Neural Architecture Search for low-power MCUs

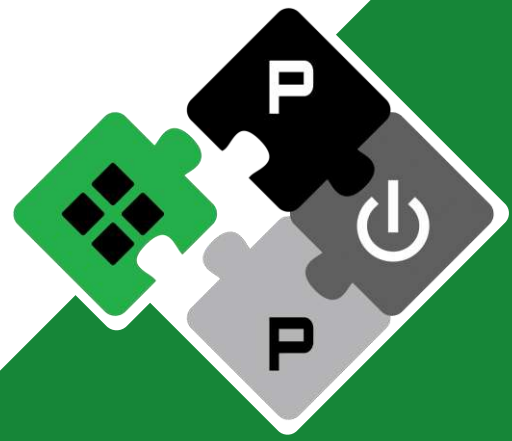
Alessio Burrello alessio.burrello@unibo.it

alessio.burrello@polito.it

Daniele Jahier Pagliari

Matteo Risso

Beatrice Alessandra Motetti



@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

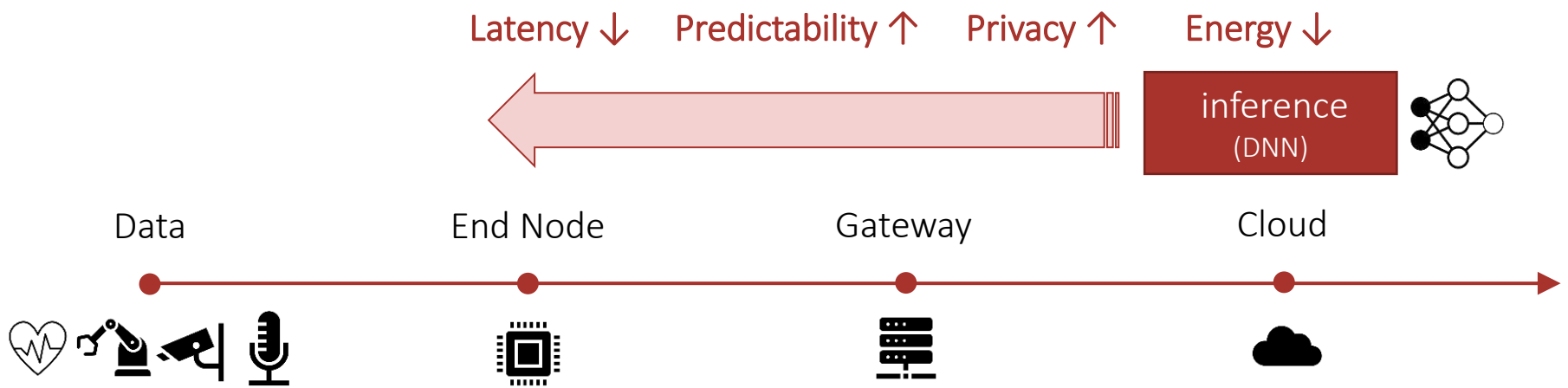
PULP Platform

Open Source Hardware, the way it should be!

DNNs at the Extreme Edge

- Near-sensor DNN inference has several potential benefits w.r.t. a traditional cloud-centric approach:
  1. More predictable and lower (*) latency
  2. Data privacy
  3. Lower energy consumption (*)

(*) possibly



Expectation

The drone follows the head of the human

Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs

E. Cereda, L. Crupi, M. Risso, A. Burrello, L. Benini, A. Giusti, D. Jahier Pagliari, and D. Palossi

Cereda, Elia, Luca Crupi, Matteo Risso, Alessio Burrello, Luca Benini, Alessandro Giusti, Daniele Jahier Pagliari, and Daniele Palossi. "Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs,", ICRA, 2023

Reality

Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs

E. Cereda, L. Crupi, M. Risso, A. Burrello, L. Benini, A. Giusti, D. Jahier Pagliari, and D. Palossi

Cereda, Elia, Luca Crupi, Matteo Risso, Alessio Burrello, Luca Benini, Alessandro Giusti, Daniele Jahier Pagliari, and Daniele Palossi. "Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs,", ICRA, 2023

Reality → Expectation

What changed from reality to expectation? Neural Architecture Search

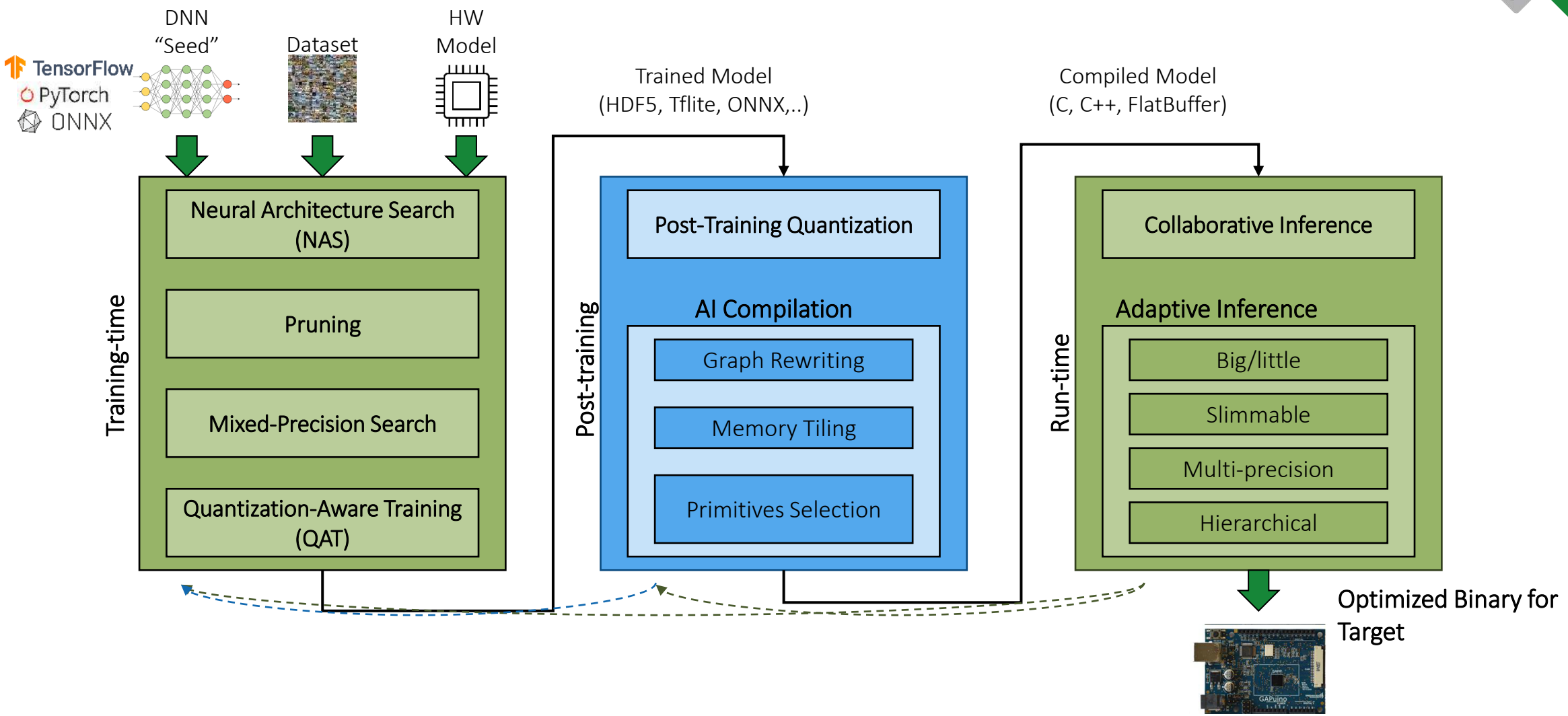
Frontnet [2]
#Params: 304k
#MACs/inference 14.7M
Max throughput 45.3 FPS
MAE→ x-axis 0.33
y-axis 0.12
angle 0.77
Mission → **Failed**

