Design Space Exploration for ST's Neural Compilation Toolchain

Fabrizio Indirli, *Politecnico di Milano, STMicroelectronics*

Cristina Silvano, *Politecnico di Milano*

Giuseppe Desoli, *STMicroelectronics*

Andrea C. Ornstein, *STMicroelectronics*

IWES 2022 – Bari, Italy – 22/09/2022

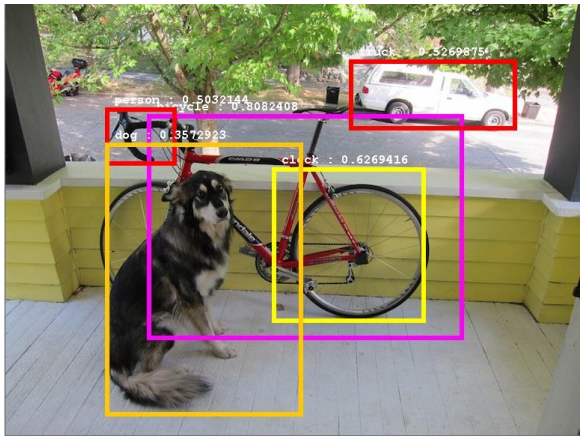
life.augmented

POLITECNICO
MILANO 1863

Introduction

Machine Learning inference on the edge is becoming pervasive for several real-time tasks:

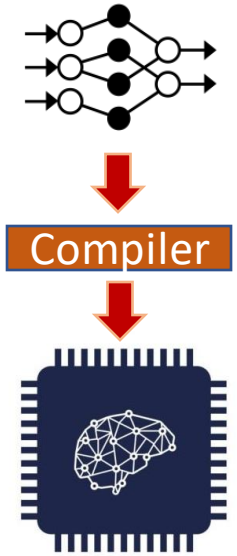
- Cameras: image classification / object detection
- Smart speakers: speech recognition, NLP
- Smart sensors: Anomaly detection, time-series forecasting
- ...



Need to enable efficient NN inference on power-constrained devices through:

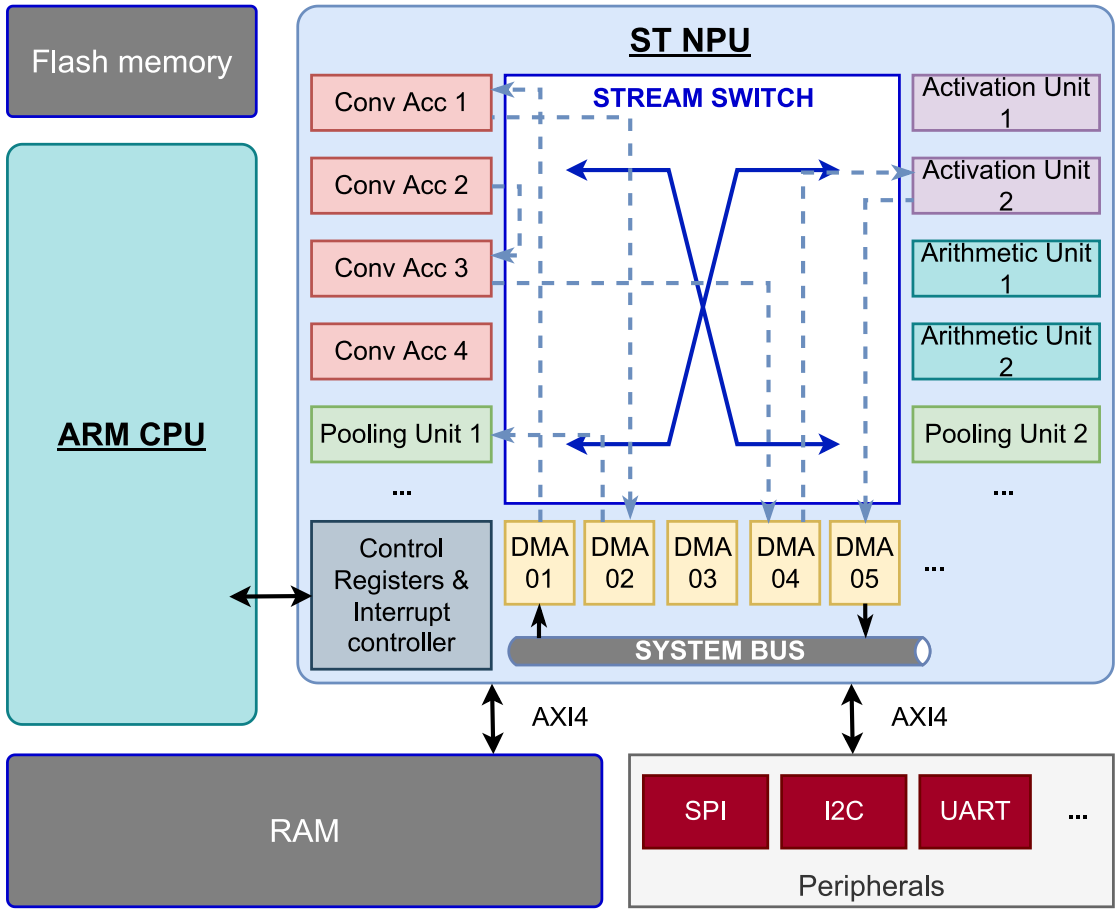
→ **Neural Processing Units Embedded in MCUs and SOCs**

