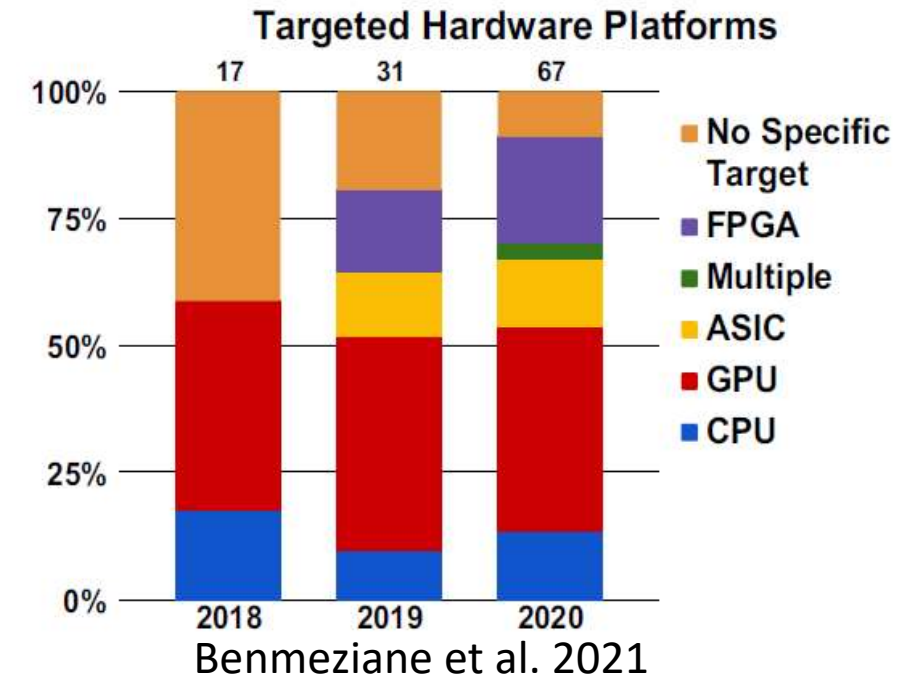


Additional Objectives:

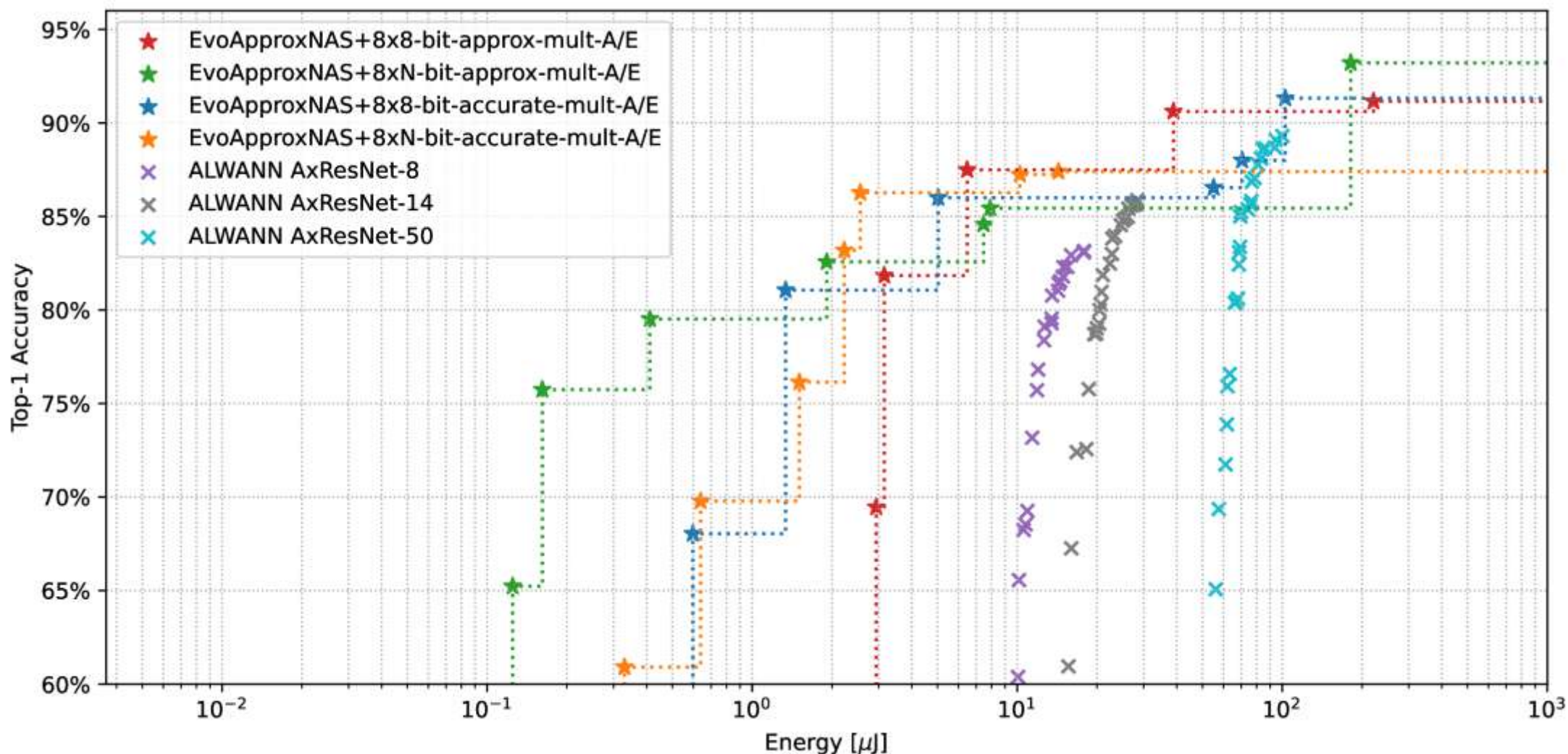
Latency
Area
Energy
RAM size
Flash size
#MAC
Reliability
etc.

Hardware-aware NAS is a NAS reflecting a given hardware executing the inference.

Can be extended by approximate operators
Pinos: Evolutionary Approximation and Neural Architecture Search.
Genetic programming and Evolvable Machines 2022

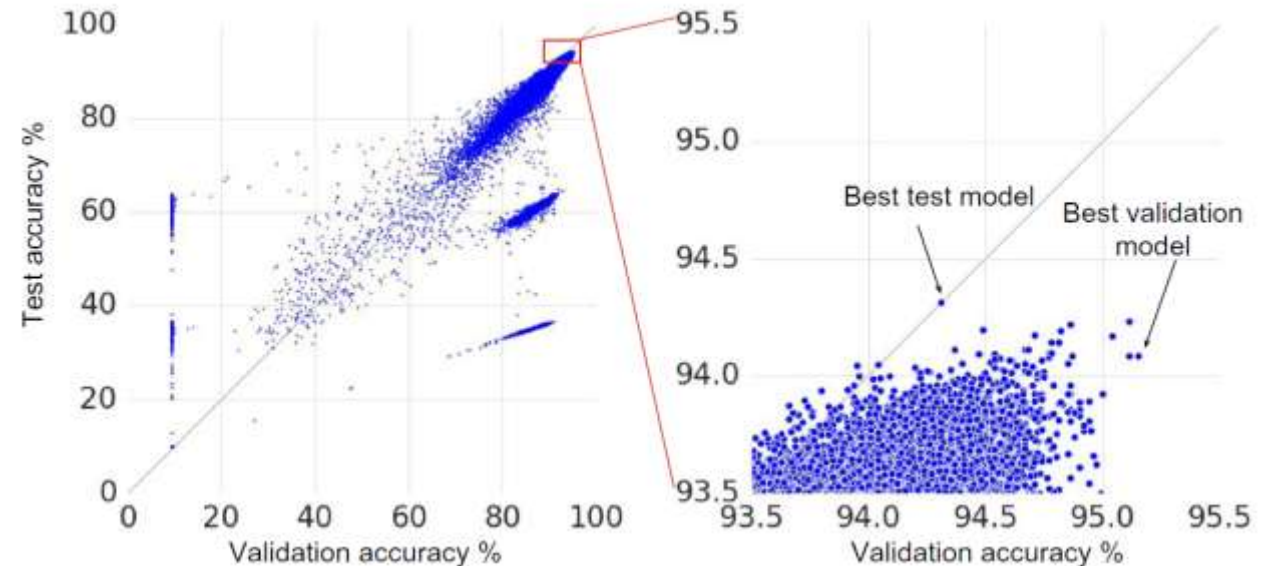
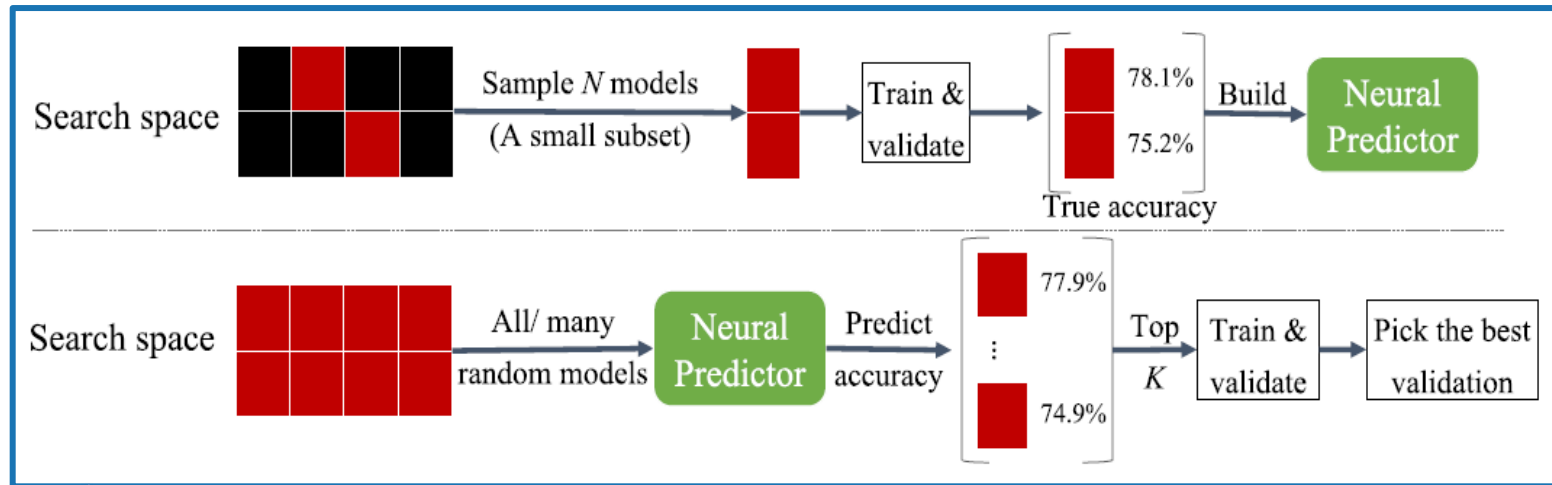


NAS with approximate components



Shortening the evaluation time: Accuracy

- Simplify the common approach
 - Employ a proxy data set
 - Reduce the number of training epochs
 - Extrapolate the learning curve
 - etc.
- Build a surrogate model – Accuracy predictor
 - NN
 - regression trees
 - Gaussian process (GP)
 - etc.