NAS →

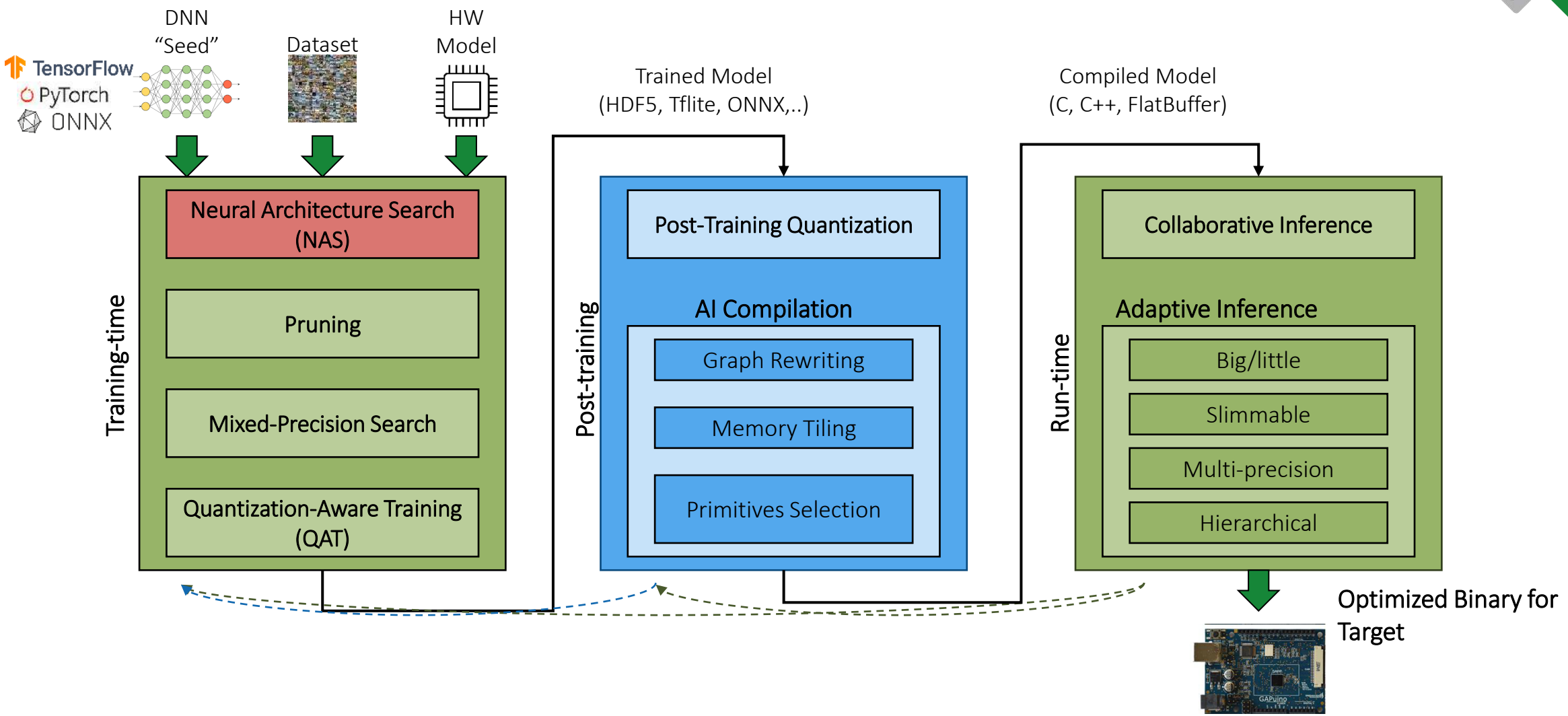
NAS network [1]
#Params: 65k
#MACs/inference 7.4M
Max throughput 51.2 FPS
MAE→ x-axis 0.25
y-axis 0.11
angle 0.52
Mission → **Complete**

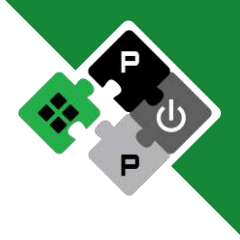
[1] Cereda, Elia, Luca Crupi, Matteo Risso, Alessio Burrello, Luca Benini, Alessandro Giusti, Daniele Jahier Pagliari, and Daniele Palossi. "Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs,", ICRA, 2023
[2] Palossi, Daniele, Zimmerman, Nicky., Burrello, Alessio, Conti, Francesco, Müller, Hanna, Gambardella, Luca Maria, ... & Guzzi, Jerome Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs. *IEEE Internet of Things Journal*, 2021

DNNs Deployment Flow



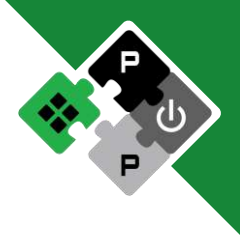
DNNs Deployment Flow





2. (Differentiable) Neural Architecture Search

Neural Architecture Search



- **Motivation: Picking hyper-parameters manually is tricky**
  - Biases (rules of thumb, traditions, etc.)
  - Fragmented and coarse design space explorations (e.g., width/res mult in MobileNets)
  - **Classic ML: hand-craft features, DL: hand-craft feature extractors**!

- **Neural Architecture Search (NAS)**
  - Automatic optimization of the network topology, exploring a large and fine-grain design space of hyper-parameter settings
  - Typically multi-objective: co-optimize accuracy and model complexity
    - Model size/#MACs….
    - …or better, latency/energy directly (requires models)!

Classic NAS

- Key steps:

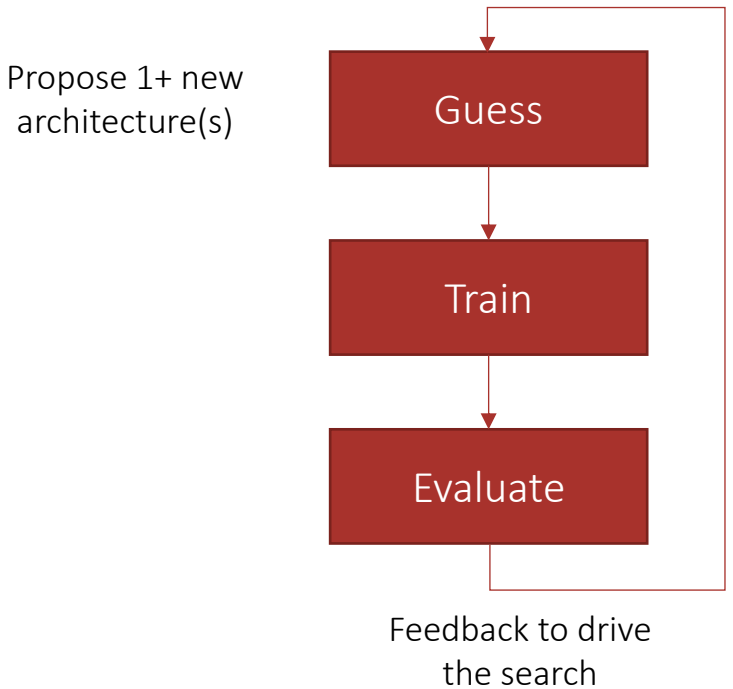
1. Define the search space

2. Define a search engine

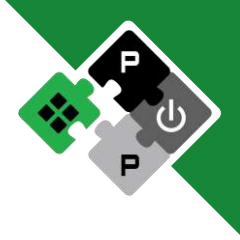
3. Build a performance estimator

- Thousands of GPU-hours per search!