→ **Dedicated compilation toolchains**



STMicroelectronics Experimental NPU

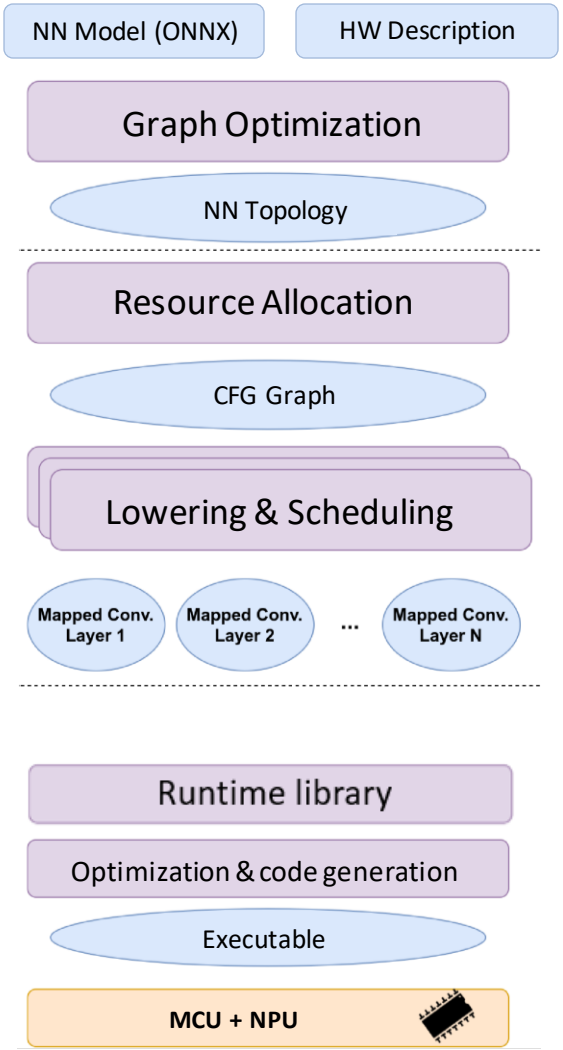
A low-power embedded CNN accelerator to implement a **data-flow based** inference engine. It is designed to be modular and parametric to address a wide spectrum of computational requirements and efficiency needs.



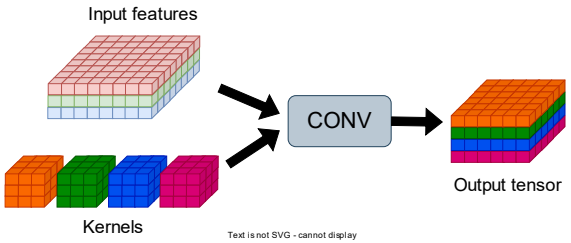
STMicroelectronics Neural Toolchain

To program this NPU, a dedicated compilation toolchain and low-level runtime library are in charge of:

- **Optimizing** the NN model
- **Binding** the NN nodes to the NPU's computational units
- **Scheduling** the operations' across several execution epochs
- **Allocating** the memory buffers
- **Producing** the final code
- **Estimating** runtime metrics:
  - Execution latency (#cycles)
  - Throughput
  - Power consumption
  - Memory footprint



