EDGE AI Academy

Summer School

7-8 July 2025, Pisa, Italy

End-to-end neural networks on RISC-V: from model optimization to full-system simulation

Alessio Burrello, Daniele Jahier Pagliari, Giuseppe Tagliavini

Politecnico di Torino, Università di Bologna

7-8 July 2025 Pisa, Italy

The EdgeAI Barrier

- The RTL of your shiny new hardware is finally available…
- … but can you run an actual DNN on it?

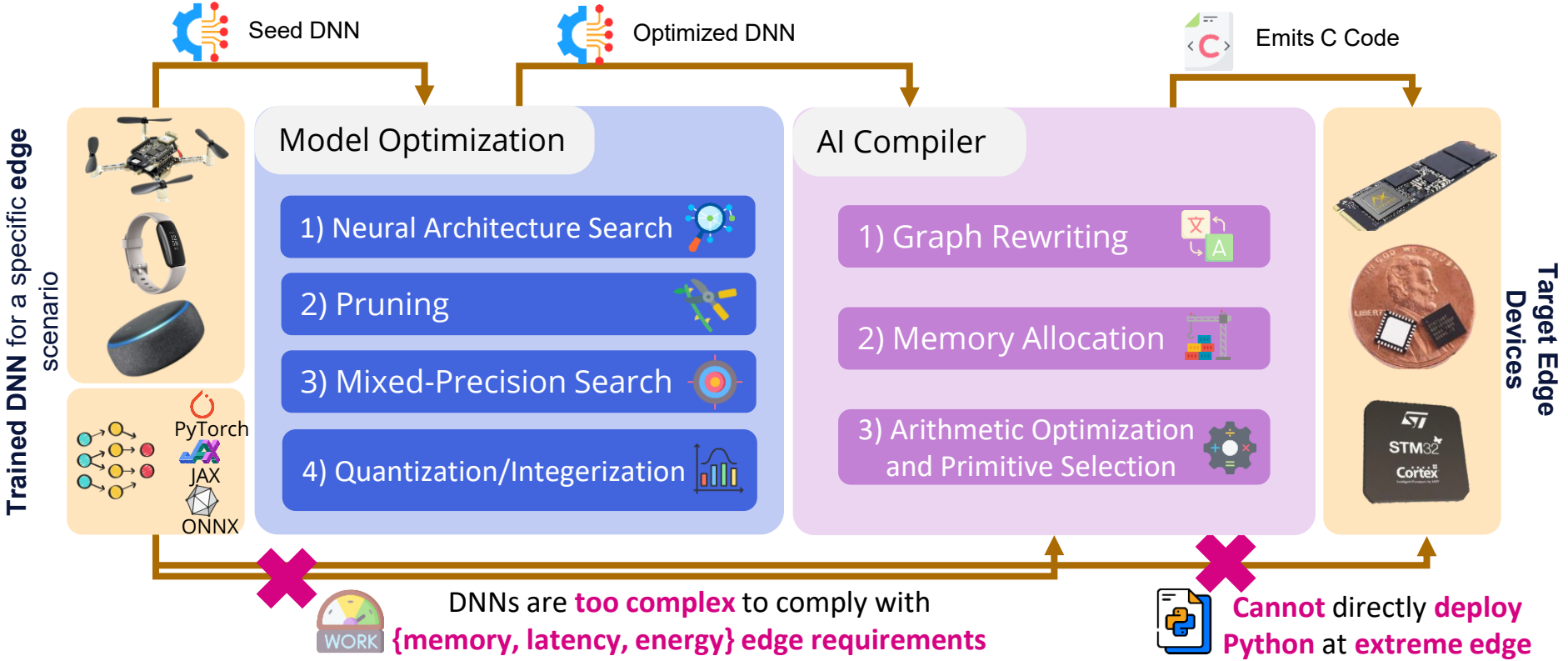
1. You have KBs of memory… and latency/energy constraints
2. Limited support for DNN operators… (no OS on target usually!!)
3. The actual HW might not be there yet, and RTL simulation of a full DNN workload is too slow.

Design Automation is the key to tame **Edge AI on heterogeneous devices**.

Copyright © 2025

Introduction

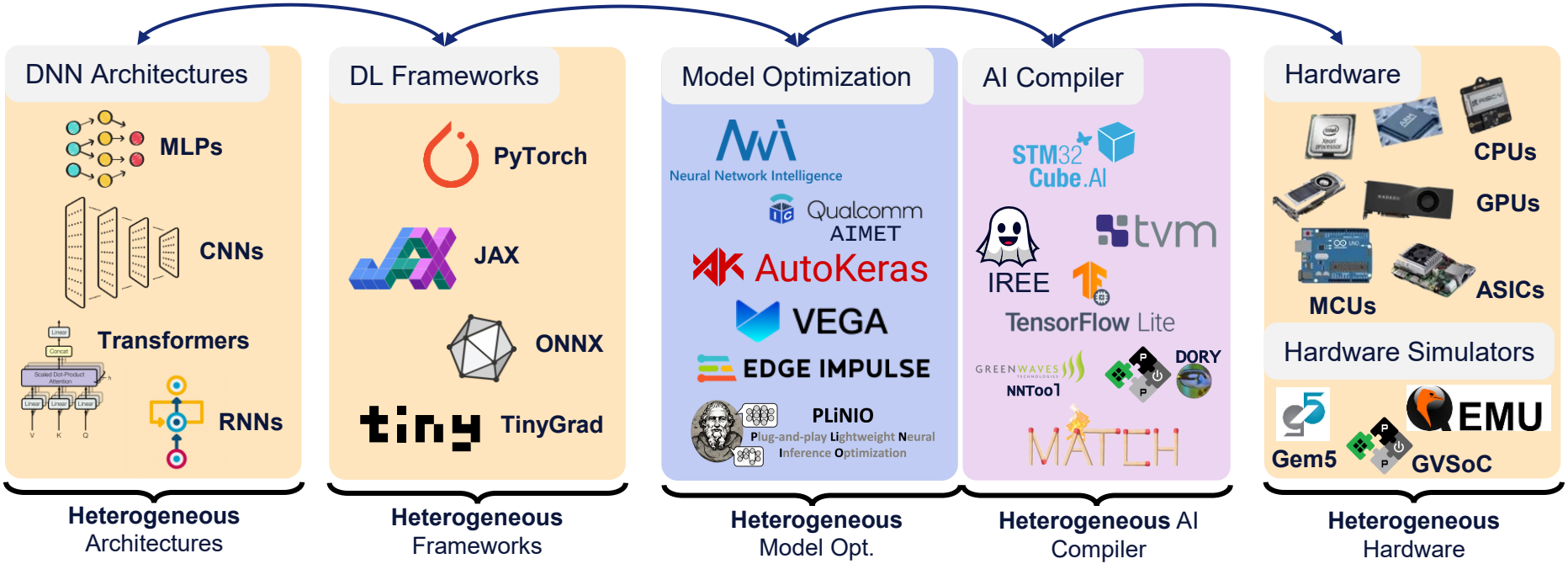
- It's a long way to the ***metal*** (if you wanna ***edge inference***)…