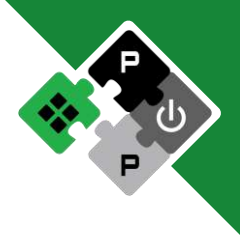
- Procedure:



Differentiable NAS (DNAS)



- Relax the search space to make it continuous and differentiable
- Optimize the topology by gradient descent during training
- Reduce search costs: Gradient-based optimization is much more lightweight than black-box methods (RL or Evolutionary)



3. PLiNIO: Plug-and-play Lightweight Neural Inference Optimizer

PLiNIO Motivation

SUPERNET: coarse-grain layer type selection

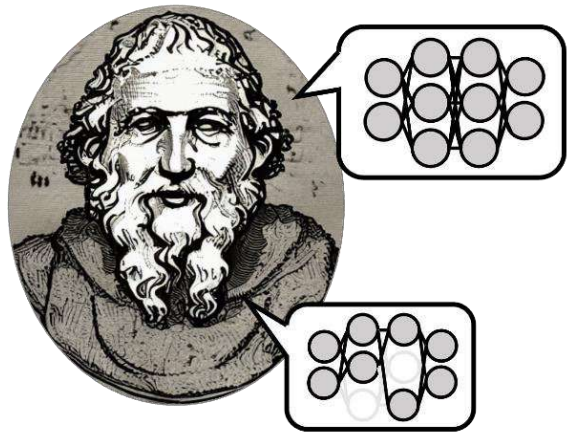
PIT: fine-grain layer's hyper-parameters selection

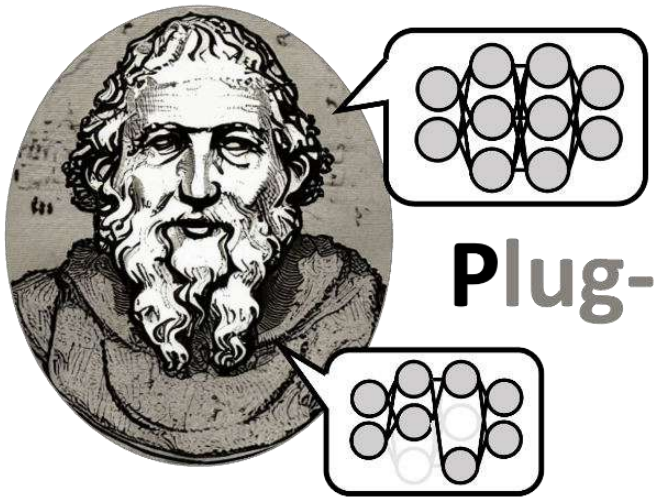
MIXPREC: precision assignement

ORTHOGONAL

DNAS BASED

Developed by us



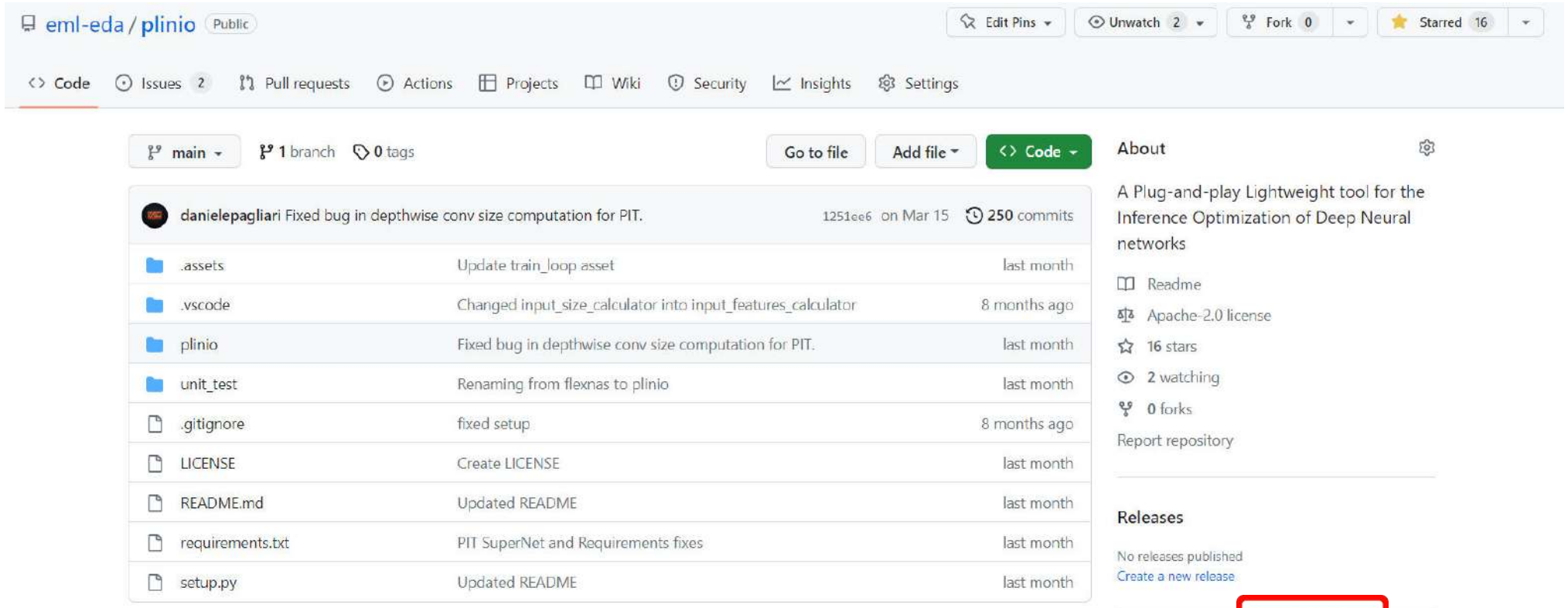


PLiNIO

Plug-and-play Lightweight Neural Inference Optimization

- **PLiNIO** is a **Python** package built on-top of the **PyTorch ecosystem** that provides a **P**lug-and-play **Li**ghtweight tool for the **I**nference **O**ptimization of DNNs.
- **PLiNIO** exploits as main optimization engine **DNAS** algorithms which notoriously balance flexibility and lightness.

PLiNIO



PLiNIO is open-sourced on github

SCAN ME

PLiNIO (cont'd)

- PLiNIO allows to automatically optimize your DNN's architecture with ***no more than three additional lines of code*** to your original training loop.

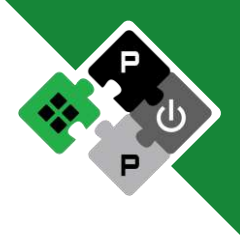
```
# A super standard pytorch training loop
model = ResNet()
for epoch in range(N_EPOCHS):
  for samle, target in data:
    output = model(sample)
    loss = criterion(output, target)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
# A plinio-enhanced pytorch training loop
model = ResNet()
model = plinio.PIT(model, input_shape=(C, H, W))  # 1 Convert the model
for epoch in range(N_EPOCHS):
  for samle, target in data:
    output = model(sample)
    task_loss = criterion(output, target)
    loss = task_loss + model.get_regularization_loss()  # Compute additional loss
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
exported_model = model.arch_export()  # 3 Export the optimized model
```