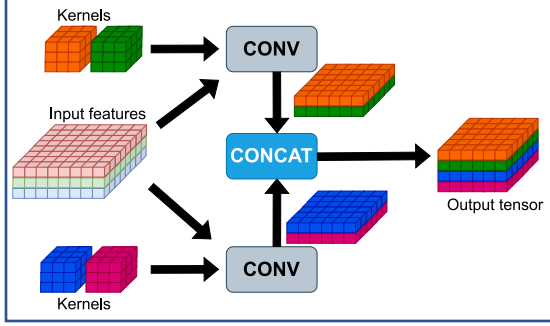
Convolutional layers mapping options



The Neural compiler can apply **optimizations** on the **Convolutional layers**, to use the NPU resources more efficiently. Some of these optimizations passes are:

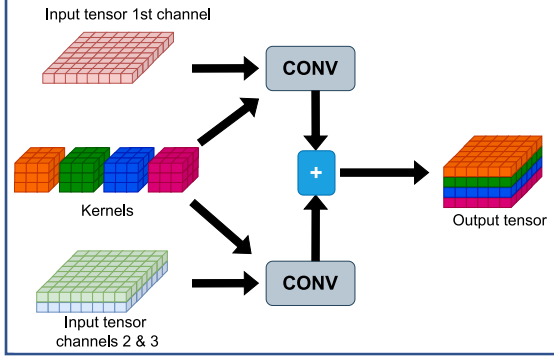
Kernelwise decomposition

Decompose the layer in N parallel convolutions, each of which computes K/N kernels



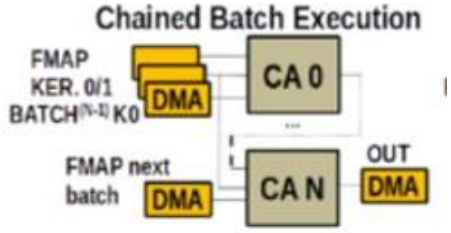
Channelwise decomposition

Decompose the layer in M parallel convolutions, each of which computes C/M input channels, then accumulates the results



Channelwise pipelining

Map the layer on a pipeline of CAs, each of which computes the convolution on C/L input channels and accumulates on the intermediate results.



Optimal mapping and space cardinality

Objectives: Given a set of hardware configurations and a NN model:
find the best hardware configuration and the associated optimal mapping that minimizes a cost function (**power consumption**, **latency**, **memory footprint**).

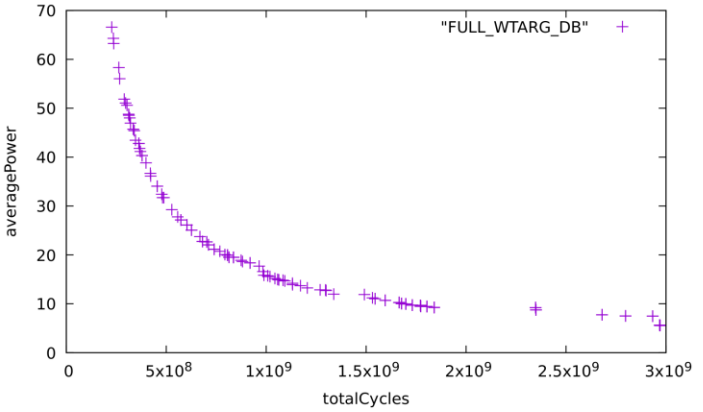
Parameter	Type	Possible values
Number of Conv Accelerators	Generic	1, 2, 4, 8
Split degree in Kernel-wise decomposition	Layer-wise	1 (off), 2, 4
Split degree in Channel-wise decomposition	Layer-wise	1 (off), 2, 4
Max length of ConvAccs pipeline	Layer-wise	1, 2, 3, 4

Exhaustive search not possible for networks with multiple layers:

- With 3 layers: # combinations > 400 000
- With 9 layers: # combinations > 10^{16}

Automatic Exploration techniques needed to:

- Find optimal configurations and/or the Pareto frontiers
- Explore tradeoffs when moving in the design space