Results on NASBench-101 (CIFAR-10) by Wen W. et al. ECCV 2020

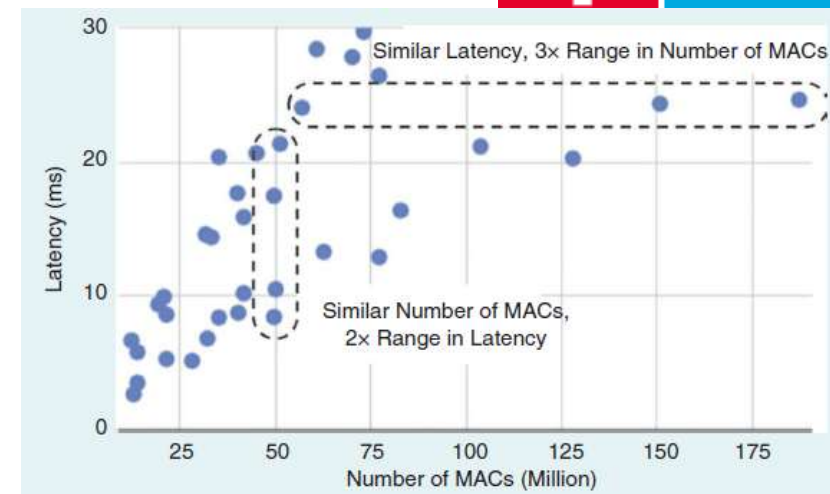
Shortening the evaluation time: Hardware metrics



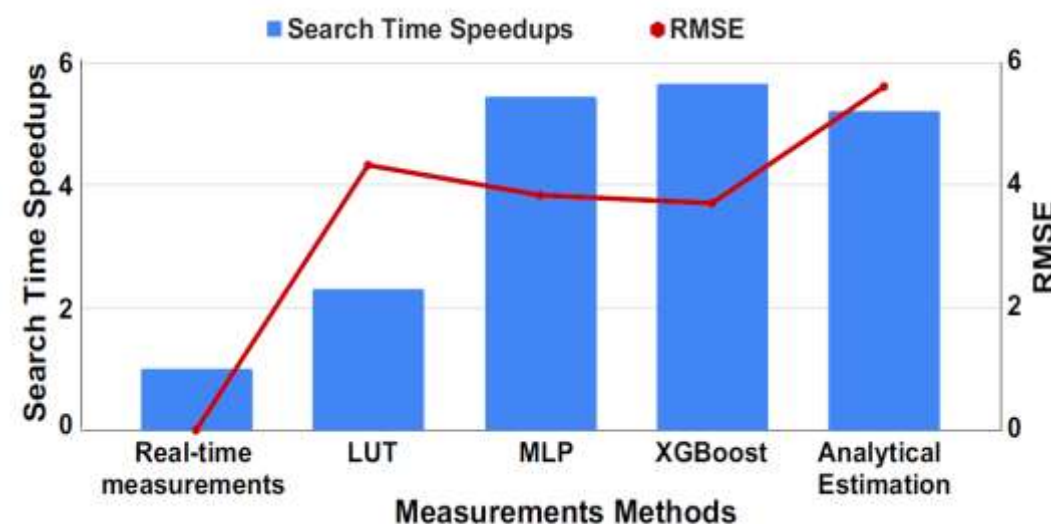
Hardware metrics: Latency, Energy, Area, Memory etc.

Methods according to Benmeziane et al. 2021:

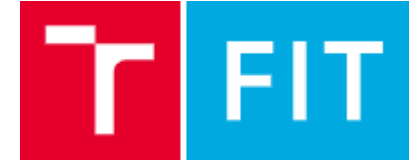
- **Baseline:** Real-time measurements on target hardware.
- **Lookup Table Models** - a lookup table is created beforehand and filled with each operator hardware metrics on the targeted hardware. Once the search starts, the system calculates the overall cost from the lookup table.
- **Analytical Estimation** - consists of computing a rough estimate using the processing time, the stall time, and the starting time.
- **Prediction Model** a ML model is built to predict the cost using architecture and dataset features.



#MAC is not a good proxy for latency! Shown for various NN models on a Google Pixel phone.

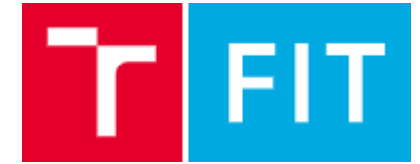


Outline

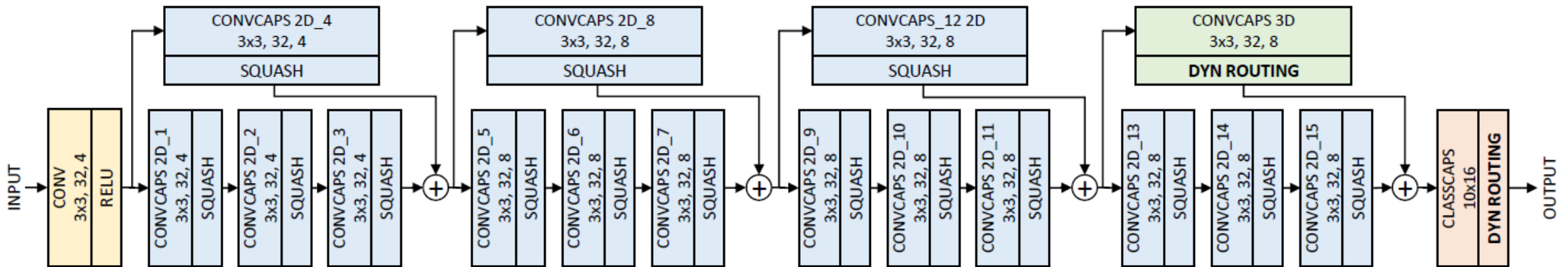


- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Analysis of the error resilience



1. Where can we introduce the error?
2. How much energy savings can we achieve?
3. What is the impact on the overall accuracy?

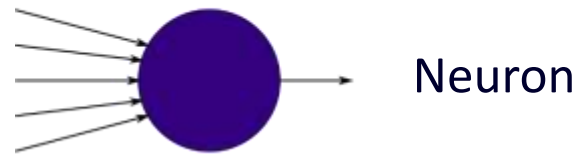


S. Sabour, N. Frosst, G.E. Hinton. Dynamic Routing Between Capsules @ NeurIPS'17

Alberto Marchisio, Vojtech Mrazek, Muhammad Abudllah Hanif, Muhammad Shafique: ReD-CaNe: A Systematic Methodology for Resilience Analysis and Design of Capsule Networks under Approximations @ DATE'20. arXiv: 1912.00700

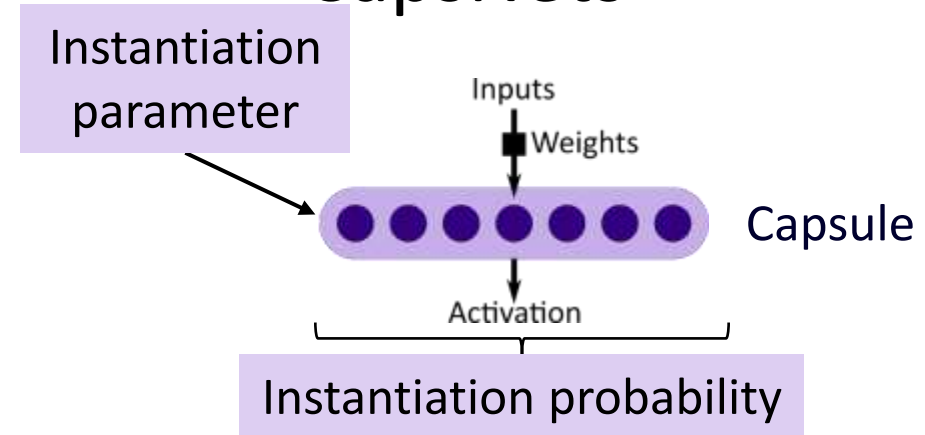
Capsule Neural Networks

Traditional DNNs



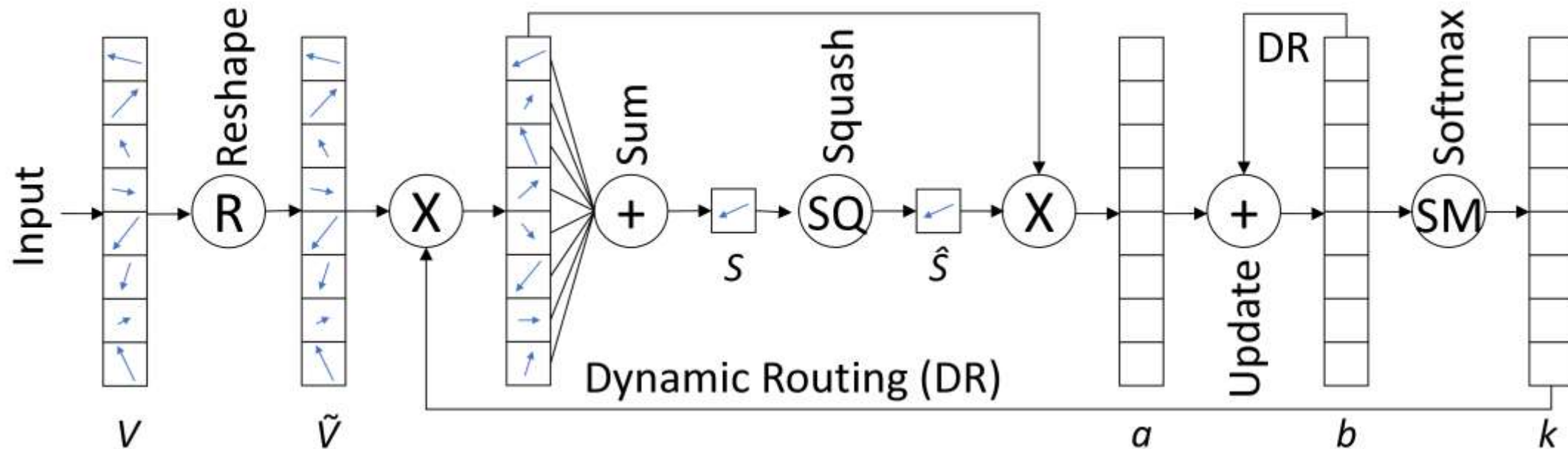
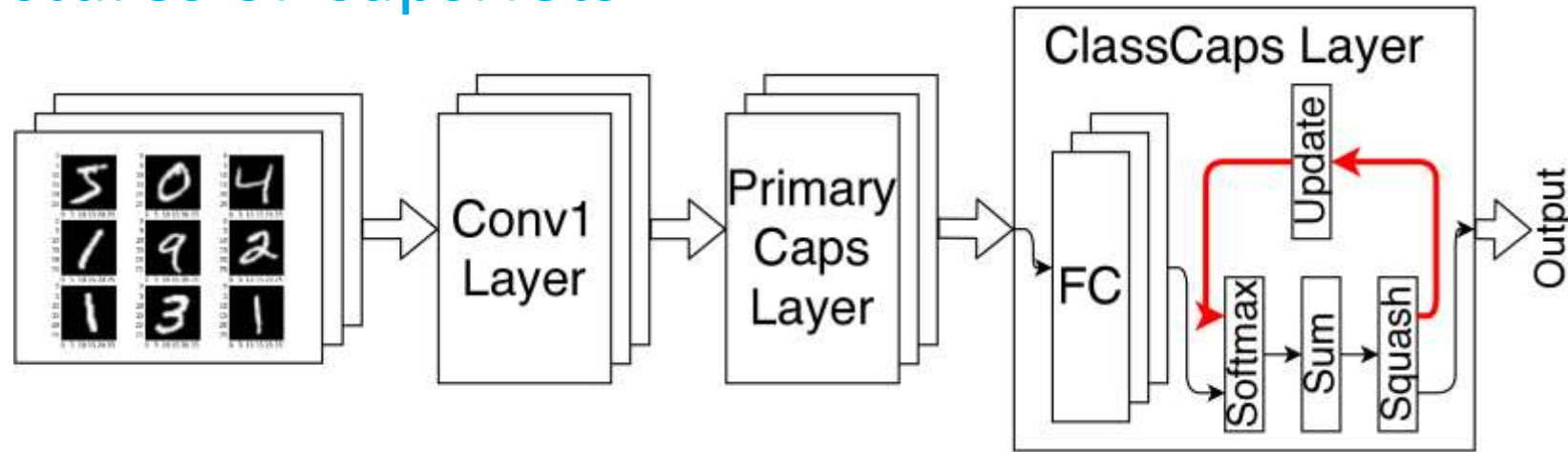
- **Scalar** values
- Weighted sum + nonlinear function
- **Pooling** layers
- Detect **features**