Copyright © 2025

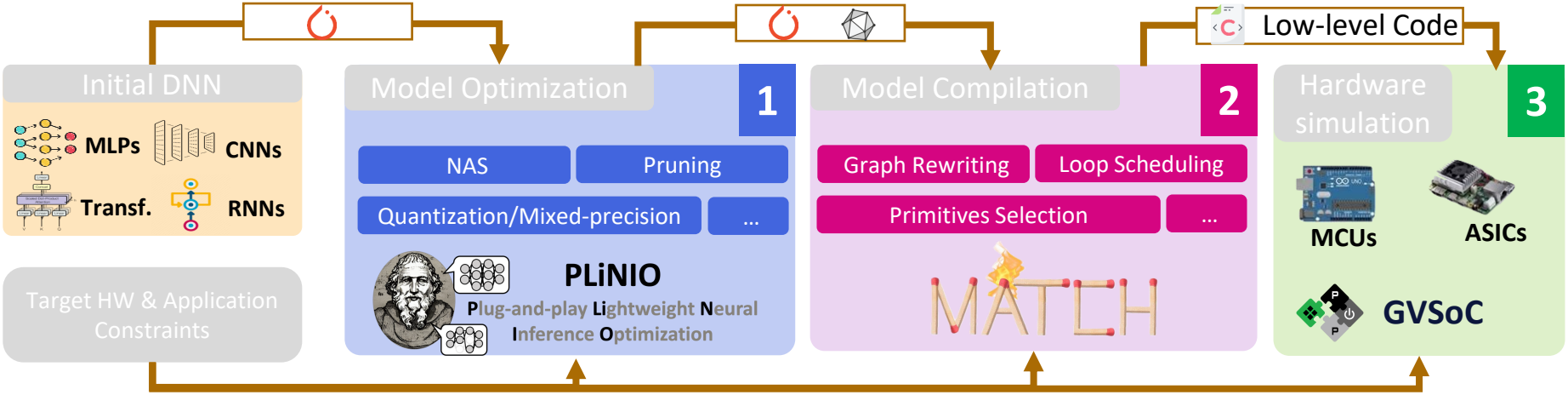
Introduction

- A **heterogeneous** state-of-the-art:



Copyright © 2025

Outline of the presentation



- **Plug-and-play**: Re-use SotA training loops
- **Lightweight** gradient-based optimizations
- **Hardware-aware** through easily encoded cost models

Fully integrated

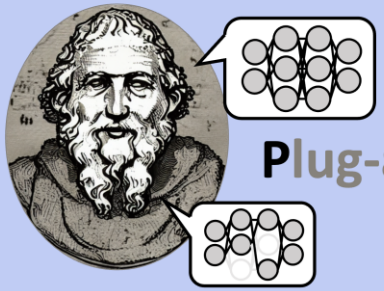
- Close to **optimal hardware utilization**
- Broad **support** for **DNN operators**
- **Easy extensibility** to future HW targets

Fully integrated

- Timing-accurate instruction simulator
- **Less than 10%** maximum **latency error**
- **Easy to extend to new platforms**

Copyright © 2025

Model Optimization



PLiNIO

Plug-and-play **Li**ghtweight **N**eural **I**nference **O**ptimization

open source

Copyright © 2025

Model Optimization - Plinio

- **PLiNIO: Python** package built on-top of **PyTorch** that provides a **P**lug-and-play **Li**ghtweight tool for the **I**nference **O**ptimization of DNNs.
- It allows you to automatically optimize a DNN architecture adding ***3 lines of code*** to your standard training loop.

Mathematically:

$$\min_{W,\theta} \mathcal{L}(W,\theta) + \lambda \mathcal{R}(\theta)$$

$\mathcal{L}$: criterion
λ : regularization strength
$\mathcal{R}$: model.cost
W : DNN weights
θ : architectural parameters

```
1  # A PLiNIO enhanced training loop
2  model = MyNN()
3  model = plinio.Method(model, cost=cost_fn, …) # 1. Convert the model
4  for epoch in range(N_EPOCHS):
5      for sample, target in data:
6          output = model(sample)
7          loss = criterion(output, target) + model.cost # 2. Add cost-dependent loss term
8          optimizer.zero_grad()
9          loss.backward()
10         optimizer.step()
11 model = model.export() # 3. Export the optimized model
```

1 2 3

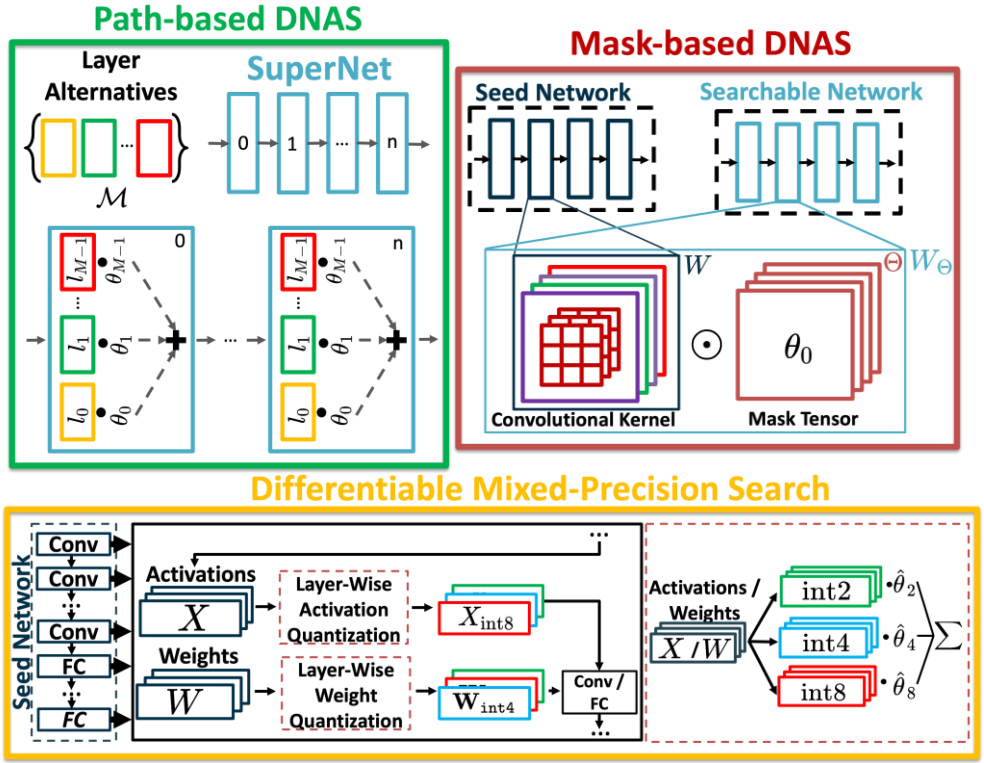
- PLiNIO employs **gradient-based** optimization algorithms that keep **search cost low**.

[1] D. Jahier Pagliari et al., "PLiNIO: A User-Friendly Library of Gradient-based Methods for Complexity-aware DNN Optimization," IEEE FDL, 2023

Copyright © 2025

Plinio – NAS algorithms

PLiNIO supports the following optimizations:



- **Path-based DNAS**: select among different layer alternatives.
- **Mask-based DNAS**: fine-grained hyper-parameters optimization (e.g., channel pruning, N:M pruning)
- **Differentiable Mixed-Precision Search**: fine-grained mixed-precision quantization of activation and weights.