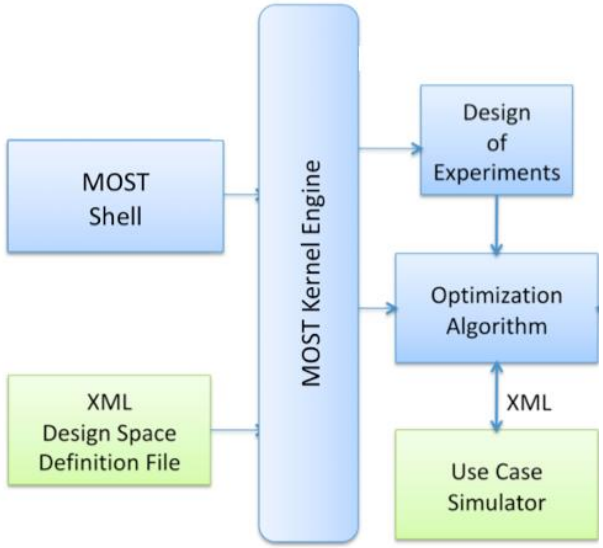
MOST Exploration framework

MOST (Multi-Objective System Optimizer) [5] is an open-source design space exploration tool developed at Politecnico di Milano.

It is an interactive program to **explore a design space** of configurations for a particular architecture for which an executable model exists
(In our work: the **Neural compiler**).

This DSE framework is flexible and modular in terms of:

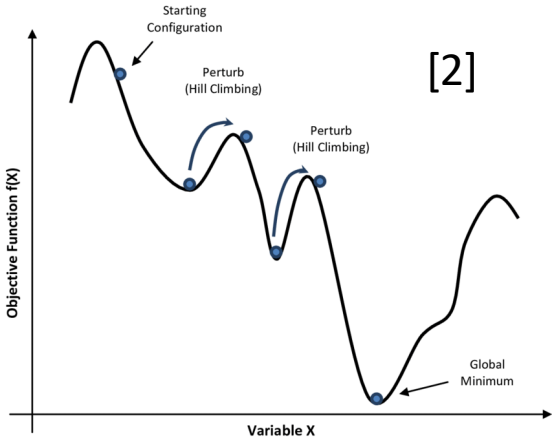
- target architecture
- system-level models and simulator
- optimization algorithms
- Design of experiments algorithms
- system-level metrics



[5] github.com/vzaccaria/most

Exploration algorithms (1)

- **Full search** (baseline): Exhaustive search that finds the optimal solution in exponential time (only for small networks)
- **Search with one-fits-all-layers pruning**: Prune the design space by considering only the configurations in which the layer-wise parameters are identical for all the layers
- **MOSA (Multi-Objective Simulated Annealing)** with Face-Centered Central Composite initialization: initially, a set of points is generated including full factorial designs, center points, and face-centered axial points.
  Starting from this initial set, the Simulated Annealing search [1] is performed for several epochs.
  In each epoch, new configurations are constructed by imposing a random displacement, and they are evaluated:
  - They are accepted if they show an improvement;
  - Otherwise, they get randomly accepted or rejected according to a probability distribution with the acceptance probability decreasing in later epochs.

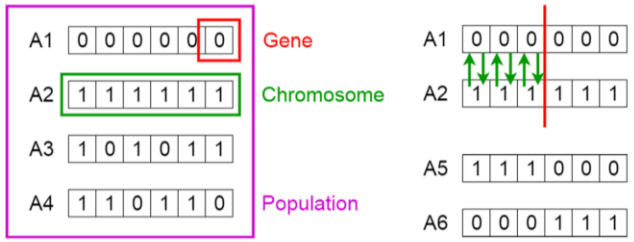


[2]

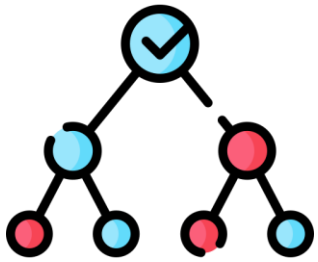
[1] K. I. Smith et al., "Dominance-based multiobjective simulated annealing," IEEE TECV, vol. 12, no. 3, pp. 323–342, 2008.
[2] Picture from O. Ghasemalizadeh et al. "A Review of Optimization Techniques in Artificial Networks", International Journal of Advanced Research, 2016

Exploration algorithms (2)

- **NSGA-II** (Non-Dominated Sorting Genetic Algorithm II [3]) with random initialization:
  at the beginning, a parent population is generated **randomly**. Each point of the current population is evaluated and gets assigned its non-domination level, which serves as fitness function.
  The lowest-levels fronts (sets of points) are selected to build the next generation through recombination, mutation and elitist cloning. The last front is partitioned based on crowding distance.