1

2

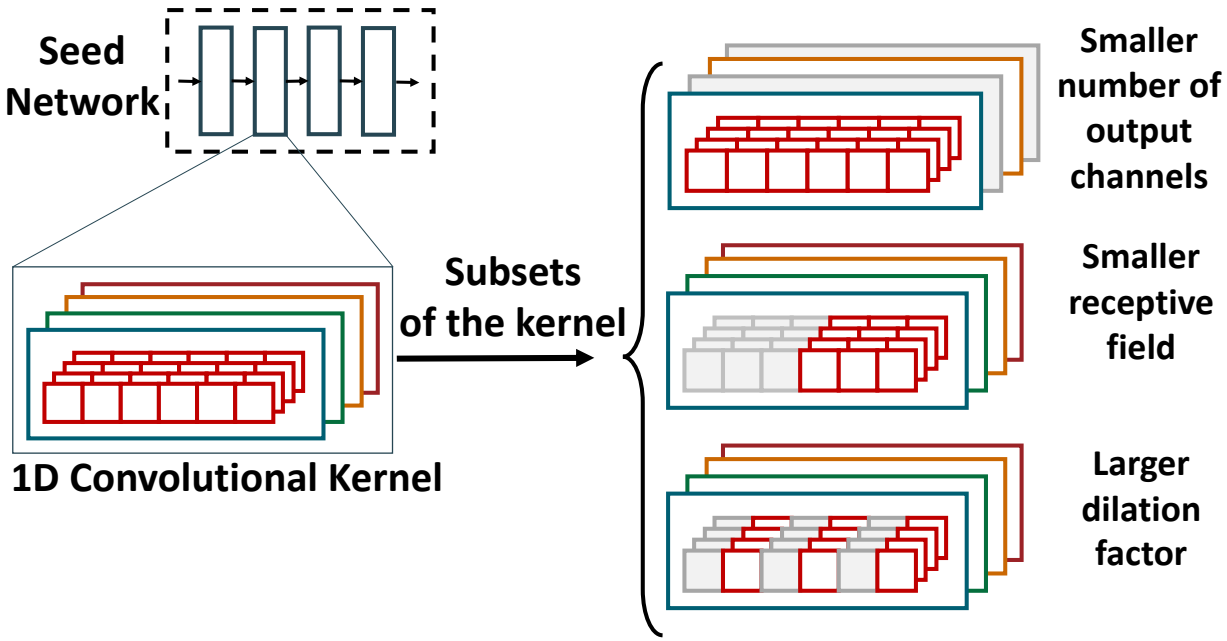
3



3. Developed Differentiable NAS algorithms

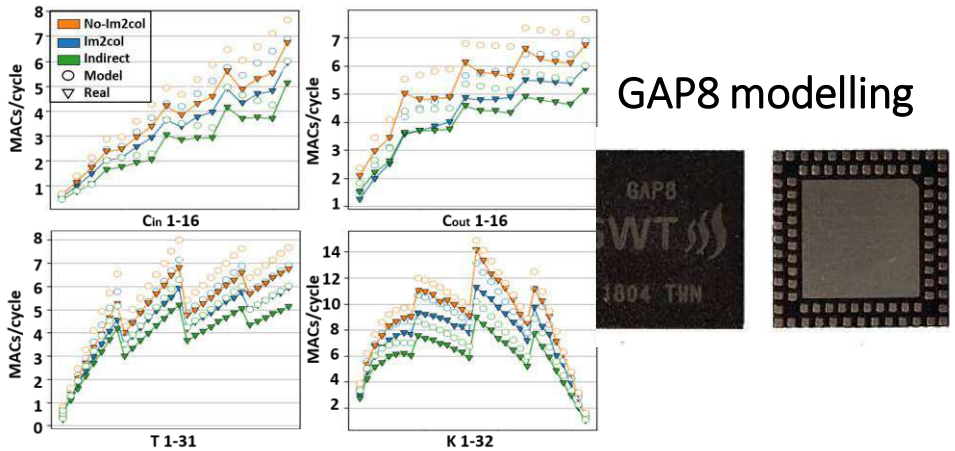
PIT: Pruning in Time

- Search space: For each Convolutional or Fully-Connected layer



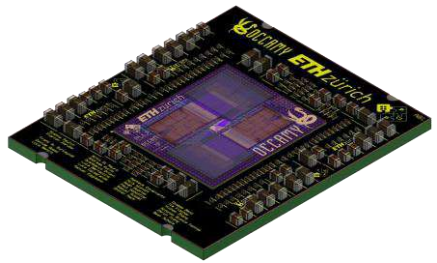
PIT: Pruning in Time

- Add a L1 regularization term to the training loss function that brings masks to 0
  - More 0-valued masks → smaller network
  - **Classical regularizers**:
    - **N. of weights**, correlates with memory occupation
    - **N. of MACs**, correlates with latency/energy
  - **HW regularizers**: piece-wise polynomial functions



GAP8 modelling

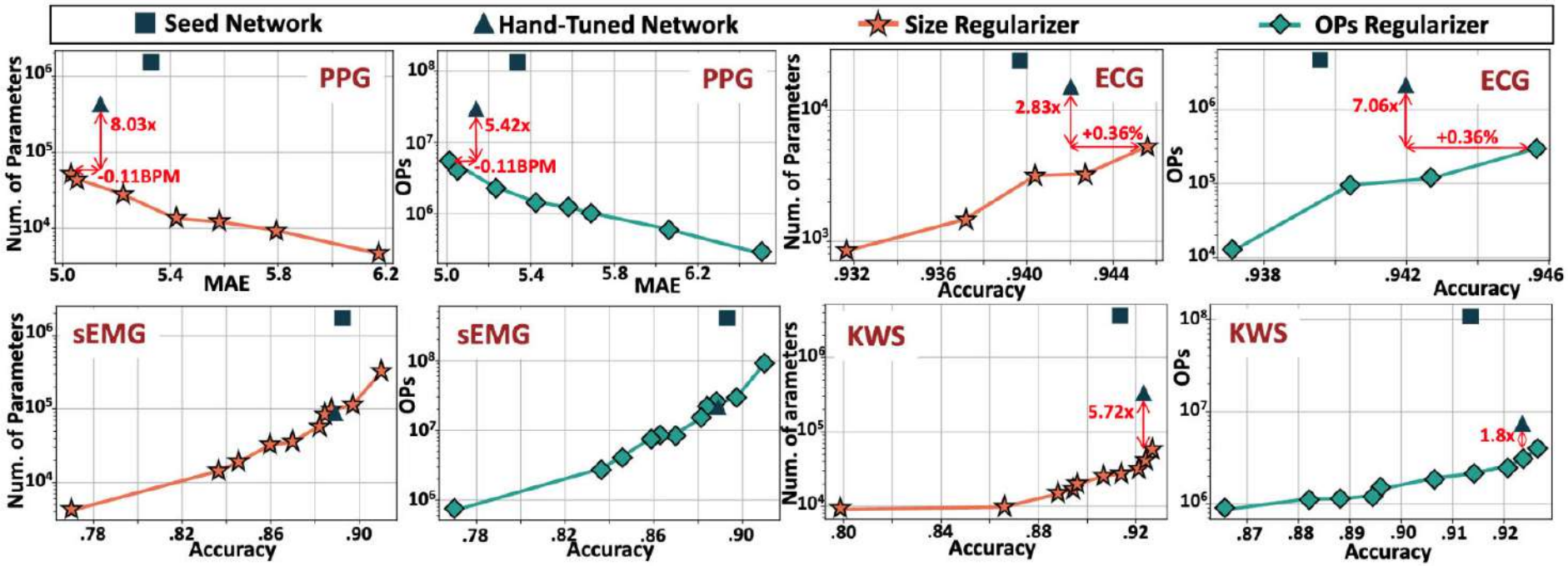
Future: **GAP9, Occamy** and many others…



- Final Loss Function: $\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta)$ → Regularizer, function of Trainable binary masks

PIT: Results

- 4 edge-relevant benchmarks (biosignals, keyword spotting).
- Up to 8x smaller and 7x faster models at iso-performance