All framed as gradient-based opt:

$$\min_{W,\theta} \mathcal{L}(W,\theta) + \lambda \mathcal{R}(\theta)$$

Copyright © 2025

Plinio – Cost Models

One of the key advantage of **PLiNIO** is the **flexible DNN Cost specification**:

$$\min_{W,\theta} \mathcal{L}(W,\theta) + \lambda\mathcal{R}(\theta)$$

- \+ Possibility to combine **more than one cost model** simultaneously
- \+ E.g., maximize accuracy **under memory and latency constraints**:

Natively Supported Targets:

Hardware unaware

Number of Params

Number of Bits

Number of OPs

Number of BitOPs

Hardware aware

GAP8/9 Latency (multi-core SIMD MCU)

DIANA Digital (Systolic Array)

DIANA Analog (In-Mem Computing)

NE16 (Convolution Accelerator)

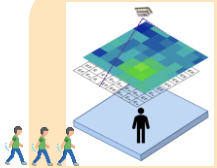
- \+ Possibility to define **any** cost model that can be expressed as a **differentiable function** of:
  - \+ Layer's hyperparameters
  - \+ Output tensor shape
  - \+ Parameters and activations bitwidths

Copyright © 2025

Plinio – Application Results

We used **PLiNIO** to **optimize DNNs** for various applications:

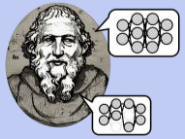
People Counting on Low-resolution Infrared Arrays



Sensor Positioning

One Person

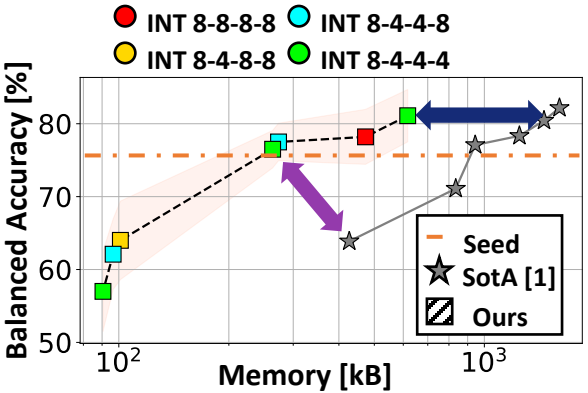
Three People



PLiNIO

Mask-based DNAS
Mixed-Precision Search

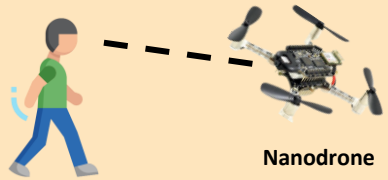
SotA CNN [1]



- With BAS > 80% wrt SotA, **2.4x smaller** model.
- SotA outperformed by **more than 10% BAS**.

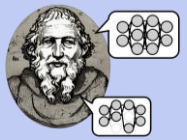
[1] C. Xie et al., "Efficient deep learning models for privacy-preserving people counting on low-resolution infrared arrays," IEEE IoT J., 2023

Human-pose Estimation on Nanodrones



Walking Person

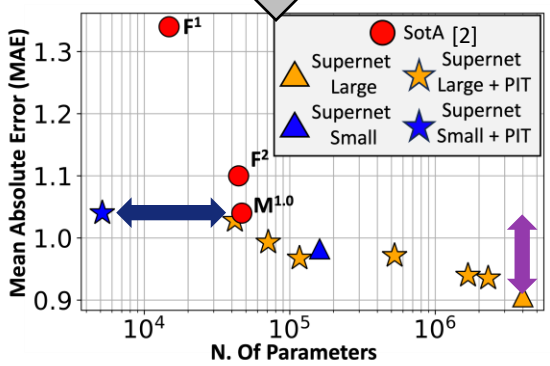
Nanodrone



PLiNIO

Path-based DNAS
Mask-based DNAS

SotA CNN [2]



- **9x size reduction** at iso-error wrt SotA.
- **Error reduced** by **13.8%** wrt SotA.

[2] E. Cereda et al., "Deep Neural Network Architecture Search for Accurate Visual Pose Estimation aboard Nano-UAVs," 2023 IEEE ICRA

Copyright © 2025

Combined Cost Models - DUCCIO

DUCCIO: DNAS Under Combined Constraints in One-Shot

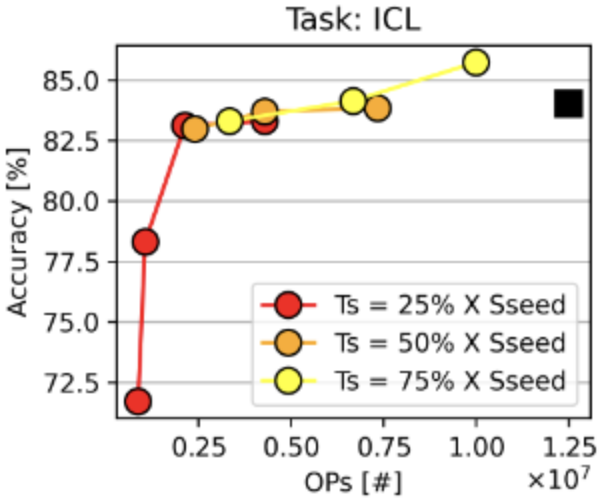
$$\min_{W,\theta} \mathcal{L}(W;\theta) + \sum_{j=0}^{J} \lambda_j \max(0, \mathcal{R}_j(\theta) - T_j)$$

- One cost dimension → J cost dimensions (e.g., **size** and **latency**)
- **Max(0, cost-target)**: Remove penalization within the target.
- Use **increasingly large λ_i** to ensure all constraints are respected.

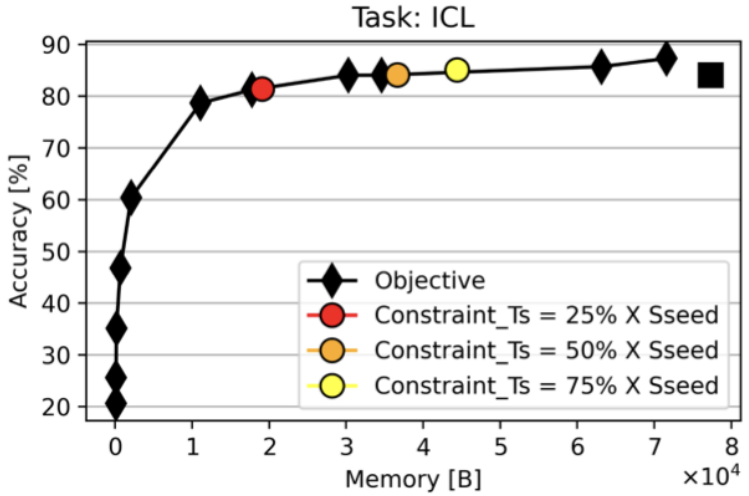
Copyright © 2025

Plinio – DUCCIO Results

- Applies to any DNAS, e.g. PIT (Mask-based) and SuperNet (Path-based)
- Example results on CIFAR-10 for GAP8 (RISC-V-based Multi-core ULP SoC).