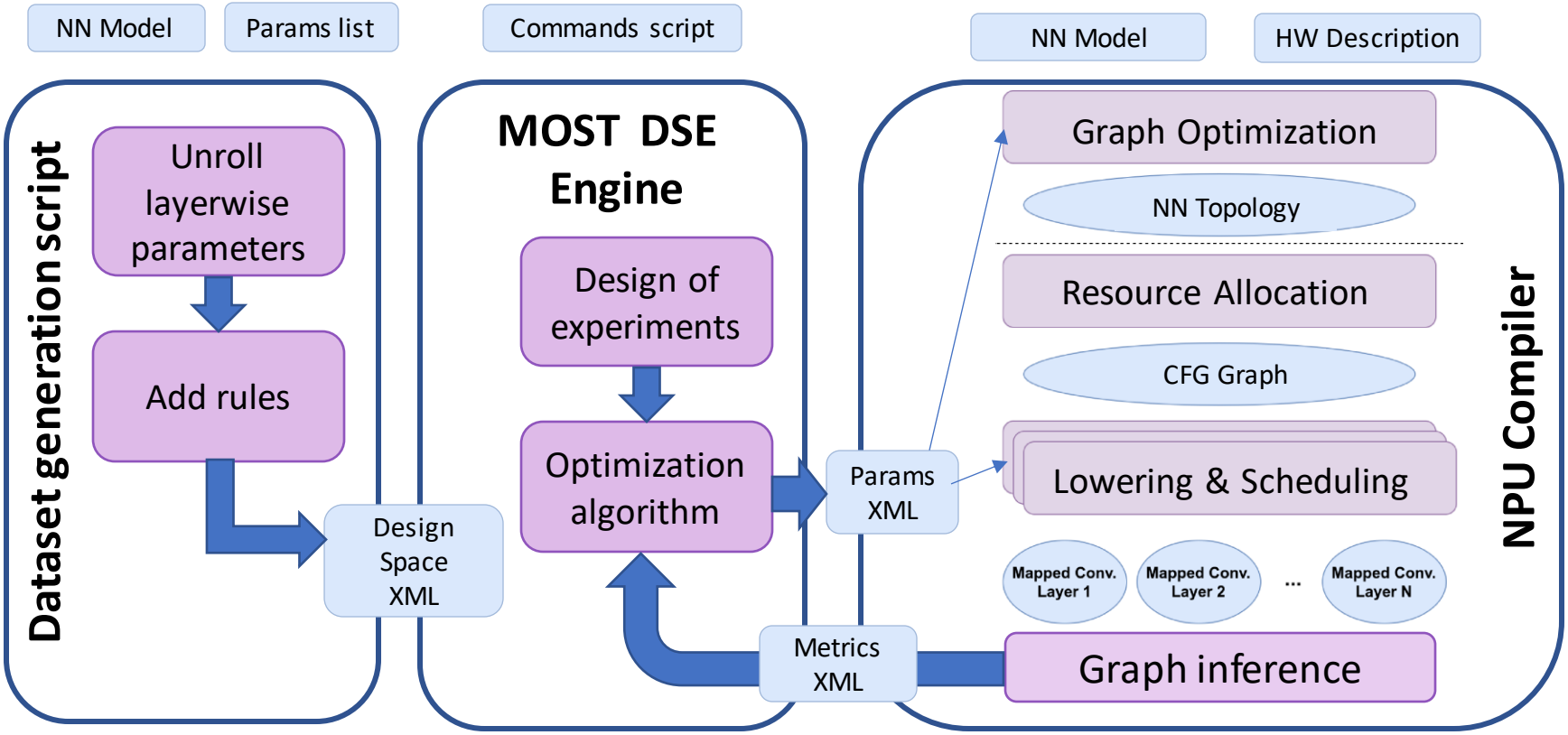


- **Greedy Exploration**: first, an initial set of points is generated randomly or through the Face-Centered Central Composite design of experiments. Then, the algorithm starts to greedily move within the design space through neighborhood points trying to minimize a single objective, and it stops once a local minimum is found.