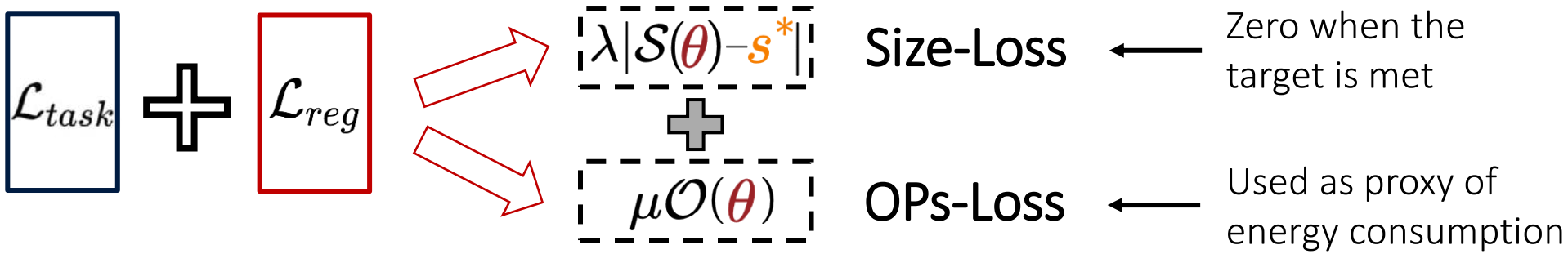


PIT into the wild

- PIT has been now extended to 2D networks for vision.
  - Example: drone-to-human pose estimation in low-power nanodrones
  - **Same results** of previous hand-tuned network with 3x less memory, thanks to PIT
  - Collaboration with POLITO + UNIBO + ETHZ + IDSIA (Lugano) → presented @ ICRA23

Multi-Regularization Loss

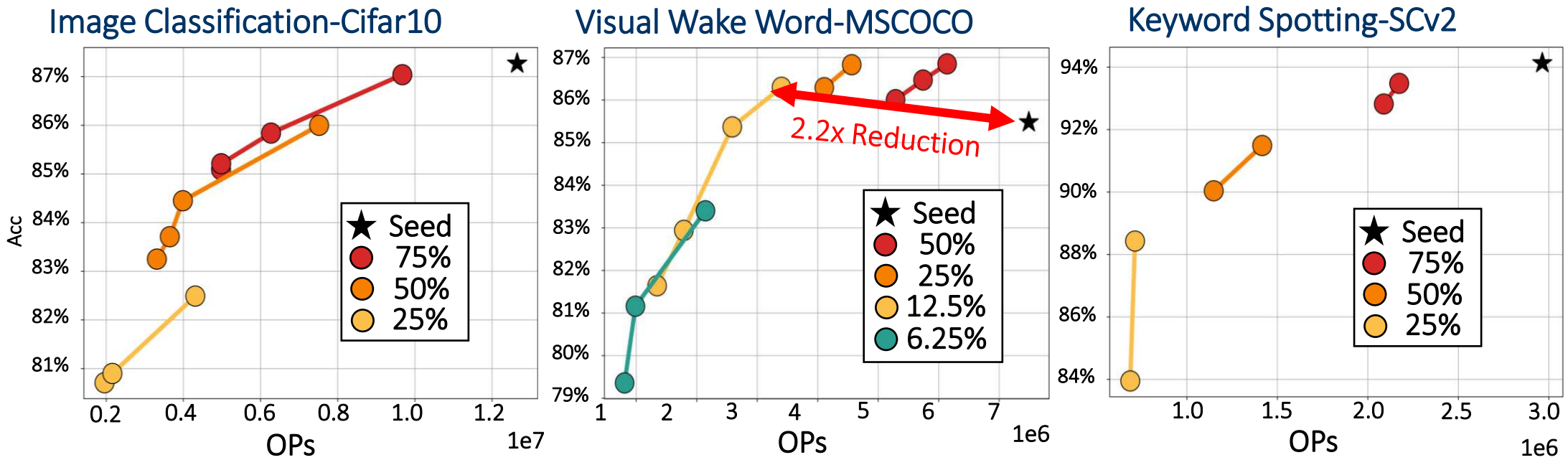
- From the designer perspective the main goal is finding an optimal trade-off between accuracy and complexity while satisfying the memory requirements s^* of the target.
- We develop a novel **multi-regularization loss** formulation:



  - The mutual importance of regularization loss terms is controlled with λ and μ:
    - λ is fixed and such as to satisfy $\lambda >> \mu$
    - μ is tweaked to explore different Accuracy vs Energy tradeoffs

Multi-Regularization: Results

- Experiments on three edge relevant use-cases from **MLPerf Tiny Benchmark Suite** which proposes reference optimized network implementations.



- We obtain rich Pareto sets of architectures in the **OPs vs. accuracy space**, with memory footprints spanning from 75% to 6.25% of baseline networks.

Other works

- Multi-constraint loss: a new NAS formulation to respect both a memory constraint and a maximum latency

- Multi-precision search: a different precision can be chosen for each channel of a tensor using gradient-descent.
  - We support the export of quantized networks which can be imported from DORY (Deeploy soon) and executed on PULP successfully!!!

- Heterogeneous-NAS: NAS for heterogenous hardware. It maps different part of a layer to different accelerators
  - Optimize network during NAS based on the type of layer/precision supported by each accelerator in a heterogeneous SoC
  - Tested on DIANA AIMC and Digital Accelerators
  - Accepted at ISLPED2023

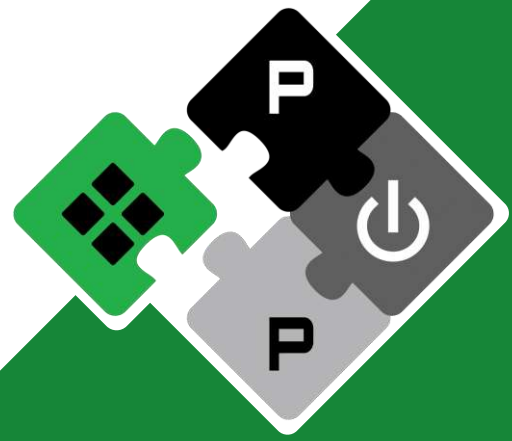
What's next?

- Adding new hardware models to improve the NAS search (GAP9, Occamy...)
- Insert the hardware in the loop to have a precise feedback of the network on the MCU
- Targeting full application, trying to optimize a task and not only a loss
- Extend PliNIO to include all methods and allow for automatic end-to-end optimization pipelines
- Interface the NAS tools with the deployment pipelines

ETH zürich ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Neural Network Deployment on Heterogeneous Systems

Moritz Scherer scheremo@iis.ee.ethz.ch



@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

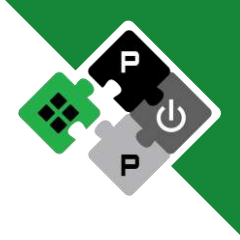
PULP Platform

Open Source Hardware, the way it should be!

Neural Network Deployment

- Until very recently, residual CNNs dominated the state-of-the-art
  - ResNets
  - MobileNets (v1, v2, v3, …)
  - EfficientNets
- Dory was specifically designed for integer-quantized residual CNNs
  - Support for two concurrent branches
  - Support for integer arithmetic on the PULP Cluster
  - Support for memory-aware layer-wise tiling
  - Efficient parallelization strategies for various operators
- **A match that led to advancements in the SoA several times over!**