CapsNets

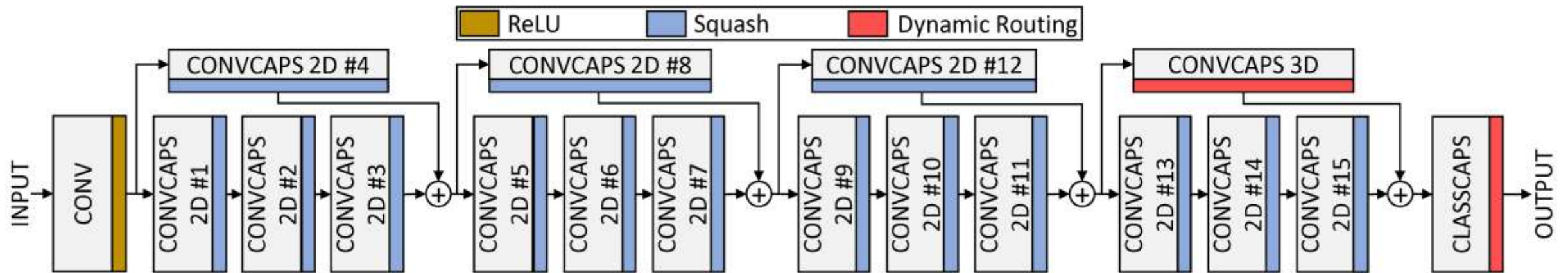
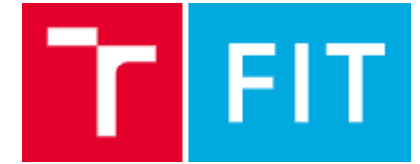


- **Vectors**
- Complex vectorial function (**squash**)
- **Dynamic routing**
- Detect **entities**

Architectures of CapsNets



DeepCaps architecture

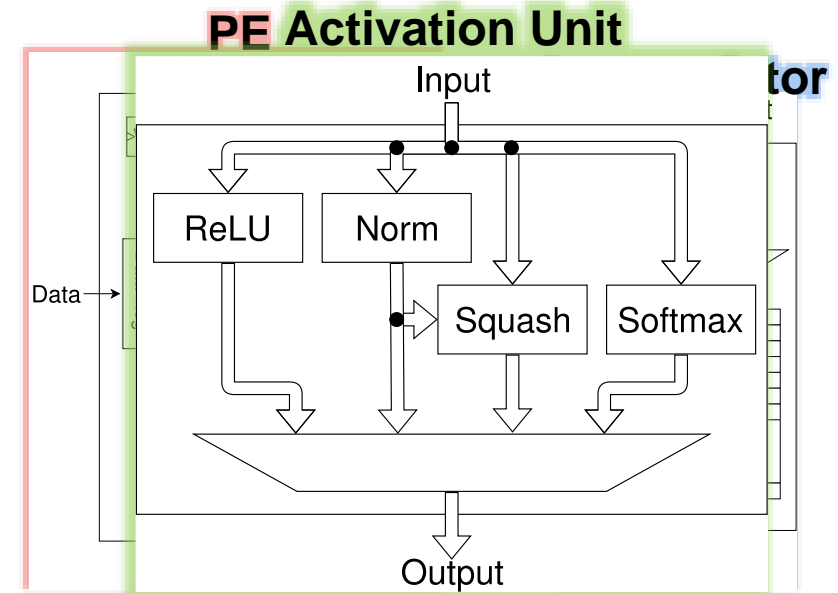
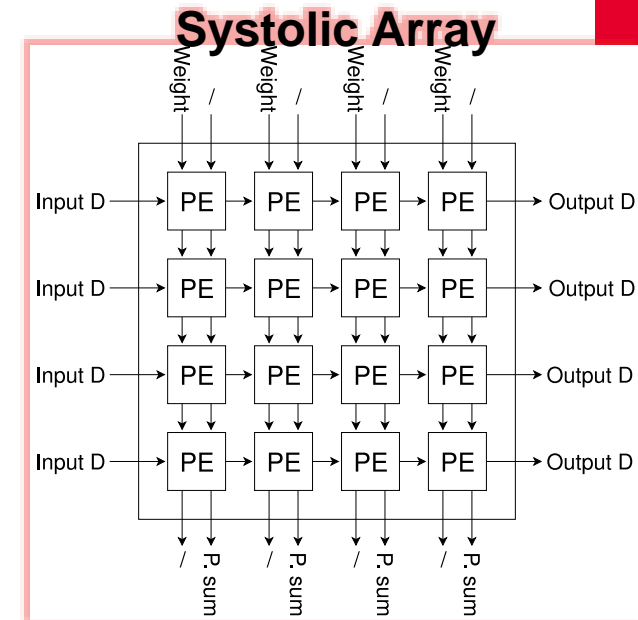
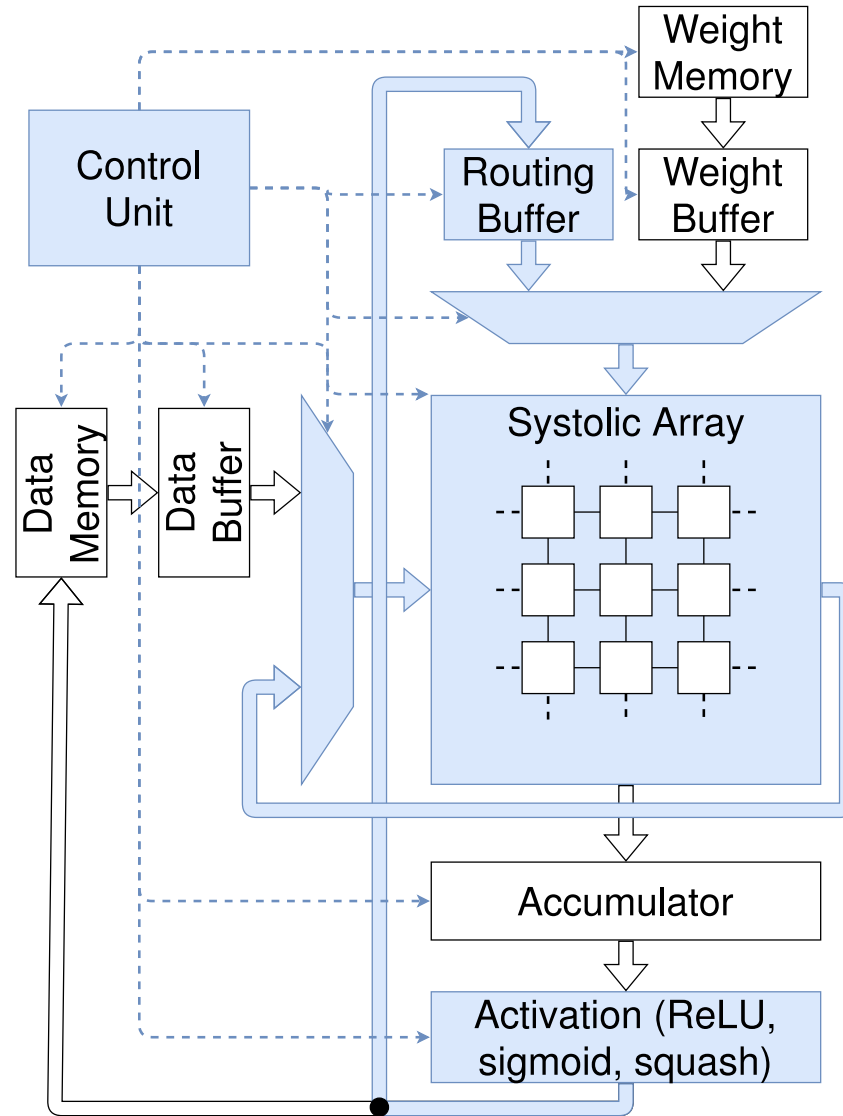


Outline

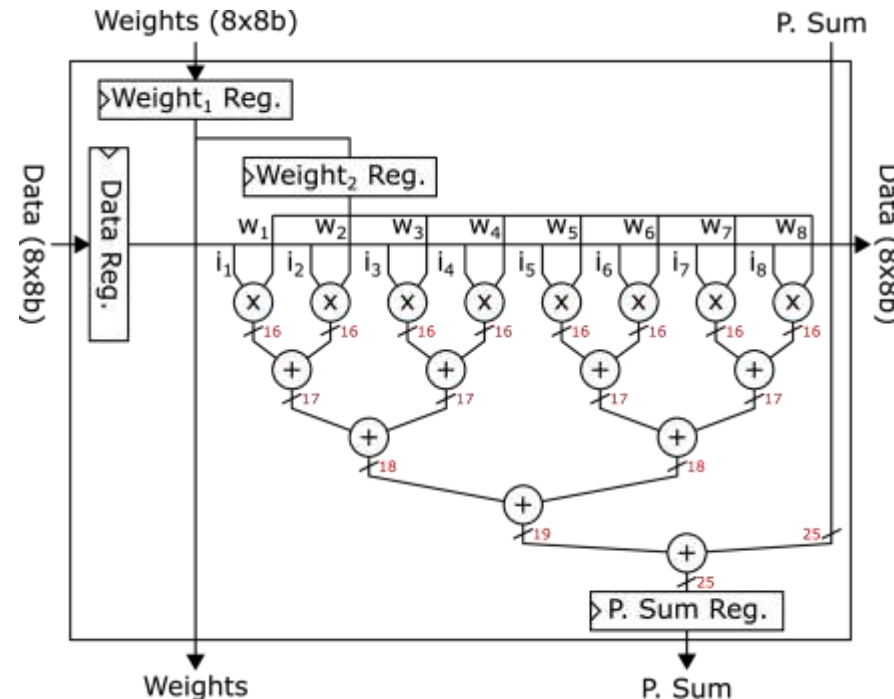
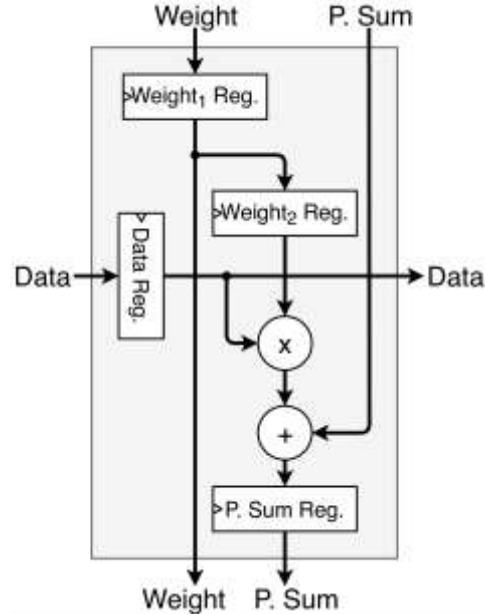


- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

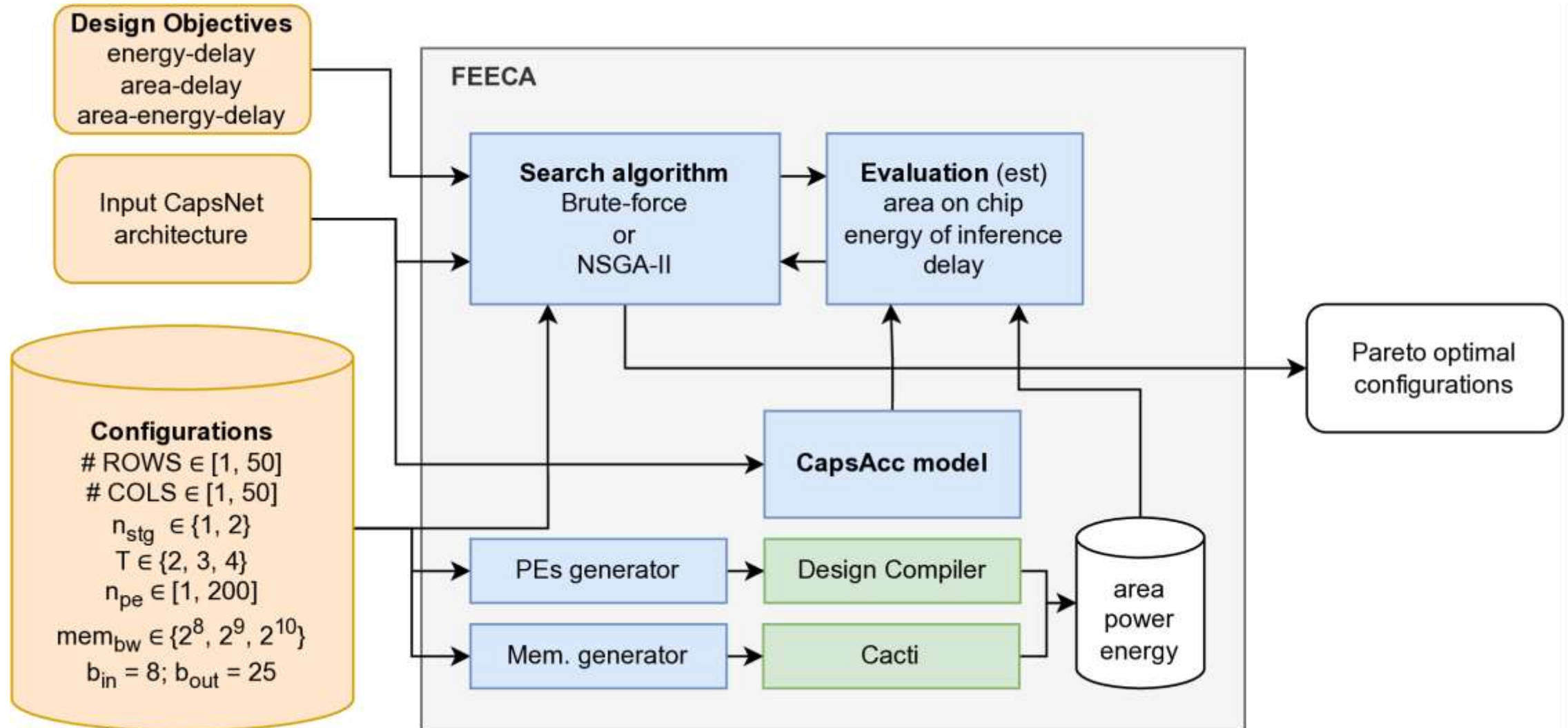
CapsAcc accelerator



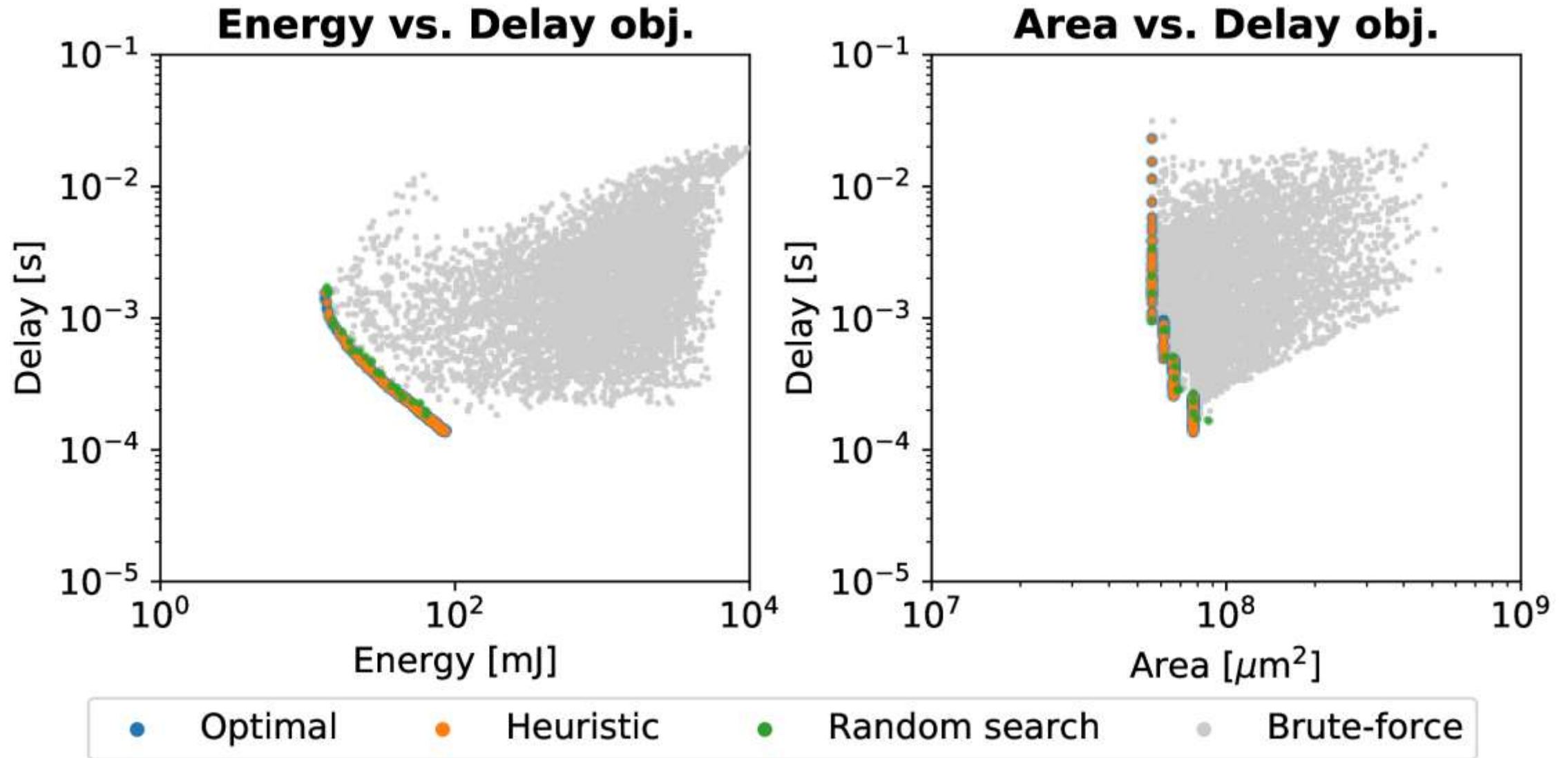
- The goal of this methodology is to optimize parameters of accelerators to get Pareto optimal configurations
 - **Input Parallelism:** Number of input pairs n_{pe} with bit-width b_{in}
 - **Output Precision:** Partial sum bit-width b_{out}
 - **Pipeline Depth:** Number of pipeline stages n_{stg}
 - **Array Dimensions:** PE array rows and columns (#ROWS, #COLS)