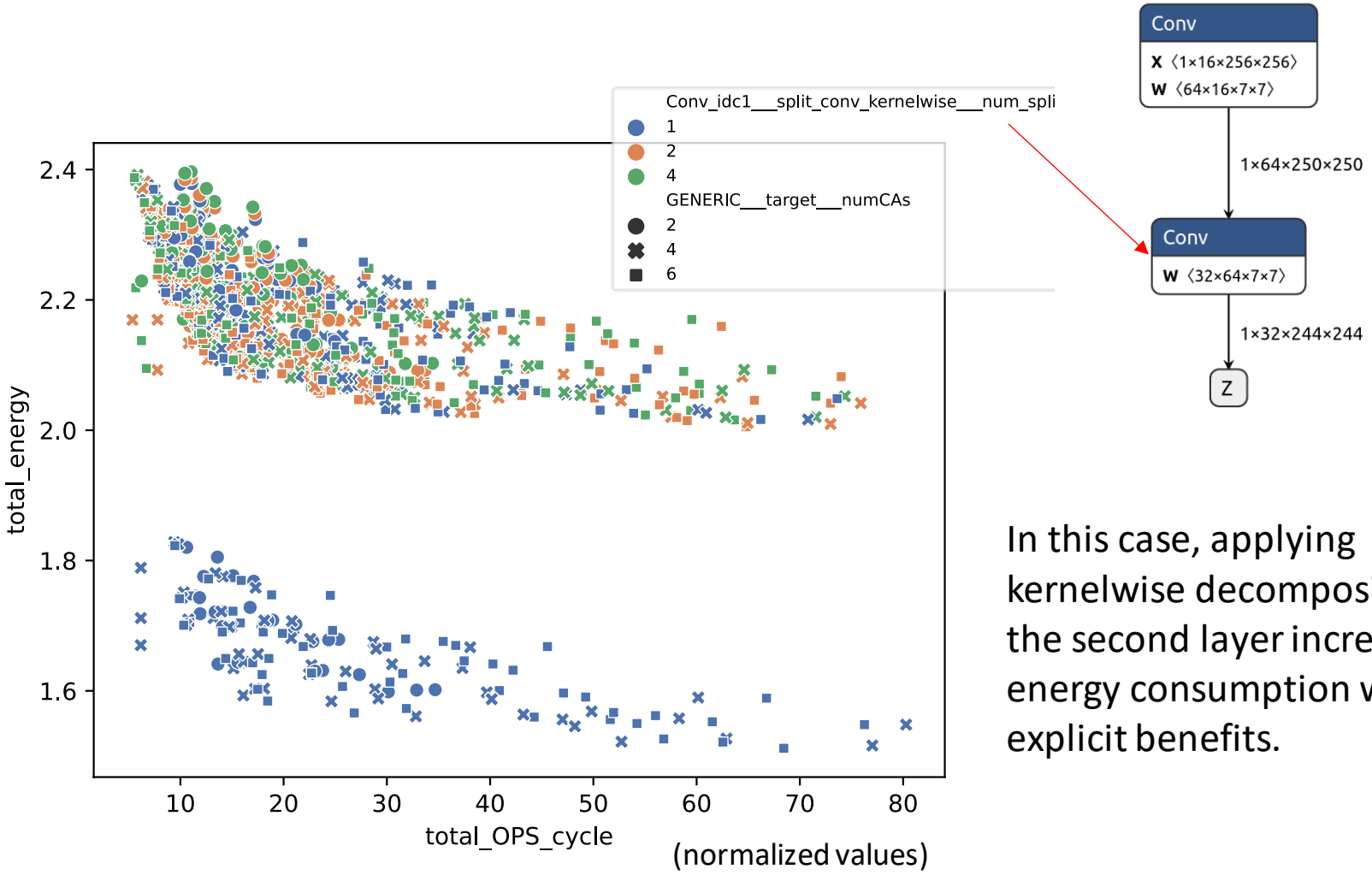


[3] K. Deb et al., "A fast and elitist multiobjective genetic algorithm: NSGA-II," IEEE TEVC, vol. 6, no. 2, pp. 182–197, Apr. 2002

Integrating the NPU Compiler with MOST



Exhaustive search on 2-layer model: Total energy consumption vs OPS/cycle



In this case, applying kernelwise decomposition on the second layer increases the energy consumption without explicit benefits.