Acuracy vs Ops under different memory constraints



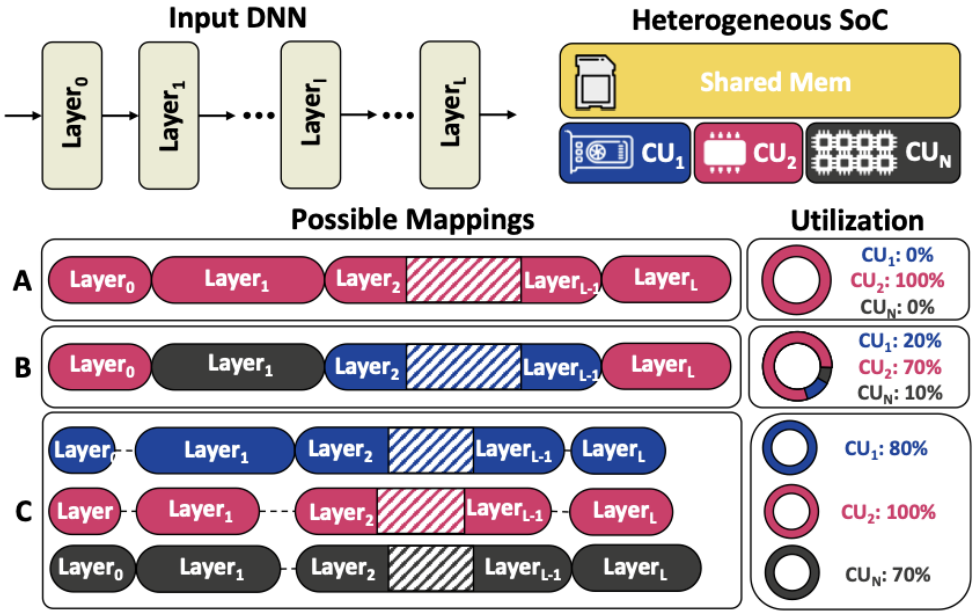
Constraint-based vs Objective-based R term.

[1] A. Burrello et al., "Enhancing Neural Architecture Search With Multiple Hardware Constraints for Deep Learning Model Deployment on Tiny IoT Devices," IEEE TETC, 2024

Copyright © 2025

DNN and HW Mapping Co-Optimization: ODIMO

GOAL: Can we co-optimize the **DNN topology** and its **mapping** onto a heterogeneous multi-accelerator SoC, at **training-time**?



Mapping space depends on accelerators' **operator support**, e.g.:

- **Int/float precision**
- **Type of layer** (e.g. Conv vs DW Conv)

Copyright © 2025

DNN and HW Mapping Co-Optimization: ODIMO

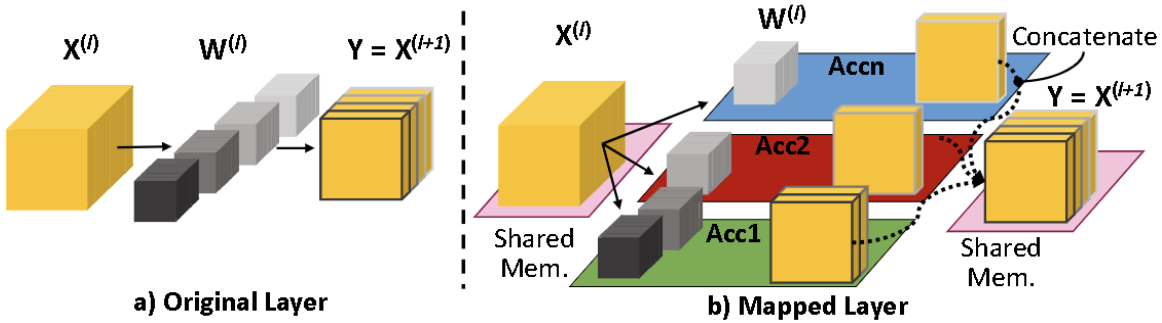
Problem framed similarly to a **Path-based NAS** or **Mixed-Precision Search** for **layer type-based** or **precision-based** mapping respectively:

- Assignment of layers to accelerators regulated by **trainable categorical parameters**.
- The optimal assignment is chosen at the **channel-level**.
- Optimization over a huge search space → **cannot be brute-forced**

Copyright © 2025

ODIMO – Key principle

- **Requirements:** compatible activation formats and **shared activation memory**.
- Each layer is transformed into n “sub-layers”, **running in parallel on the accelerators**.



[1] M. Risso et al., “Optimizing DNN Inference on Multi-Accelerator SoCs at Training-time,” IEEE TCAD, 2025

Copyright © 2025

ODIMO – Cost model

Co-optimization of **accuracy** and **latency/energy** obtained through a specific R loss.

Example for latency:

$$R(\alpha) = \sum_{l} M^{(l)}, \qquad M^{(l)} = \mathbf{max}(LAT_1^{(l)}(\alpha), \dots, LAT_n^{(l)}(\alpha))$$

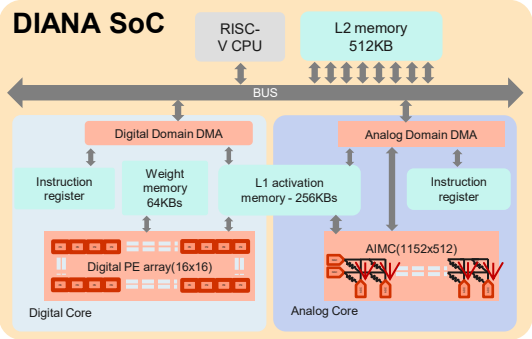
(l) = l-th layer, $LAT_i^{(l)}(\alpha)$ = Usual PLiNIO cost model for i-th accelerator

Accelerators running in parallel → max()

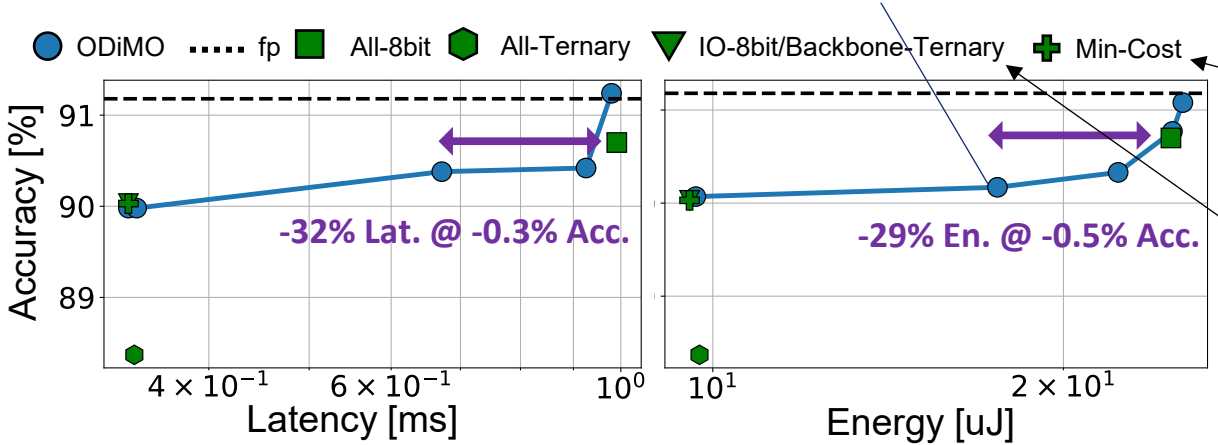
Goal = maximize utilization.