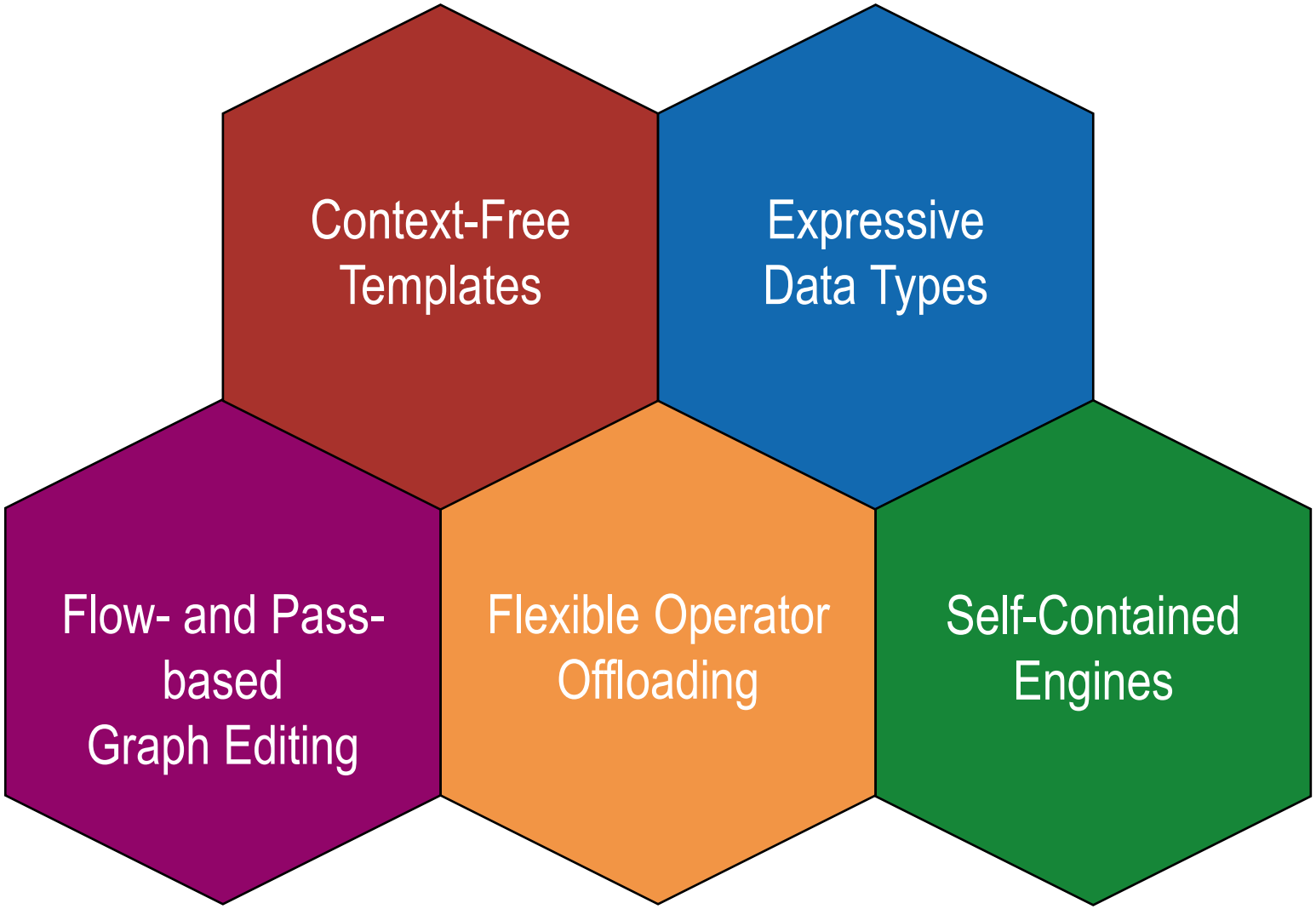
Dory for Deployment – Challenges & Limitations



- Dory deployment with accelerators is challenging
  - Some layers have very low arithmetic intensity
    - Depthwise convolutions, Matrix multiplications, …
  - Depth-first tiling helps to keep execution compute bound
    - Siracusa: Executing IRB layers depth-first improves MobileNetv2 performance by 60%!

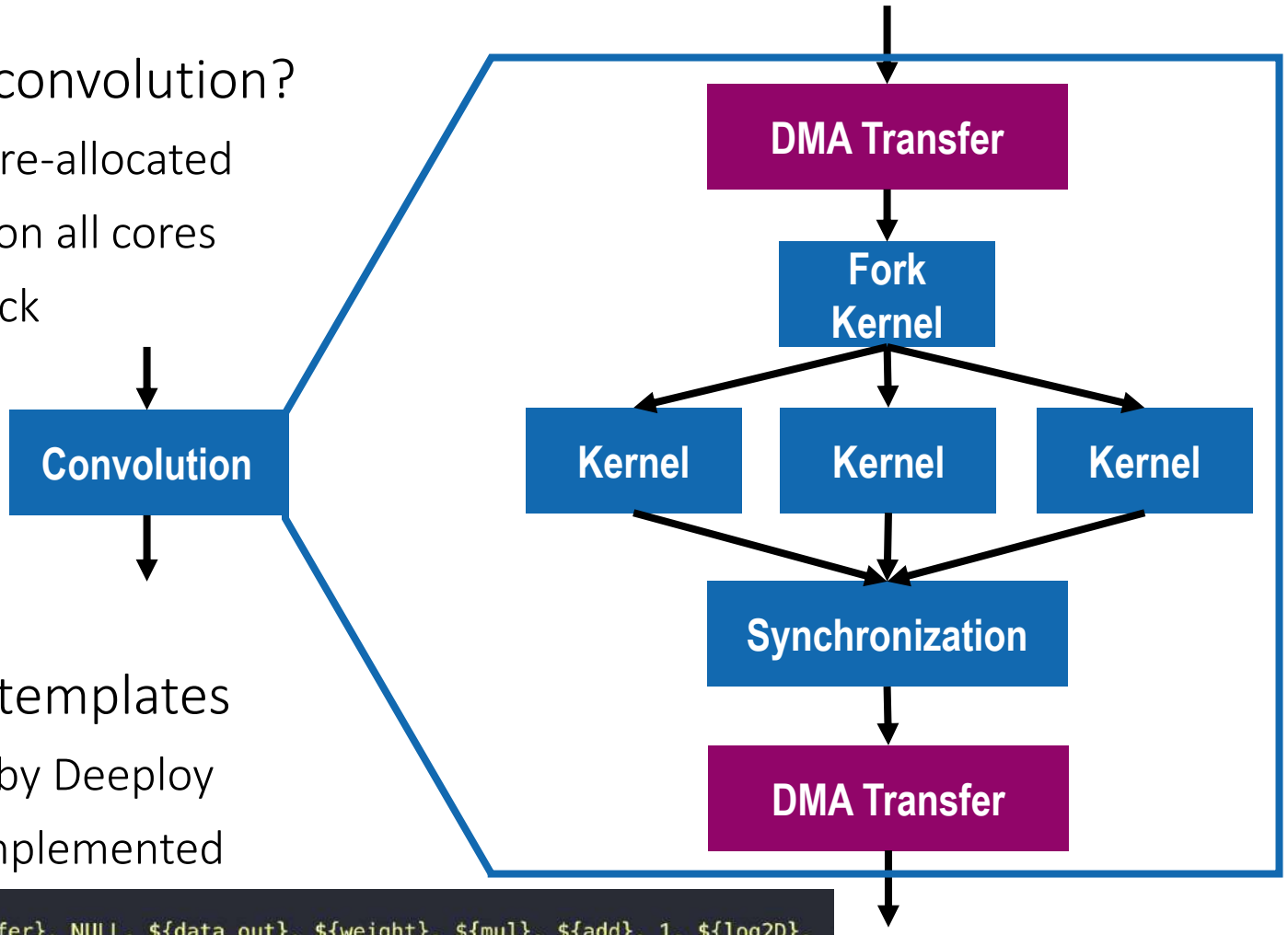
- Even more challenges for our deployment tools
  - Transformers dominate all ML benchmarks
  - Low-precision floating point training & inference on microcontrollers is gaining traction
  - Occamy & MemPool are breaking ground on HPC PULP systems

Deeploy – Enabling Heterogeneous Deployment



Deeploy – Context-Free Templates

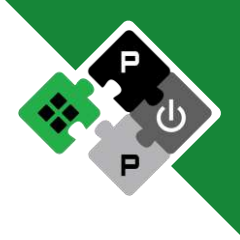
- What does it take to run a convolution?
  - Inputs & weights need to be pre-allocated
  - Kernel templates need to run on all cores
  - Outputs need to be moved back