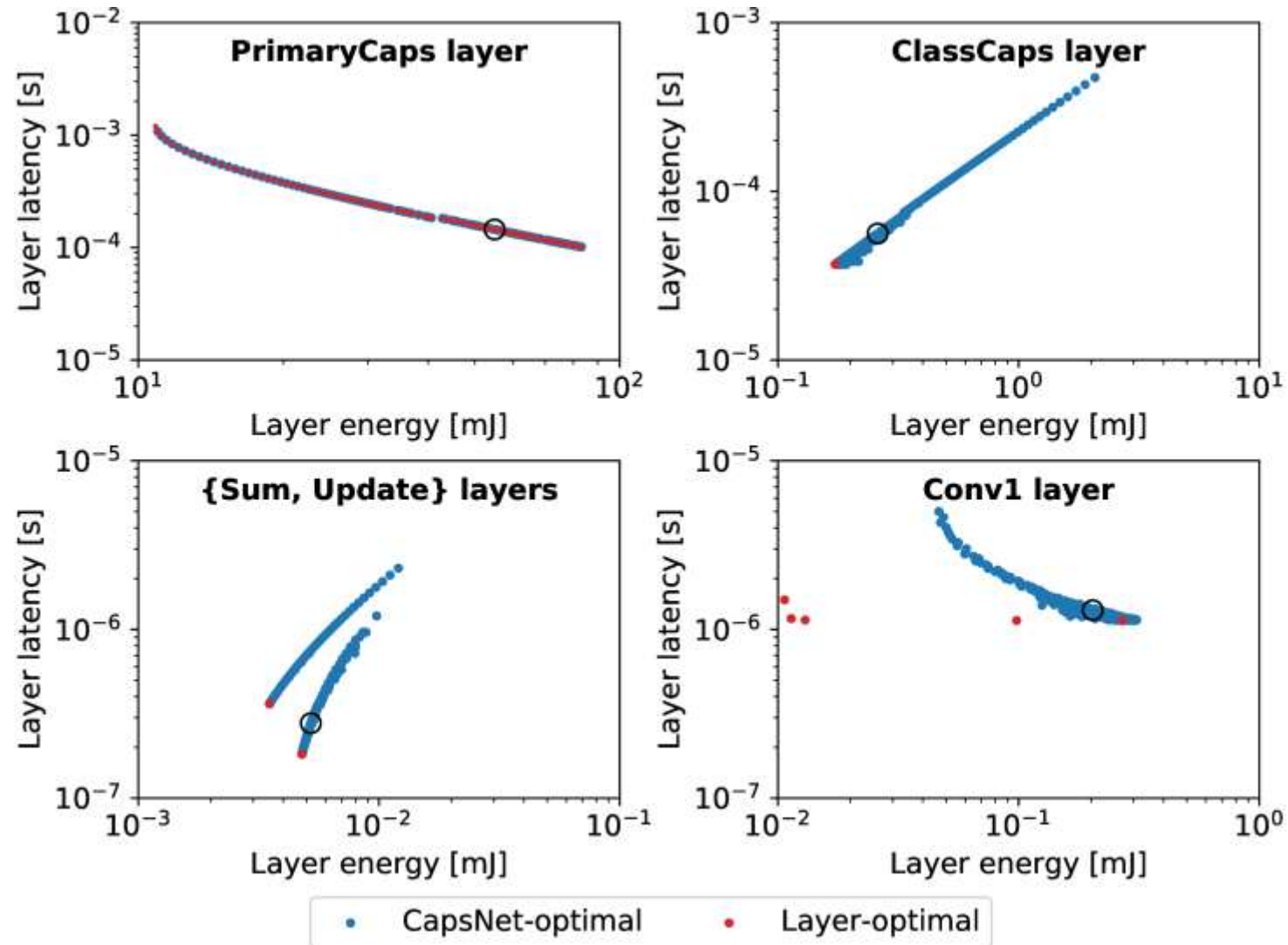
Searching algorithm



Comparison of Brute-force and NSGA-II search



Individual CapsNet layers



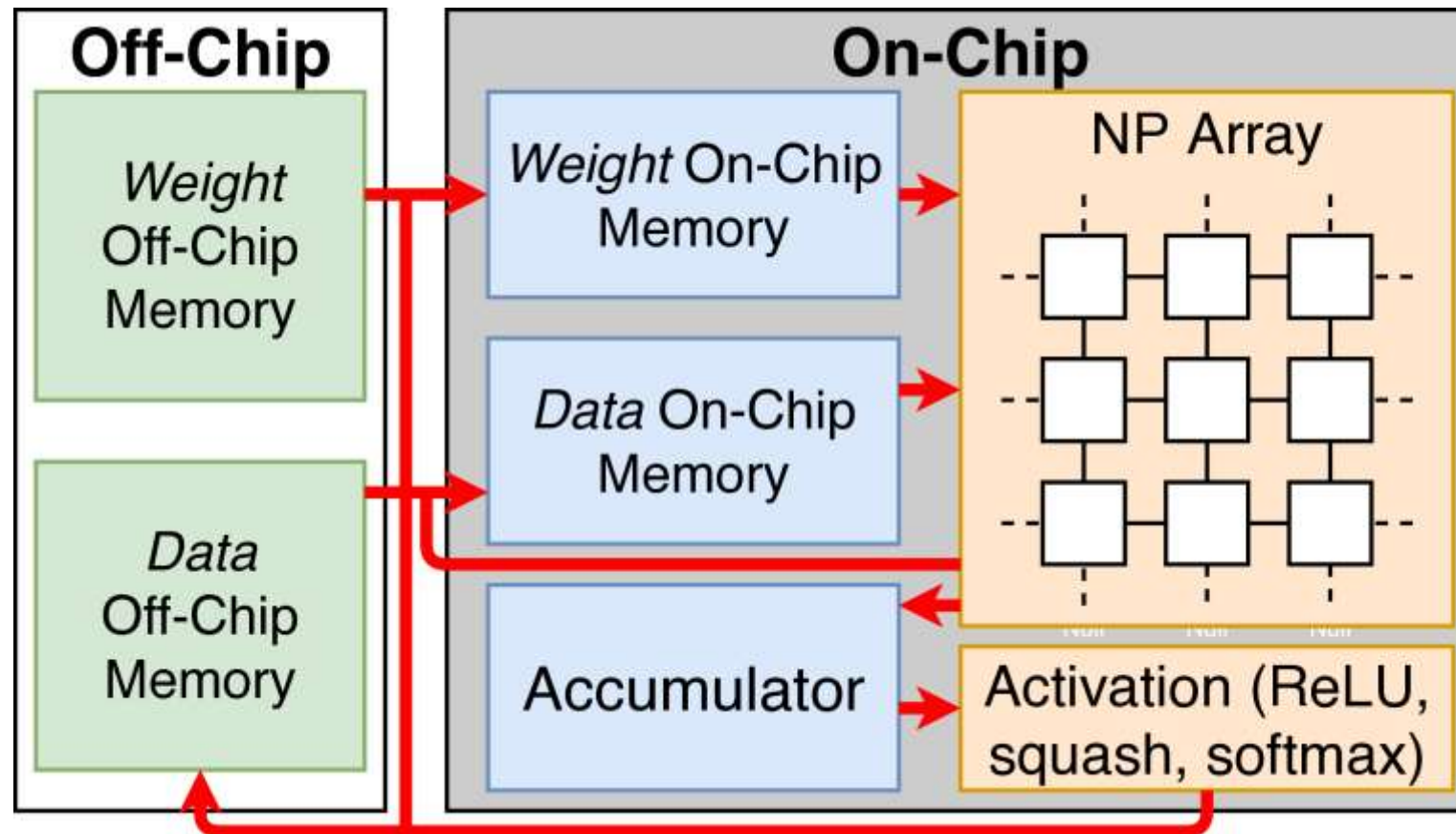
Outline



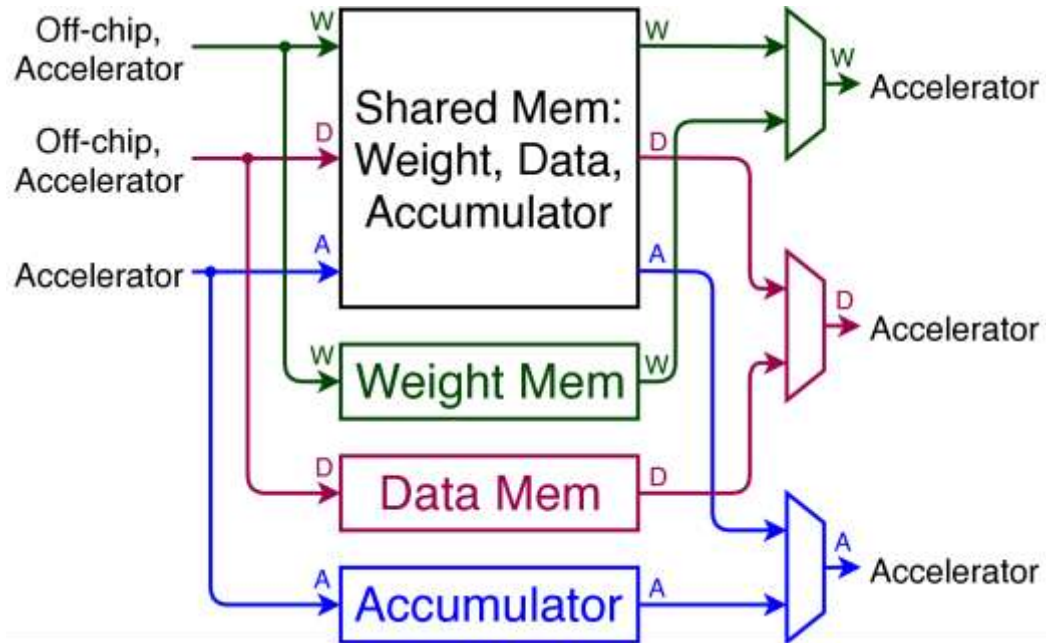
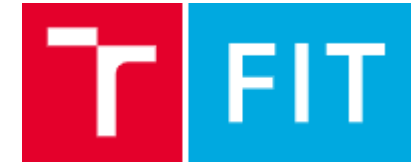
- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - **Memory organization**
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Memory architecture

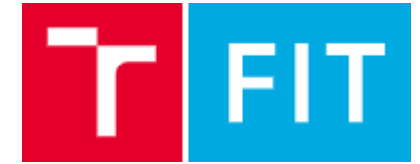
- Advanced architectures employ several types of memories – off=chip and on-chip