Comparison with full exhaustive search

How can we **compare** the approximate sets produced with MOSA/NSGA-II to the exact Pareto Set (Π) that was found with the Full Exhaustive search? An useful metric is the **Average Distance from Reference Set (ADRS) [4]:**

$$\mathrm{ADRS}(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{\boldsymbol{x}_R \in \Pi} \left(\min_{\boldsymbol{x}_A \in \Lambda} \{\delta(\boldsymbol{x}_R, \boldsymbol{x}_A)\} \right)$$

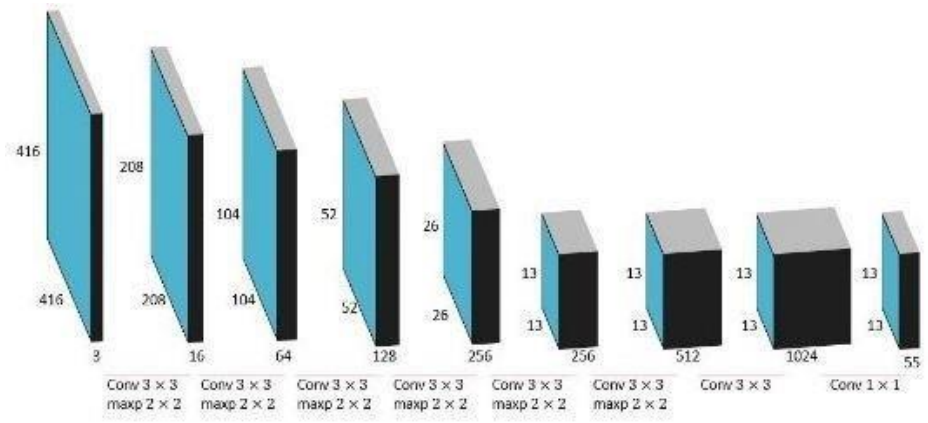
The ADRS is usually measured in terms of percentage; **lower is better.**

Algorithm	# Points	ADRS
Full Search	1864	Ref.
MOSA	1001	3,4 %
NSGA-II	129	17,2 %

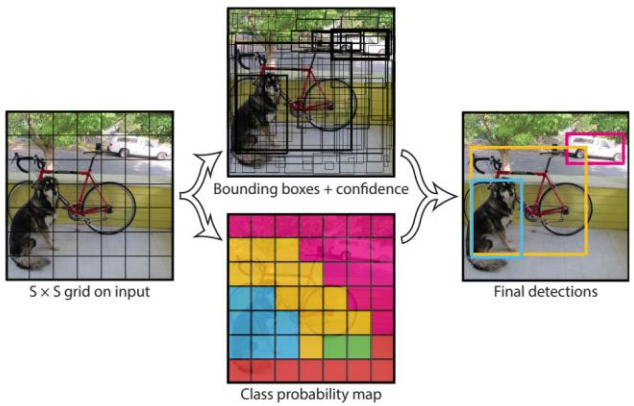
[4] ReSPIR: A response surface-based pareto iterative refinement for application-specific design space exploration, *Palermo G, Silvano C, Zaccaria V,* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Tiny-Yolo-v2

This model is a real-time neural network for **object detection** that detects 20 different classes. It is made up of 9 convolutional layers and 6 max-pooling layers and is a smaller version of the more complex full YOLOv2 network.