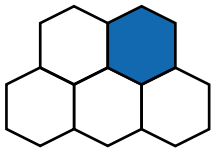
- Deeploy uses context-free templates
  - DMA calls, etc. are generated by Deeploy
  - Only kernel calls need to be implemented

```
PULPConv2D_8_Template = PULP2DConvTemplate("""
pulp_nn_conv${signatureString}(${data_in}, ${ctxtBuffer}, NULL, ${data_out}, ${weight}, ${mul}, ${add}, 1, ${log2D},
${dim_im_in_x}, ${dim_im_in_y}, ${ch_im_in}, ${dim_im_out_x}, ${dim_im_out_y}, ${ch_im_out}, ${dim_kernel_x}, ${dim_k
ernel_y}, ${pads[0]}, ${pads[2]}, ${pads[1]}, ${pads[3]}, ${strides[0]}, ${strides[1]}, 1, 1);
""")
```

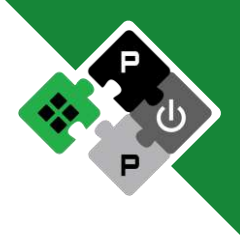
Deeploy – Expressive Data Types



- Deeploy uses expressive primitive types
  - Immediate, Pointer, Struct & Future
- Bring your own immediate types
  - Only need to implement a function that checks a value
  - Compose your own types in pointers, structs, and futures
- Automatic strong type checking
  - For your own immediate types, and all composed types



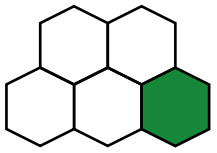
Deeploy – Self-Contained Engines



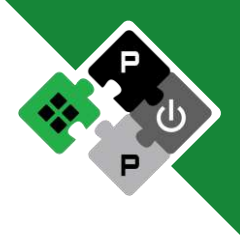
- The PULP SoC is designed for adding accelerators
  - General-Purpose Accelerators like the PULP Cluster
  - Application-specific Accelerators like N-EUREKA , NE16, CUTIE, ITA, …

- Compute engines are highly customizable
  - Data types, Programming model, Memory access

- Deeploy keeps each engine self-contained
  - Engine-specific, context-free templates, programming model, and data types
  - The same engine in a different SoC works the same



Deeploy – Simple Microcontrollers



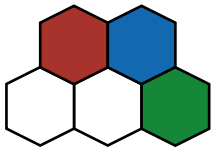
- This lets us generate network inference code!

Type-checked, nested, auto-generated CMSIS-NN structs

```
DeeployNetwork_Deeploy_BUFFER_output_0_ctxt = (cmsis_nn_context){.buf = NULL, .size = 0};
DeeployNetwork_Deeploy_BUFFER_output_0_activation = (cmsis_nn_activation){.min = -64, .max = 63};
DeeployNetwork_Deeploy_BUFFER_output_0_fc_params = (cmsis_nn_fc_params){
    .input_offset = 0, .output_offset = -64, .filter_offset = 0, .activation = {.min = -64, .max = 63}};
DeeployNetwork_Deeploy_BUFFER_output_0_quant_params =
    (cmsis_nn_per_tensor_quant_params){.multiplier = 9609216, .shift = 0};
DeeployNetwork_Deeploy_BUFFER_output_0_input_dims = (cmsis_nn_dims){.n = 1, .h = 1, .w = 1, .c = 512};
DeeployNetwork_Deeploy_BUFFER_output_0_filter_dims = (cmsis_nn_dims){.n = 512, .h = 1, .w = 1, .c = 10};
DeeployNetwork_Deeploy_BUFFER_output_0_output_dims = (cmsis_nn_dims){.n = 1, .h = 1, .w = 1, .c = 10};
DeeployNetwork_Deeploy_BUFFER_output_0_bias_dims = (cmsis_nn_dims){.n = 1, .h = 1, .w = 1, .c = 10};
```

```
arm_fully_connected_s8(
    &DeeployNetwork_Deeploy_BUFFER_output_0_ctxt, &DeeployNetwork_Deeploy_BUFFER_output_0_fc_params,
    &DeeployNetwork_Deeploy_BUFFER_output_0_quant_params, &DeeployNetwork_Deeploy_BUFFER_output_0_input_dims,
    DeeployNetwork_Deeploy_BUFFER_28, &DeeployNetwork_Deeploy_BUFFER_output_0_filter_dims,
    DeeployNetwork_Deeploy_BUFFER_32, &DeeployNetwork_Deeploy_BUFFER_output_0_bias_dims,
    DeeployNetwork_Deeploy_BUFFER_classifier__QL_REPLACED__INTEGERIZE_UNSIGNED_ACT_PASS_0_add,
    &DeeployNetwork_Deeploy_BUFFER_output_0_output_dims, DeeployNetwork_Deeploy_BUFFER_output_0);
```

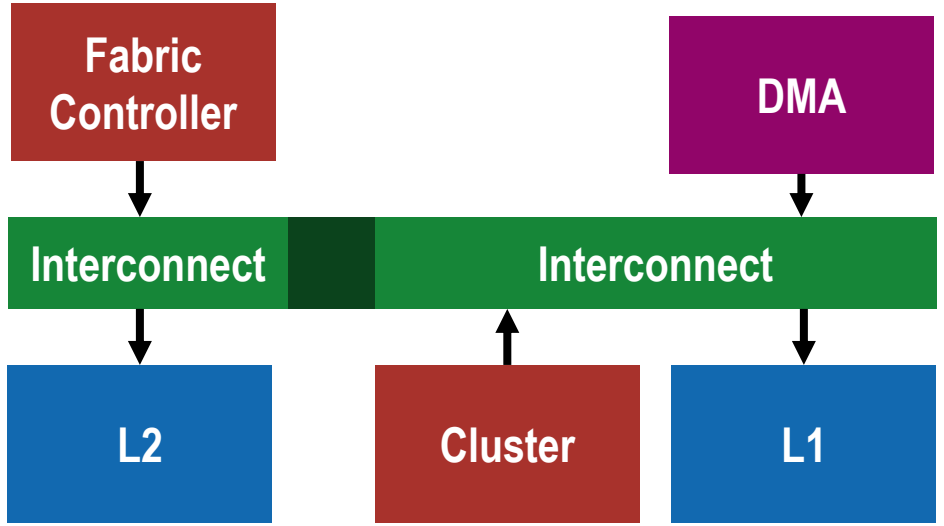
ARM CMSIS-NN Convolution kernel call



Deeploy – Flexible Operator Offloading

- But only for single-core, single-memory-level systems
  - Everything happens in the same execution context

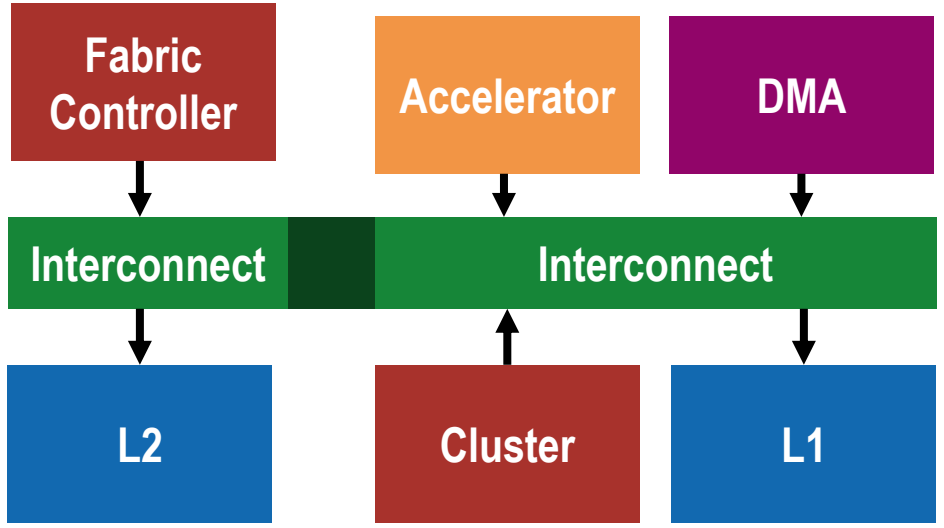
- To run on a PULP Cluster, we have to
  - Move memory with the DMA
  - Offload code to the cluster
- From the Fabric Controller's POV
  - The DMA and Cluster work asynchronously