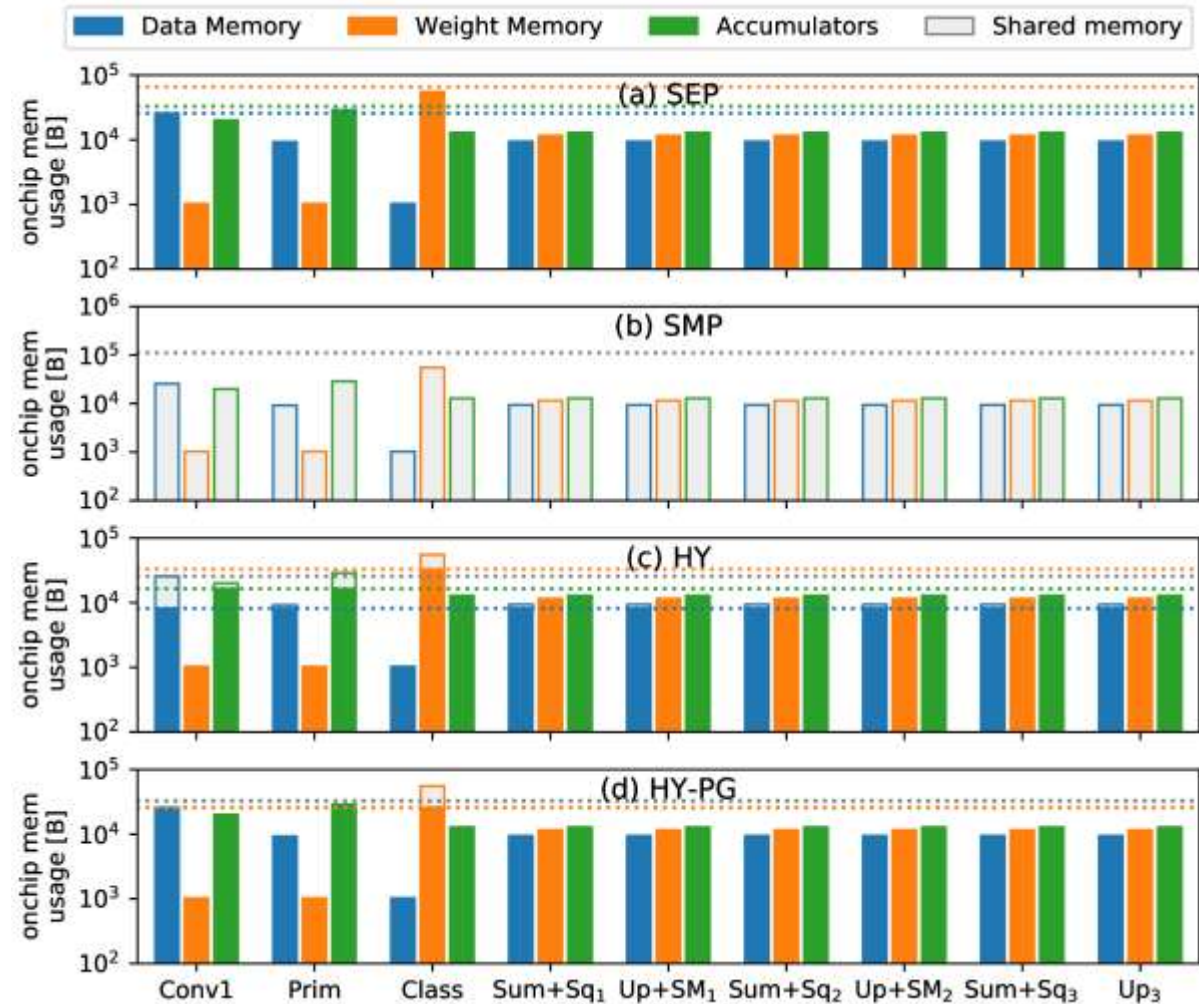
Modification of the ports



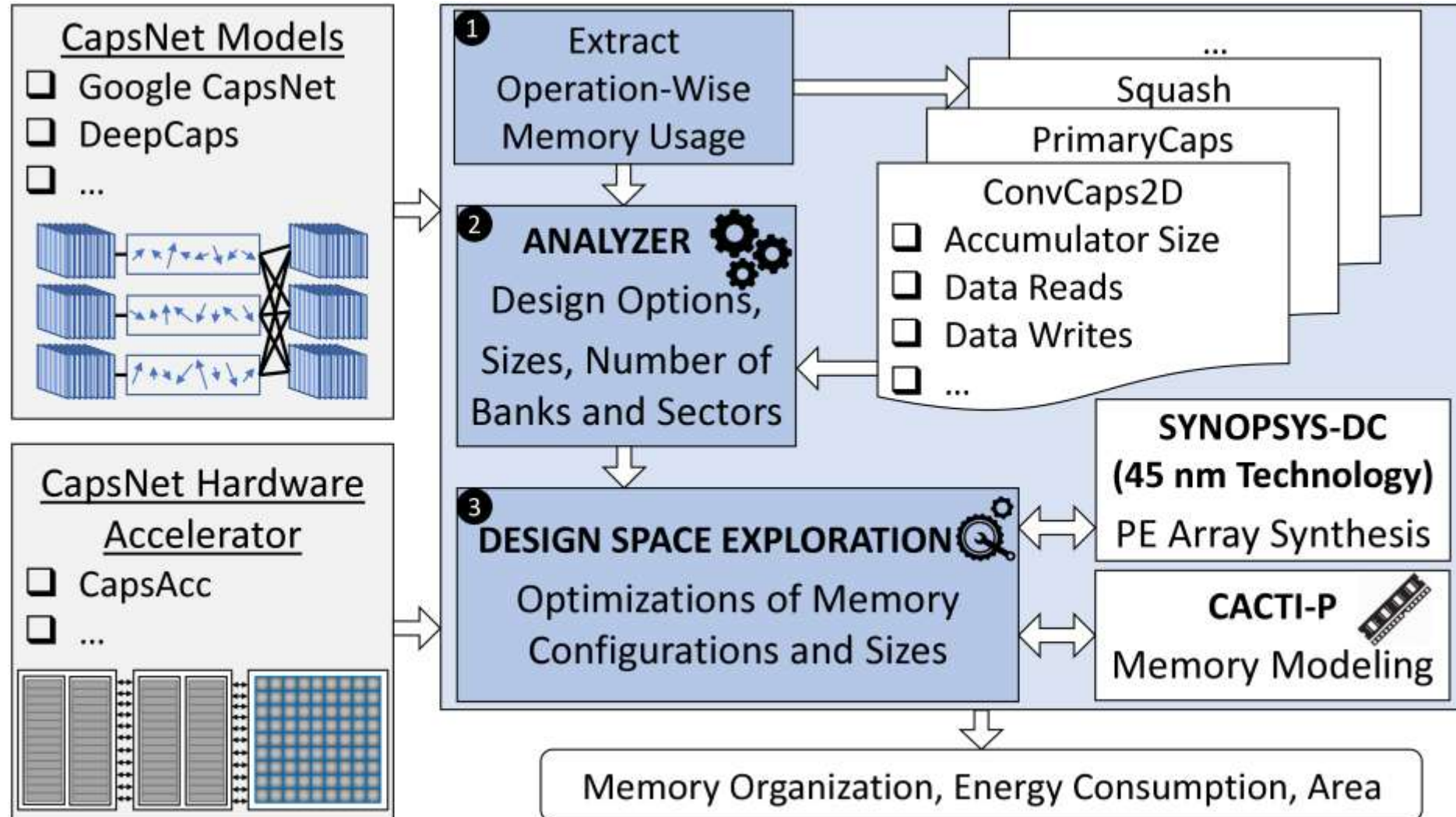
Power savings in memories



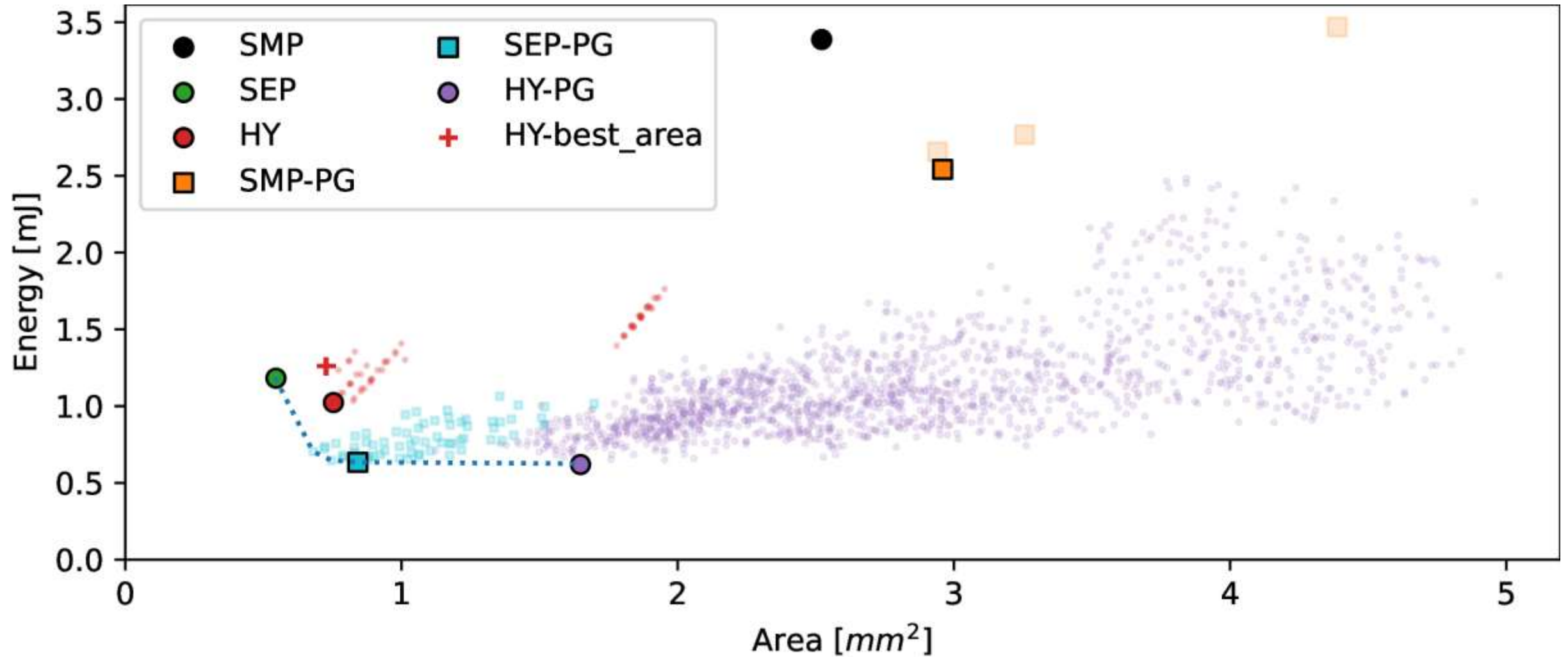
- Using shared buffers.
- Power-gating unused banks vs block size



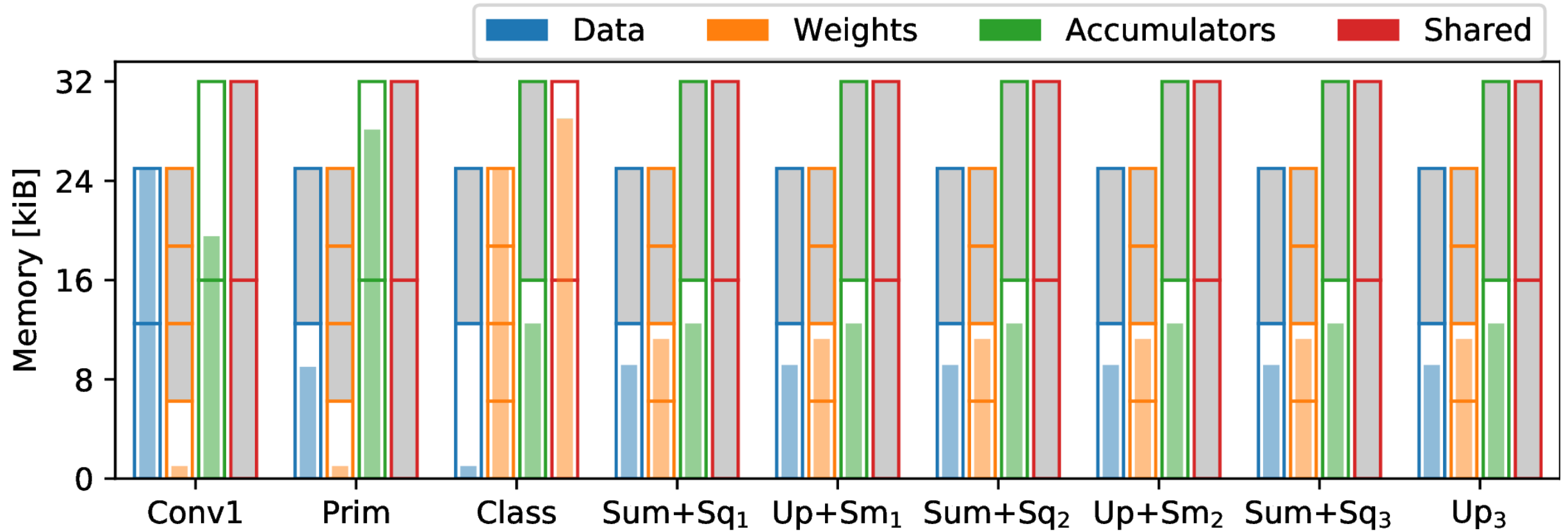
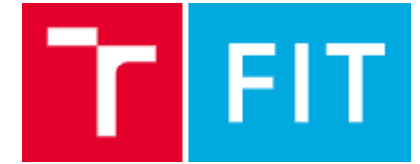
Design space exploration