Copyright © 2025

PLINIO – ODIMO Results



- **Analog Ternary accelerator** and **Digital 8-bits accelerator**
- **Shared L1 activation Mem** of 256 kB



Accuracy-unaware channel-wise mapping that statically minimizes latency/energy ($R(\alpha)$)

Heuristic mapping from prev. work

Copyright © 2025

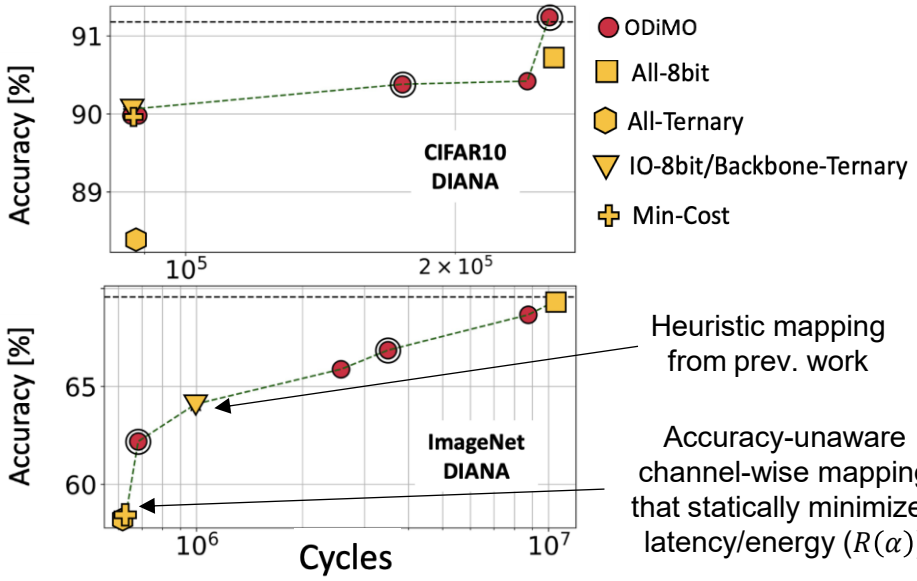
PLINIO – ODIMO Results on RISC-V-based SoCs

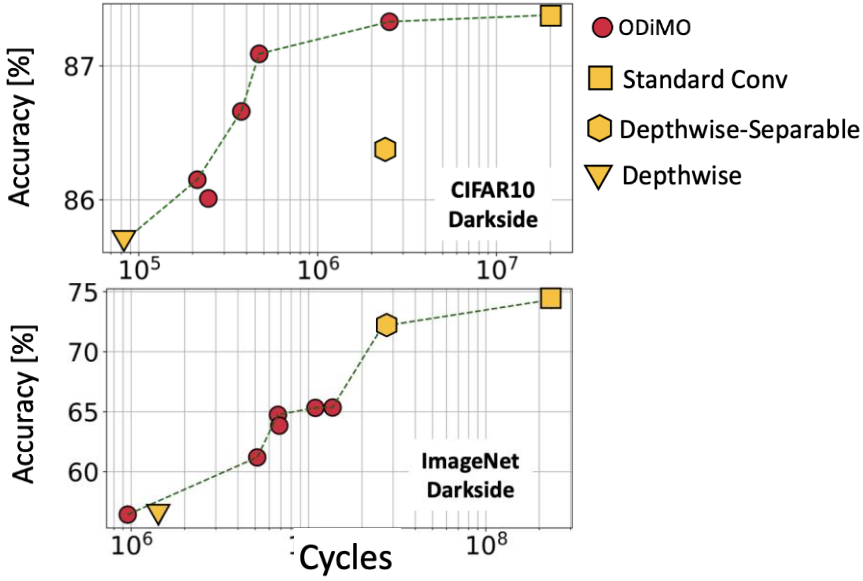
DIANA SoC:

- **Analog Ternary accelerator** and **Digital 8-bits accelerator**
- **Shared L1 activation Mem** of 256 kB

Darkside SoC:

- **Generic RV 8-core cluster** and **DepthWise Convolution Engine (DWE)**
- **Shared L1 activation Mem** of 128 kB





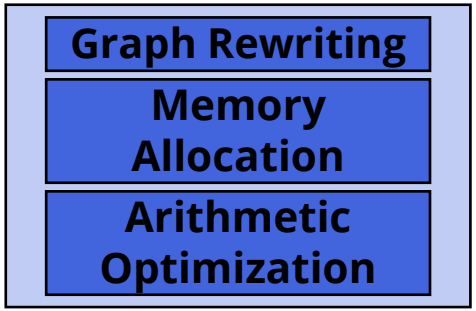
Copyright © 2025

Model Compilation: Python to Hardware Toolchain

open source

Copyright © 2025

AI Compiler: Python to Hardware Toolchain



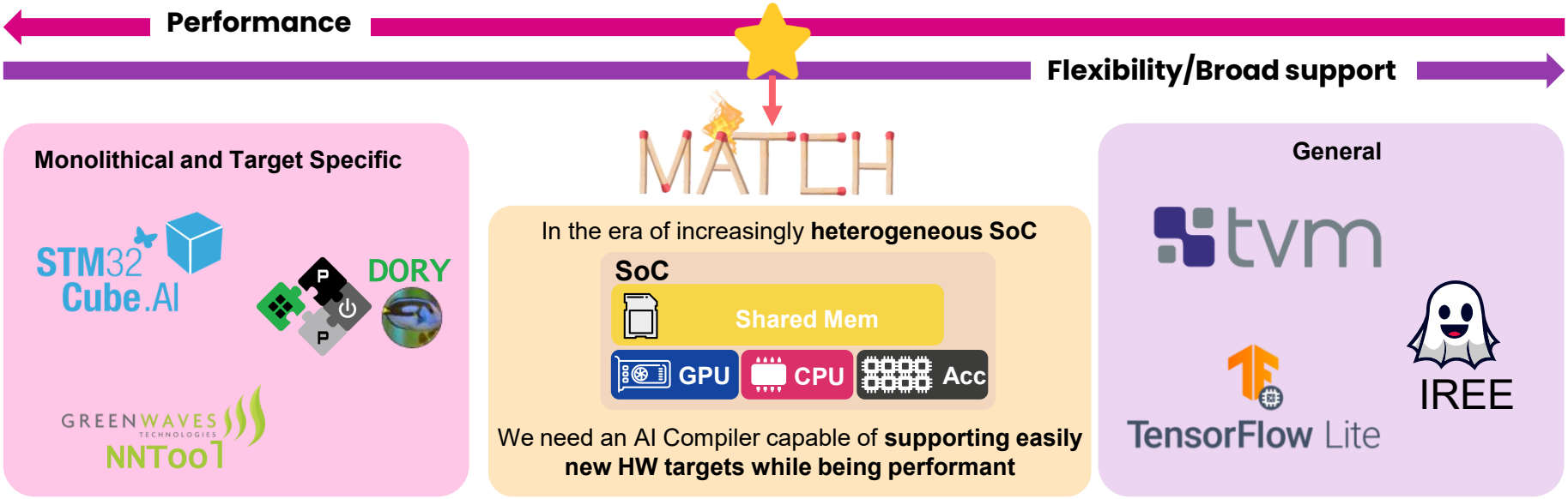
AI “Compilers”

1. Exploit the graph similarity to recognize known patterns of neural networks and associate to target backend nodes.
2. Consider the target hardware to optimize memory patterns and resource allocation.
3. Optimize the arithmetic workload, maximizing parallelization and exploiting target hardware ISA.