We need a way to model offloading and concurrent execution

Deeploy – Closures and Futures

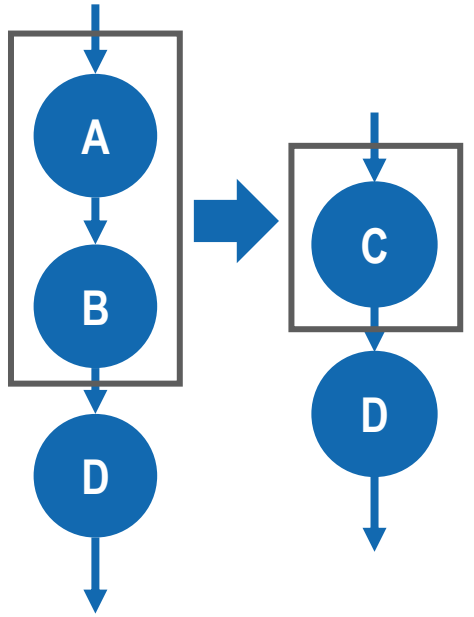
- Deeploy uses closures to offload kernels
  - A closure is a function that wraps a *kernel call* and its *state*
- Asynchronous computation produces *Future*-typed outputs
  - Futures are values that “will be available later”
  - Before generating a Future, we need to *dispatch* it
  - Before accessing a Future, we need to *resolve* it
  - Future types provide code to dispatch and resolve
- Futures enable local synchronization
  - No OS, tasks or threads required – but supported



Futures allow us to address engines concurrently

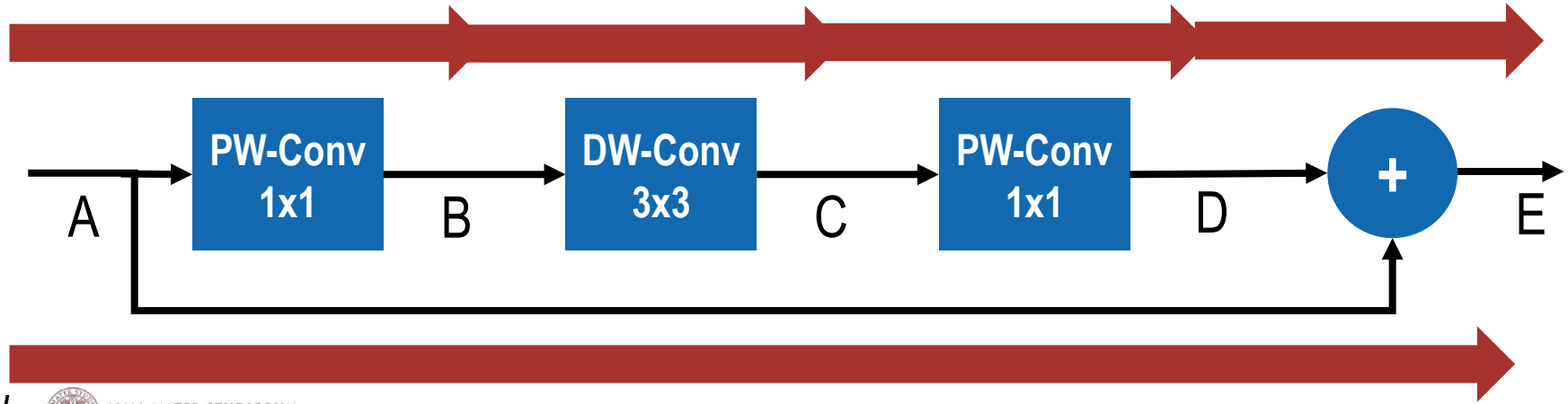
Deeploy – Tiling & Graph Manipulation

- Deeploy comes with a flexible & powerful graph editing framework
- Passes are used for match-based transformations
  - “Replace all occurrences of A->B with C”
- Flows are used for graph-level information propagation
  - Tensor type inference
  - Bias pushing
  - Tensor liveness analysis
- Deeploy’s tiling algorithm combines passes and flows
  - And allows for depth-first tiling, as well!



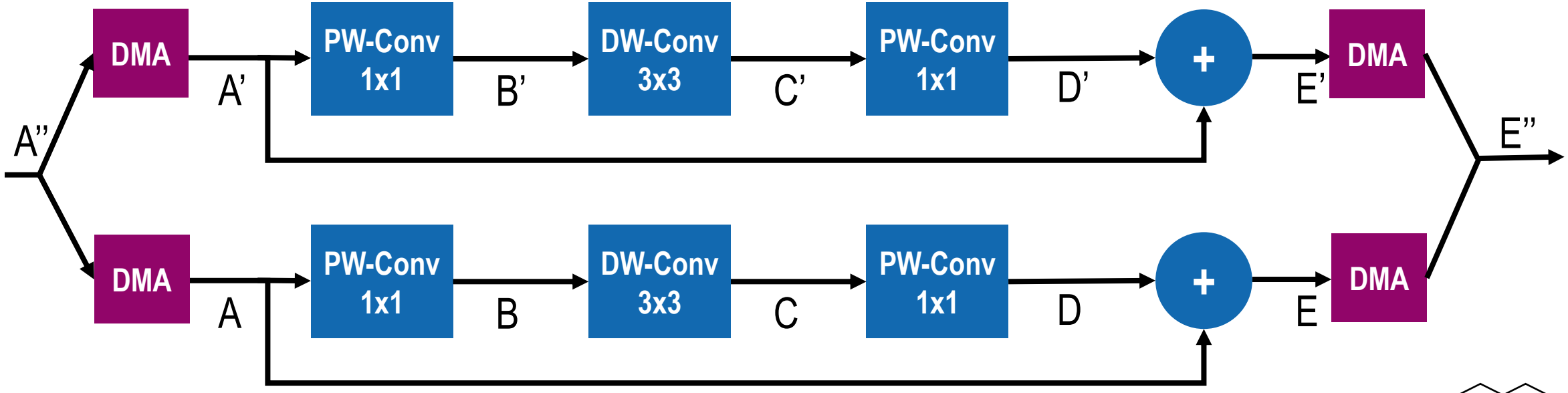
Deeploy – Tile Constraint Flow

- We find our pattern with a pass
- Tiling constraints are computed with a flow
  - Constraints(B) = PW-Constraints(Constraints(A))
  - Constraints(C) = DW-Constraints(Constraints(B))
  - Constraints(D) = PW-Constraints(Constraints(C))
  - Constraints(E) = Addition-Constraints(Constraints(A), Constraints(D))
- Using ORTools, we can compute a correct tiling strategy

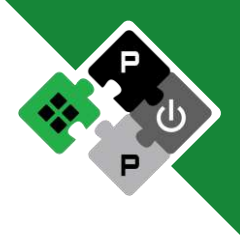


Deeploy – Graph Tiling

- With our tiling solution, implement a replacement pass
  - Duplicate subgraph
  - Add memory transfer nodes
  - And the rest of the framework manages code generation!



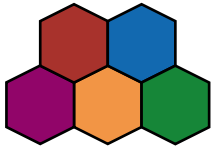
Deeploy – Ongoing and Future Work

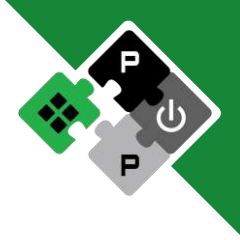


- Engine support is growing, and an open-source release is on the horizon
  - Implemented: ARM Cortex-M, MemPool, ITA, PULP Cluster
  - WIP: N-EUREKA, Floating point support, …
  - Future Work: Multi-Cluster systems like Occamy, Carfield, …

- Deeploy is designed with extensions in mind
  - Flows, Futures, Closures, and Tiling were designed as extensions
  - New engines and systems are crucial and easy to get started on

- **We are looking for contributors!**
  - Talk to me, Victor, Francesco, or Alessio – there's plenty to do!





Thanks for the attention