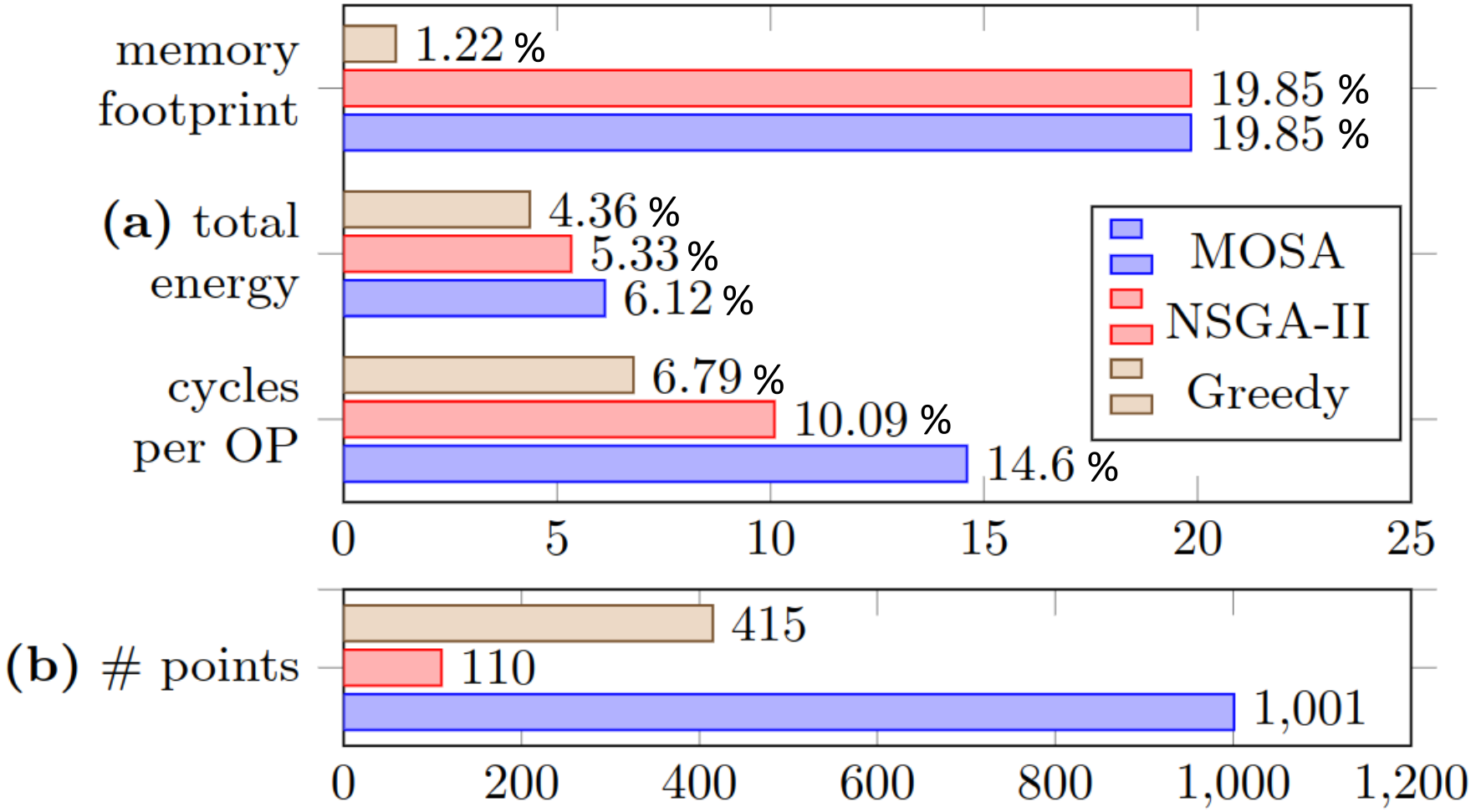


Metric	Value
Type	Detection
GFLOPs	5.424
MParams	11.229
mAP	29.11%
In img shape	3x416x416



Exploration algorithms comparison

% reduction in each objective metric, compared to the best results obtained with one-fits-all-layers pruning



Baseline: one-fits-all exploration (with #points in : 144)

Conclusions and future works

Conclusions:

- we have described **the HW architecture of ST experimental NPU** and its compilation toolchain for embedded ML
- we have integrated a **DSE engine** with the NPU compiler and evaluated several **exploration methodologies** to efficiently find the optimal mapping of DCNNs on the NPU
- the MOSA exploration algorithm yields the best results at the cost of a longer exploration time, while the NSGA-II is faster

Future works:

- Evaluate the optimizations order as an additional parameter to explore
- Extend the exploration methodology to more parameters
- Implement a 2-steps hierarchical exploration to first prune the search space
- Compare more optimization algorithms, evaluate ML-based autotuning

Fabrizio Indirli
fabrizio.indirli@polimi.it

Thank you for your attention!

Questions?