Copyright © 2025

AI Compiler – MATCH (1)

AI Compilers ecosystem:



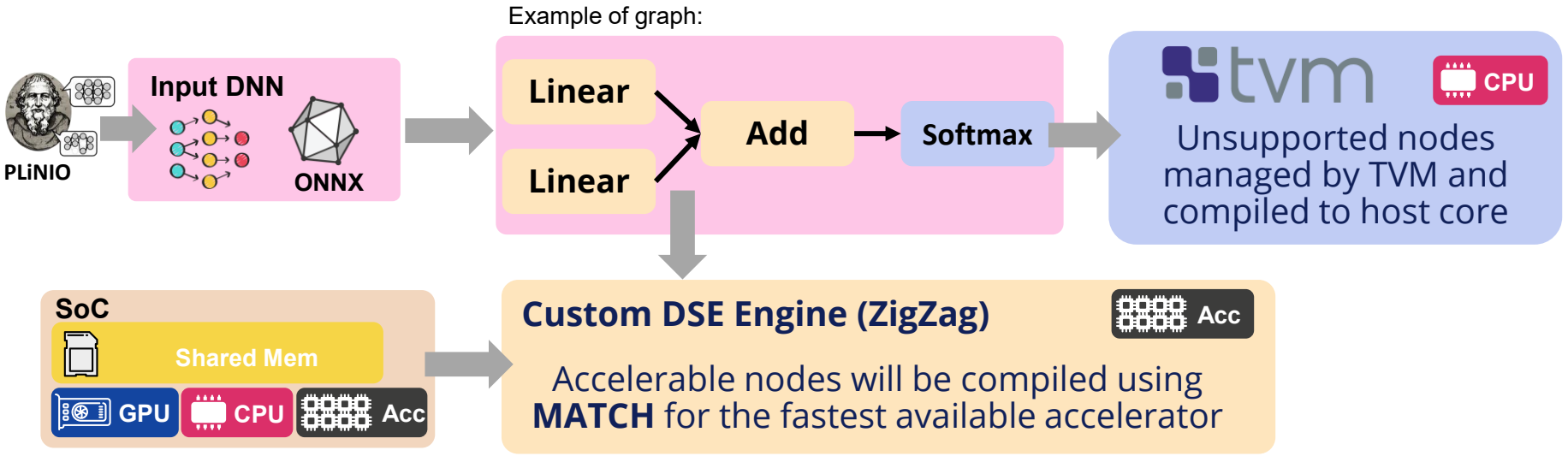
Copyright © 2025

AI Compiler – MATCH (2)

- **MATCH** is built around **Apache TVM.**
- **Rationale:** Exploit TVM's **generality** while enabling **customization** for heterogeneous accelerators.
- **Reuse:** Front-end, IR(s), Memory planner, lowering for host CPU, …
- **Customize (with BYOC):** Code generation for accelerated layers; loop ordering, tiling, data transfers scheduling, …

Copyright © 2025

MATCH Execution Flow

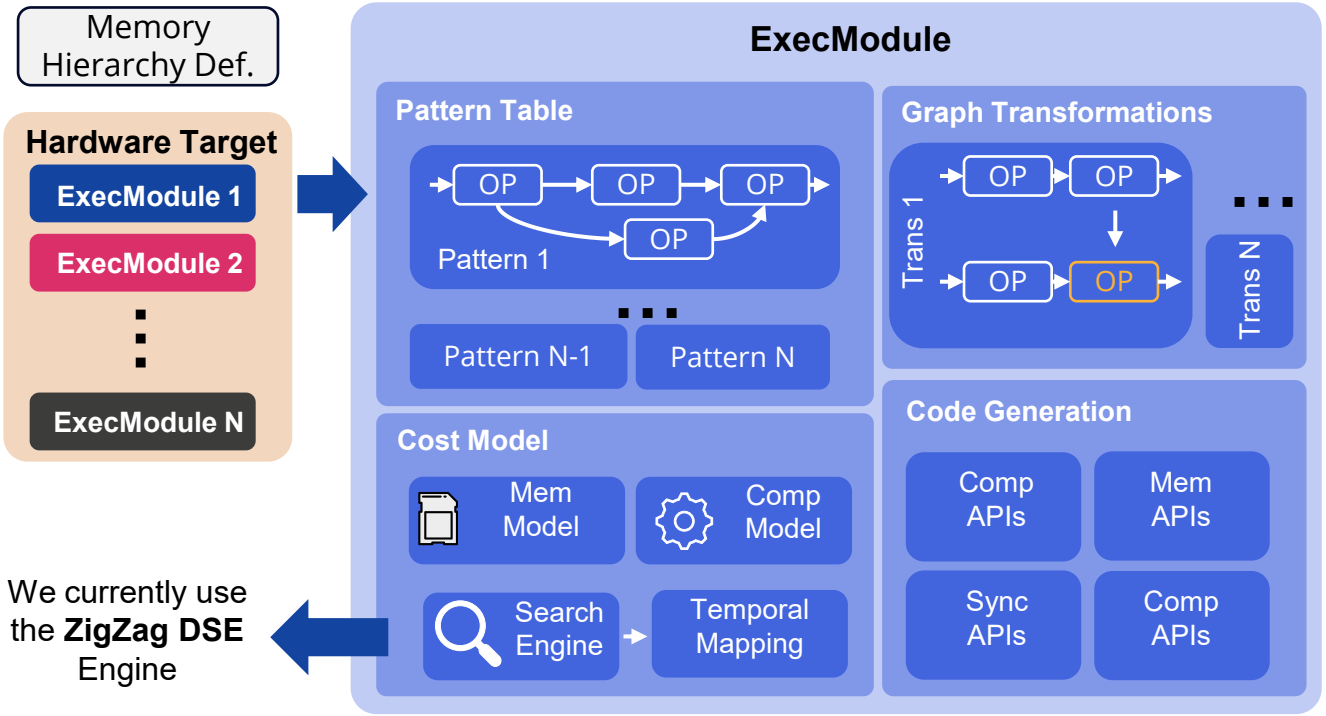


- Streamlined **HW target** definition to guide codegen and optimization
- Uses **pattern matching** to identify accelerable ops (sequences)

Copyright © 2025

MATCH HW Targets

MATCH abstracts accelerators as ExecModules:



1. **Table of supported DNN ops. patterns**
2. **Graph level Transformations** (fusion, ops replacement, etc.)
3. **Cost model for DSE**
4. **Code generation APIs:**
   - Synch APIs
   - Platform APIs
   - Memory APIs
   - Computational APIs

Copyright © 2025

Pattern Checker and Code Generation

How to Offload a node?

1. Check ExecModule supported patterns and constraints
2. Offload Pattern Parameters to ZigZag
3. Use ZigZag's output and HW Target information to generate the final code

TVM IR Module

```
(Pdb) mod["main"]
fn (%input: Tensor[(1, 3, 16, 32), int8]) {
  %0 = nn.conv2d(%input, meta[relay.Constant][0],
  %1 = nn.bias_add(%0, meta[relay.Constant][1]);
  %2 = right_shift(%1, 1);
  %3 = clip(%2, a_min=-128f, a_max=127f);
  cast(%3, dtype="int8")
}
```