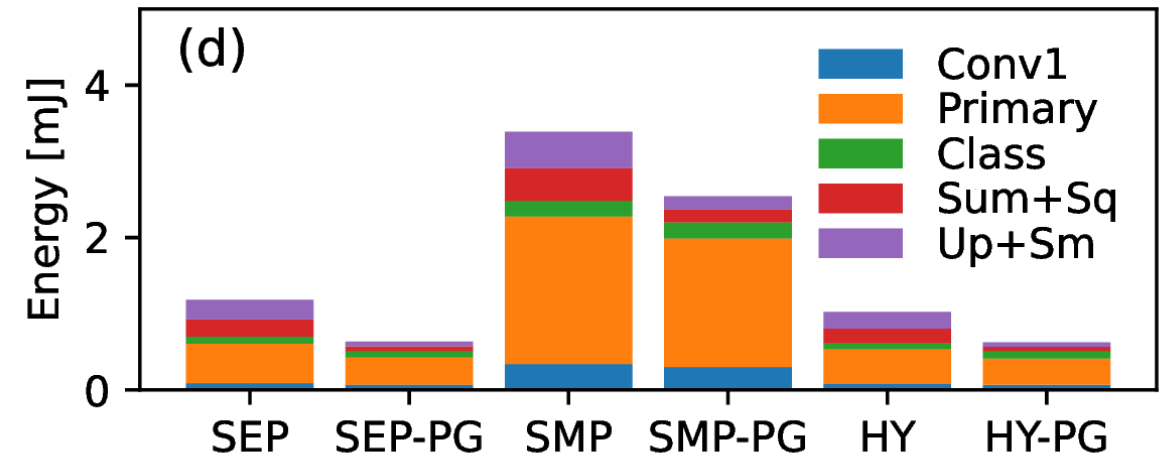
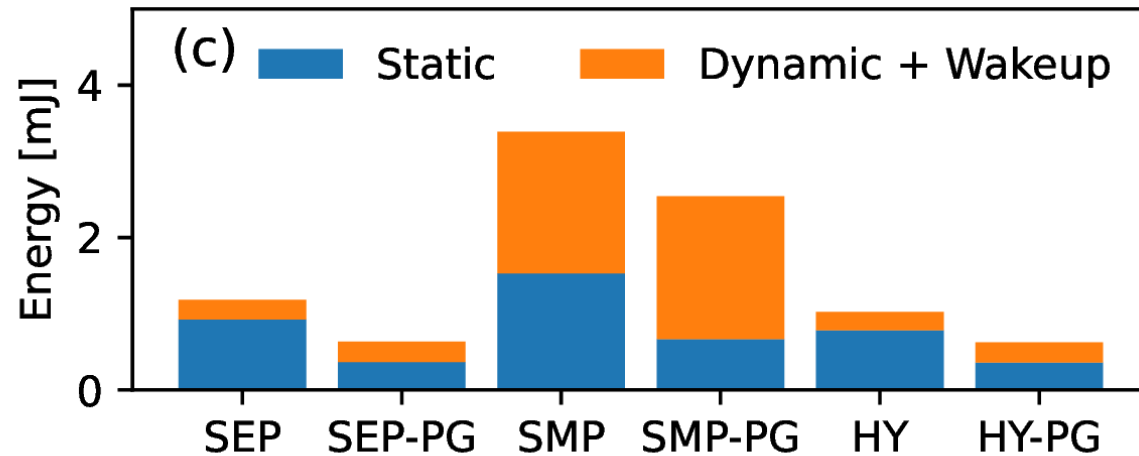
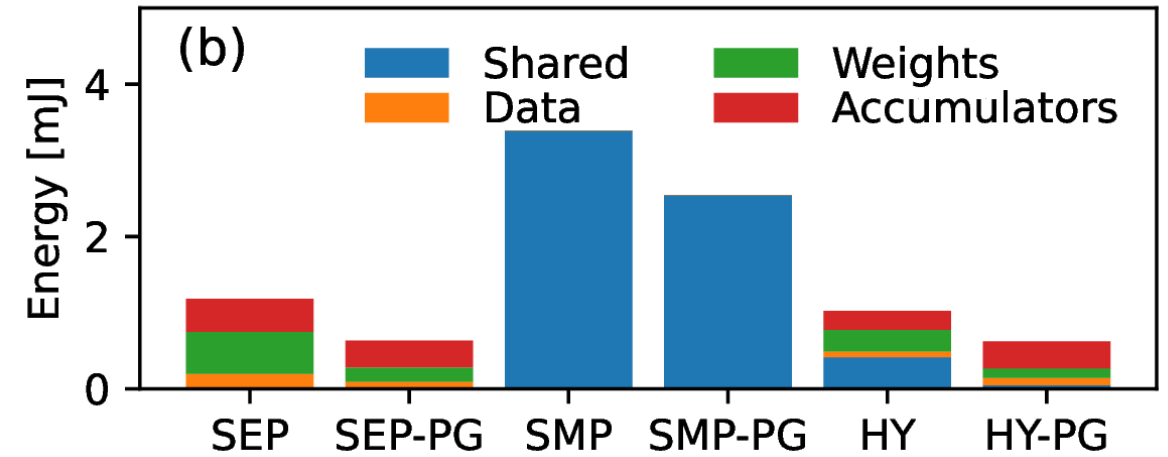
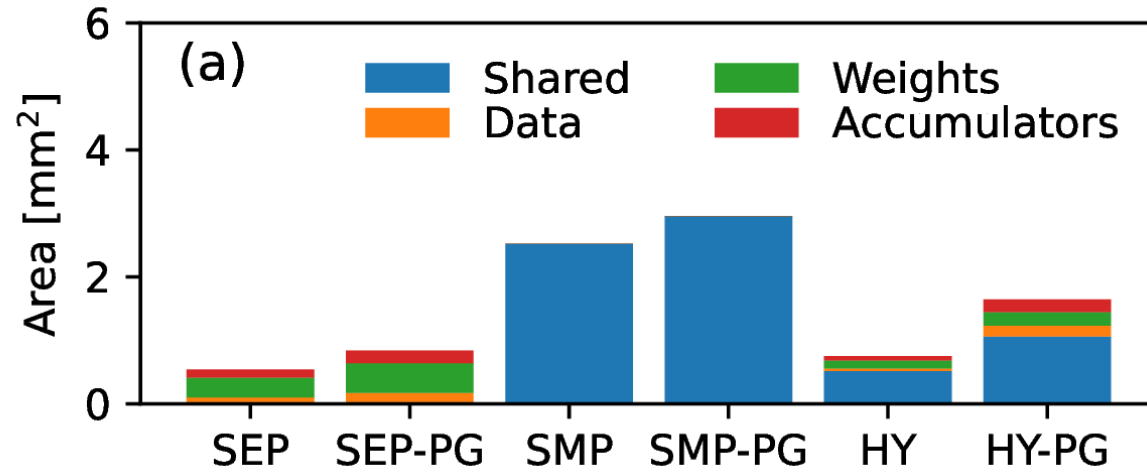
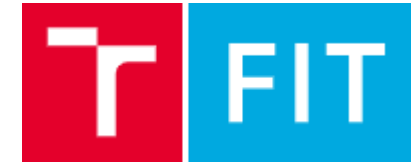
1500 design configurations



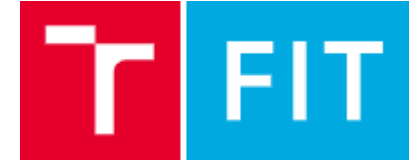
Power gating advances – HY-PG



Parameters of Pareto-Optimal solutions



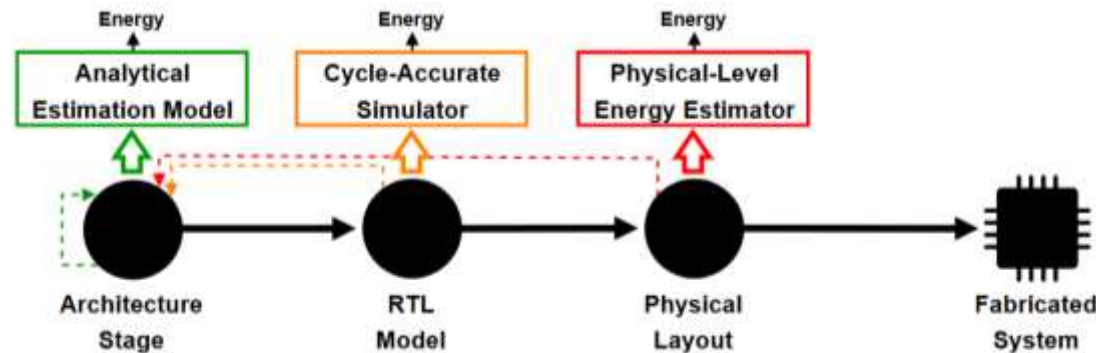
Outline



- Neural networks for embedded systems
- Approximate computing
 - Approximate components
 - Advanced approximation of NN
 - Neural Architecture search
- Hardware accelerator optimization
 - Capsule neural networks
 - Optimization of the accelerator architectures
 - Memory organization
- Execution planning
 - Convolutional networks
 - Transformer networks
- Conclusions

Methods for Modeling AI Accelerators

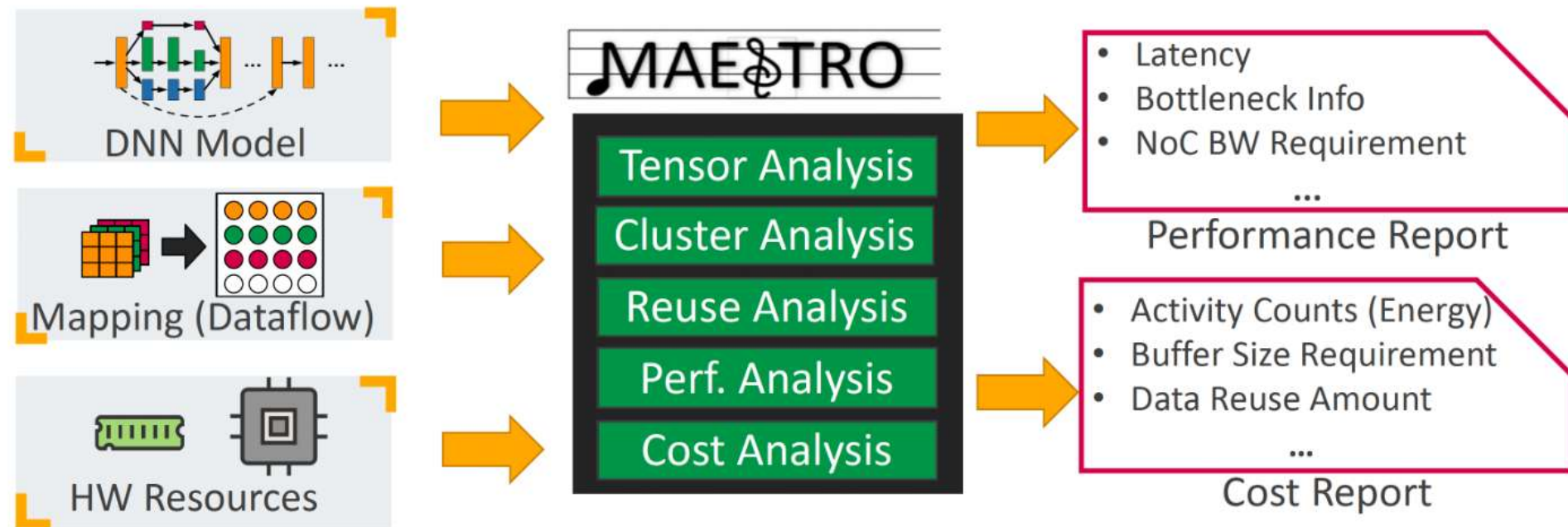
- **Huge accelerator design space** to explore (different systolic array configurations, memory hierarchies, number of ports, memory bus widths and more)
- Reliance on **configurable, flexible, fast**, yet **highly accurate** modeling approaches



However, **existing analytical tools** are **not designed to handle** the **parallelism** and **irregular memory access** patterns of **Transformers**

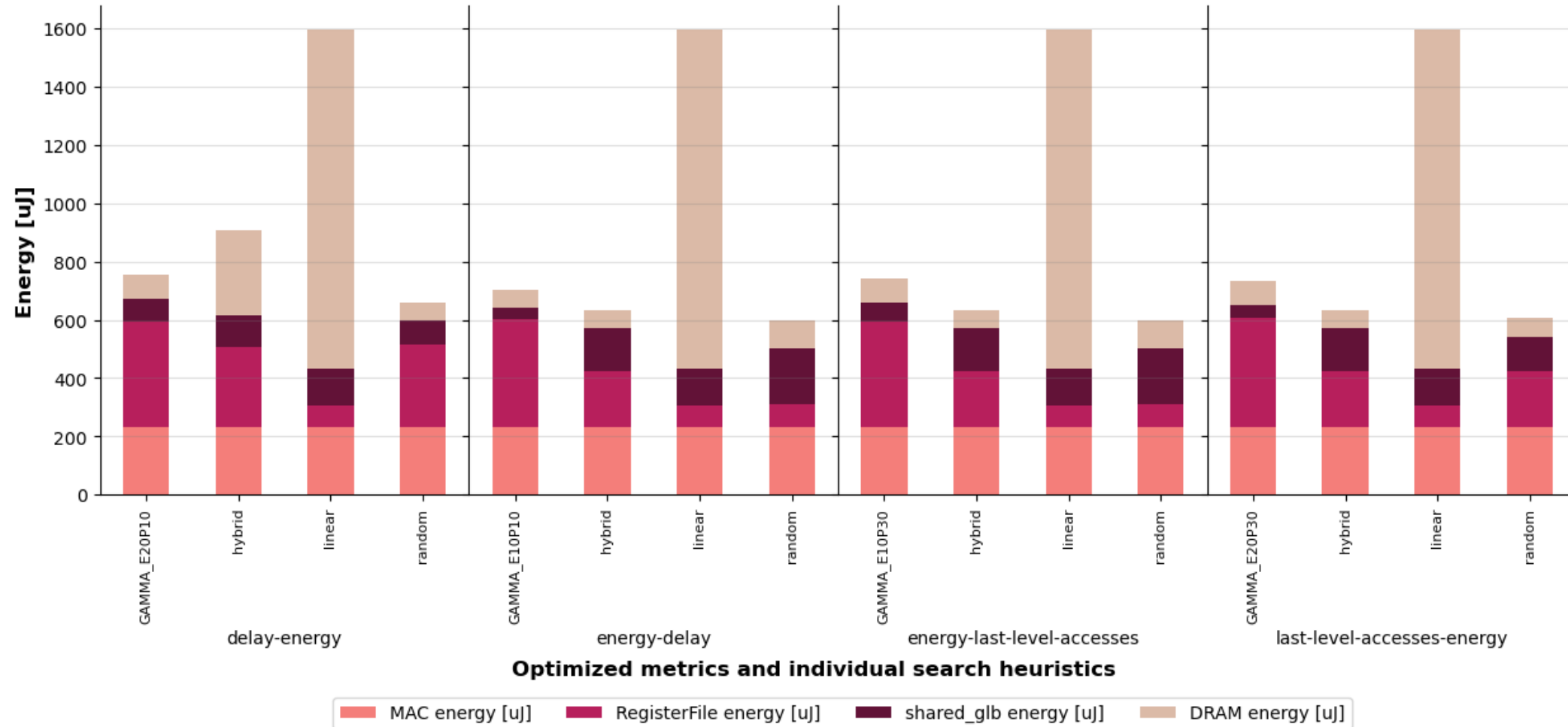
Feature	Roofline	Analytical	RTL Simulation	Estimation	TransInferSim (ours)
Speed (Fast)	✓	✓	✗	✓	✓
Accuracy (High)	✗	✗	✓	✓	✓
ASIC Design Modeling	✗	✗	✓	✓	✓
Memory Access Optimization	✗	✗	✓	—	✓
Executable Operation Plan	✗	✗	✓	✗	✓