ZigZag Workload

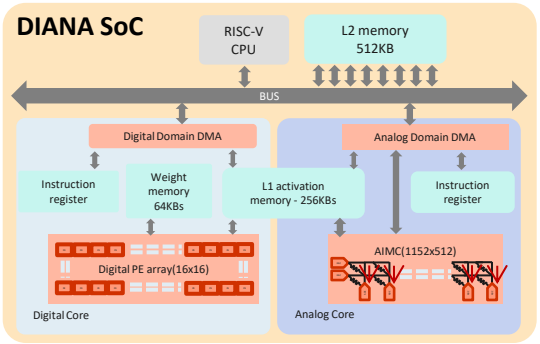
```
workload = {
    0: {
        'operator_type': 'Conv',
        'equation': 'O[b][k][oy][ox]+=W[k][c][fy][fx]*I[b][c][iy][ix]',
        'dimension_relations': ['ix=1*ox+1*fx', 'iy=1*oy+1*fy'],
        'loop_dim_size': {'B': 1,'K':32,'C':3,'OY':16,'OX':32,'FY':1,'FX':1},
        'operand_precision': {'O': 32, 'O_final': 8, 'W': 8, 'I': 8},
        'padding': {'IY':(0,0), 'IX': (0,0)},
        'operand_source': {'W': [], 'I': []},
        'constant_operands': ['W'],
        'operand_source_dimension_mapping': {'I':{'IX':'OX','IY':'OY','C':'K'}}
    }
}
```

Generated Code

```
1:  //weights and input transfers L2 to L1
2:  memory_transfer_weights()
3:  memory_transfer_inputs()
4:  //for loops
5:  for K1 ← 0 to 2 do
6:      for K ← 0 to 2 do
7:          //computational function
8:          conv2d()
9:          //output L1 to L2 transfer
10:         memory_transfer_output()
11:     end for
12: end for
```

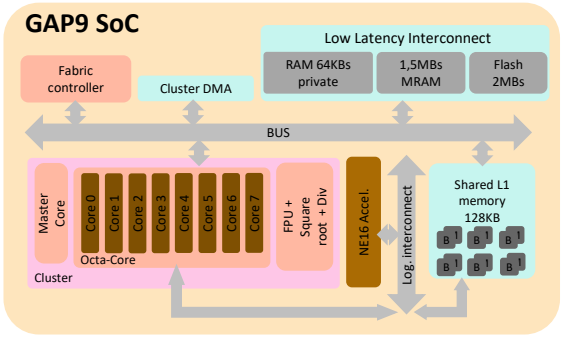

Copyright © 2025

Experimental Results



- **Analog Ternary accelerator** and **Digital 8-bits accelerator**
- **Shared L1 activation Mem** of 256 kB

MATCH chooses best HW Exec Module
conditioned on layers' Hyper-params



- **Multi-Core Cluster** w/ RV DSP Estension and **NE16 AI Accelerator**
- **Shared L1 Mem** of 128 kB

MATCH chooses best HW ExecModule
conditioned on layers' Hyper-params

MLPerf Tiny

MLC

MobileNet
ResNet
DSCNN
DAE

Proprietary + Platform Specific

tvm HTVM	MATCH
OoM	1.0x	**1.01x**
1.0x	104x	**168x**
1.0x	**6.8x**	6.7x
1.0x	**7.2x**	6.5x

Better/On-Par

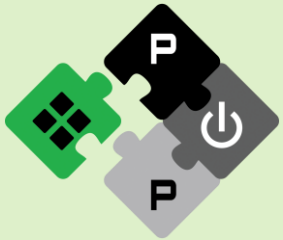
Slightly worse due to template generality

tvm DORY	NNTool	MATCH
1.0x	20.9x	**147x**	47.8x
1.0x	60.4x	**395x**	162x
1.0x	19.9x	**123x**	53x
1.0x	11.8x	**23.5x**	11.3x

2.4x Avg speedup wrt OpenSource SotA

Copyright © 2025

Hardware Simulation



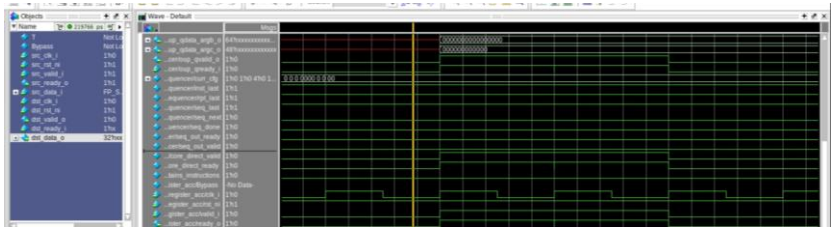
GVSoC

open source

Copyright © 2025

Simulation of RISC-V based platform

- **Simulate functionality and performance**



Cycle by cycle

gem5

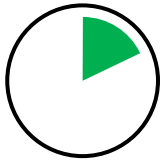
gvsoc/gvsoc

2 Contributors 0 Issues 5 Stars 9 Forks

Event by event

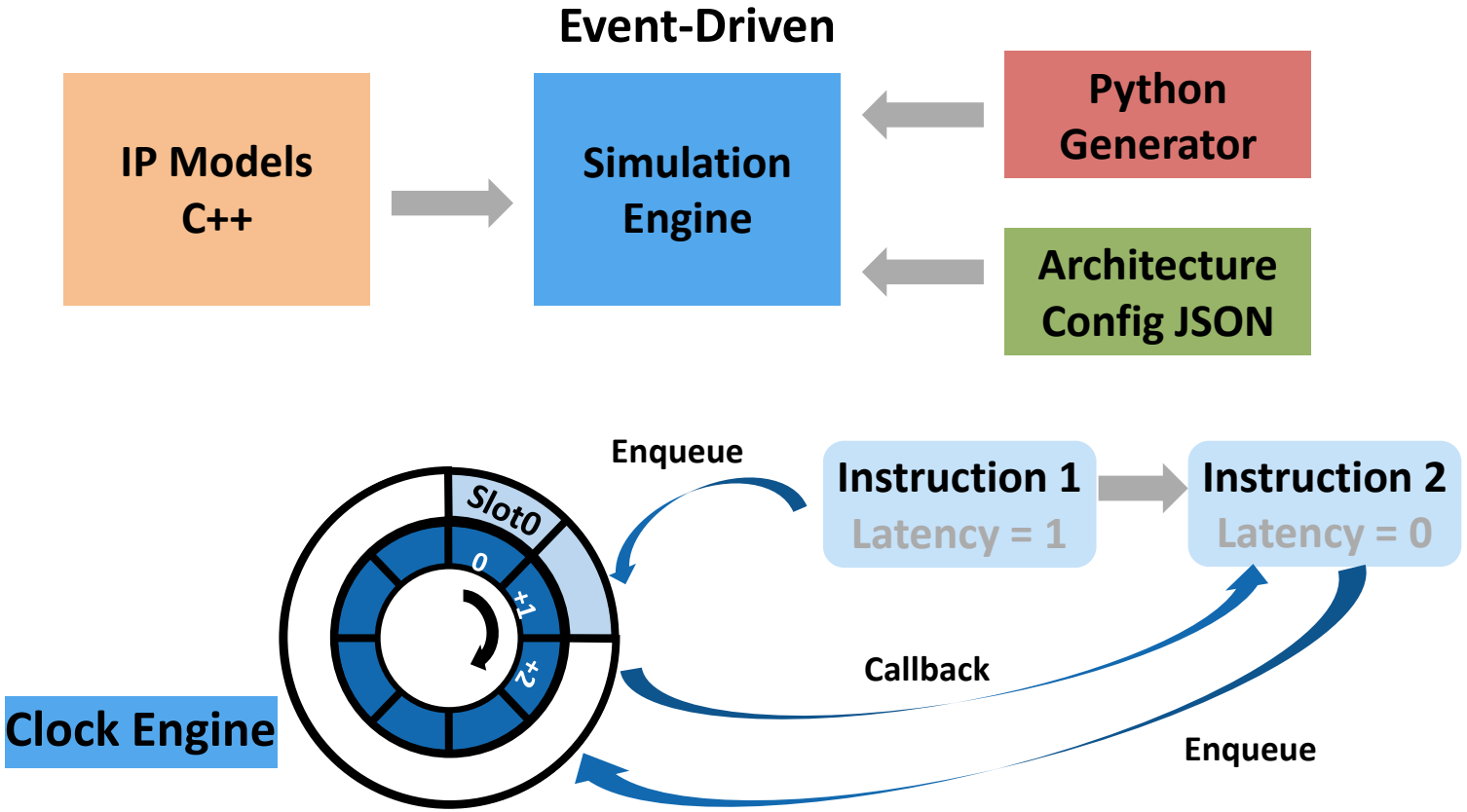
Cycle-accurate simulator
Describe HW in RTL