- Estimates the power consumption, latency and number of accesses to the memories

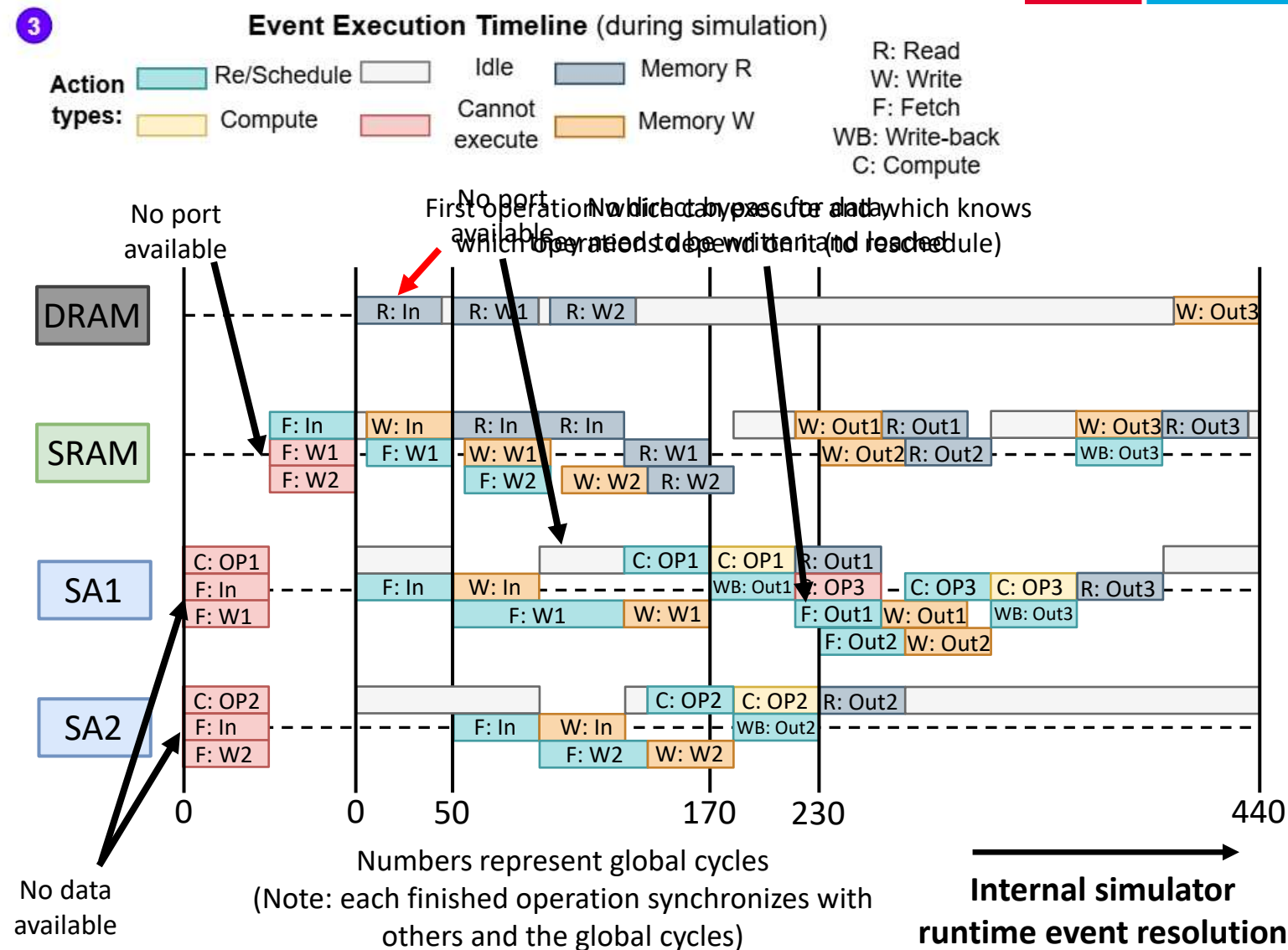
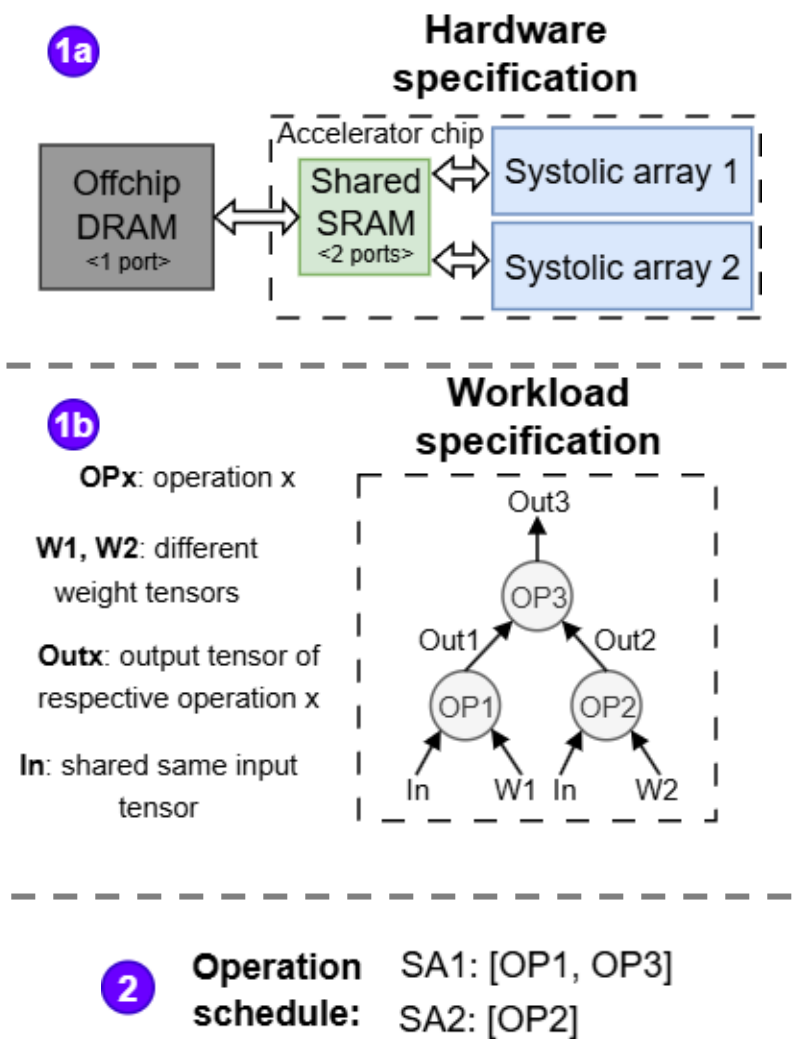


Timeloop: results of mappers

AlexNet CONV layer 1

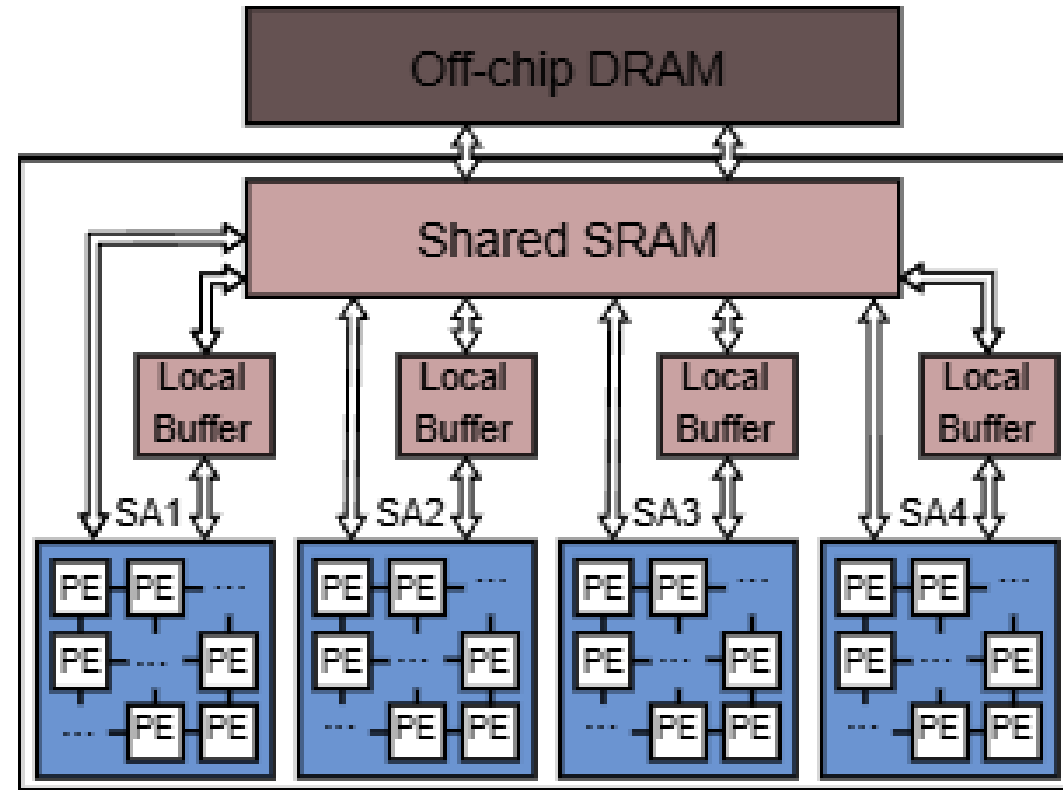


Demonstration of Schedule Execution During Simulation

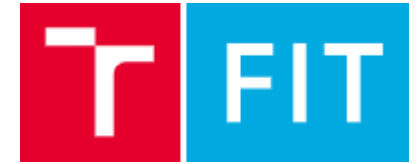


Possible applications

- Advanced planning of the operations
- Design space exploration



Conclusions



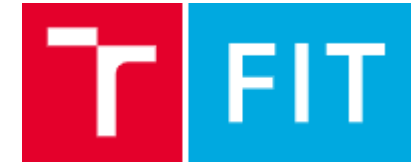
- Deep neural networks are now important in **embedded systems**.
- The NNs exhibit the **error resilience** property.
- An approximation methodology for **computational path** was introduced. For example, 30% energy savings in multiplication leads to 0.6% accuracy drop.
- The proposed methods are available as **open-source** software.



<https://github.com/ehw-fit/evoapproxlib>

<https://github.com/ehw-fit/tf-approximate>

Acknowledgements



Brno University of Technology

- prof. Lukáš Sekanina, assoc. prof. Zdeněk Vašíček
- Ing. Jan Klhůfek

NYU Abu Dhabi

- Muhammad Shafique, Muhammad Abdullah Hanif
- Bharath Srinivas Prabakaran, Alberto Marchisio
- Czech Science Foundation project GA24-10990S
- IT4I Innovation infrastructure



Thank you for your attention

mrazek@fit.vutbr.cz