Event-driven simulator
Describe HW in high-level languages



Copyright © 2025

GVSoC Architecture

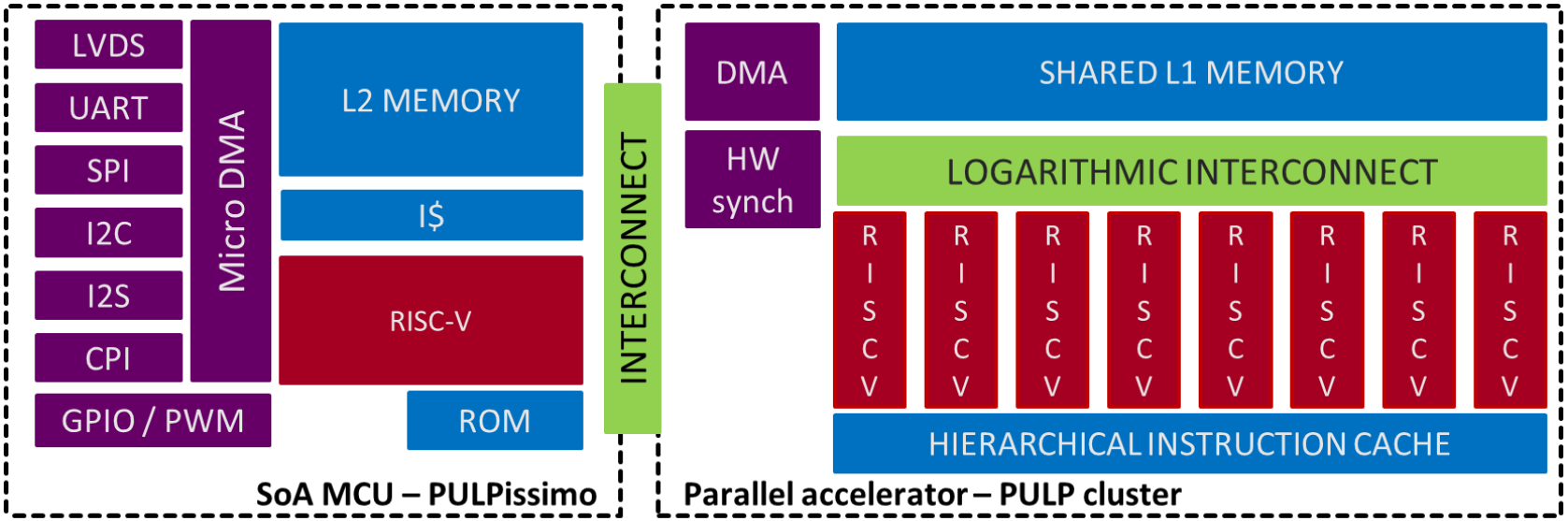


Copyright © 2025

PULP-Open

The target platform in the next examples is **PULP-Open** (https://github.com/pulp-platform/pulp)

- All the RISCV blocks are **CV32E40P cores**
- The MCU on the left may be replaced by a **CVA6 processor** (*experimental*)



Copyright © 2025

GVSoC in a Nutshell

Clone the repository

```
git clone https://github.com/gvsoc/gvsoc.git
cd gvsoc
git submodule update --init --recursive
pip3 install -r core/requirements.txt
pip3 install -r gapy/requirements.txt
```

Compile GVSoC (see the README file for specific configuration settings):

```
make all TARGETS=pulp-open
```

Test:

```
./install/bin/gvsoc --target=pulp-open --binary examples/pulp-open/hello image flash run
```

System generator (root file): `pulp/pulp/chips/pulp_open/pulp_open.py`

Copyright © 2025

Python Generator: Getting properties from JSON config

```
class Cluster(st.Component):
    """
    Cluster subsystem

    Attributes
    ----------
    cid : int
        Cluster ID

    """

    def __init__(self, parent, name, config_file, cid: int=0):
        super(Cluster, self).__init__(parent, name)

        #
        # Properties
        #

        self.add_properties(self.load_property_file(config_file))

        nb_pe               = self.get_property('nb_pe', int)
        cluster_size        = self.get_property('mapping/size', int)
        self.cluster_offset = cluster_size * cid
        self.cluster_base   = self.get_property('mapping/base', int)
        self.cluster_alias  = self.get_property('alias', int)
        ne16_irq            = self.get_property('pe/irq').index('acc_0')
        dma_irq_0           = self.get_property('pe/irq').index('dma_0')
        dma_irq_1           = self.get_property('pe/irq').index('dma_1')
        dma_irq_ext         = self.get_property('pe/irq').index('dma_ext')
        timer_irq_0         = self.get_property('pe/irq').index('timer_0')
        timer_irq_1         = self.get_property('pe/irq').index('timer_1')
        first_external_pcer = 12
        has_ne16 = False
```

Example: cluster

Location:
pulp/pulp/chips/pulp_open/cluster.py

Properties can be loaded from the JSON architectural configuration

Copyright © 2025

Python Generator: Describing components

```
#
# Components
#

# L1 subsystem
l1 = L1_subsystem(self, 'l1', self)

# Cores
pes = []
for i in range(0, nb_pe):
    pes.append(iss.ClusterCore(self, 'pe%d' % i, cluster_id=cid, core_id=i))

# Icache
icache = Hierarchical_cache(self, 'icache', self.get_property('icache/config'))

# Event unit
event_unit = Event_unit(self, 'event_unit', self.get_property('peripherals/event_unit/config'))

# Cluster interconnect
cluster_ico = Router(self, 'cluster_ico', latency=2)

# Periph interconnect
periph_ico = Router(self, 'periph_ico')

# Demux periph interconnect
demux_periph_ico = Router(self, 'demux_periph_ico')

# MCHAN
mchan = Mchan(self, 'dma', nb_channels=nb_pe+1)

# Timer
timer = Timer(self, 'timer')

# Cluster control
cluster_control = Cluster_control(self, 'cluster_ctrl', nb_core=nb_pe)

if has_ne16:
    # NE16
    ne16 = Ne16(self, 'ne16')

# Icache controller
icache_ctrl = Icache_ctrl(self, 'icache_ctrl')
```

Example: cluster

Location:
pulp/pulp/chips/pulp_open/cluster.py

Components are instantiated by specifying *name* and *parameters*

Copyright © 2025

Python Generator: Adding bindings

```
#
# Bindings
#

# L1 subsystem
for i in range(0, nb_pe):
    self.bind(l1, 'dma_%d' % i, mchan, 'in_%d' % i)
    self.bind(l1, 'dma_alias_%d' % i, mchan, 'in_%d' % i)
    self.bind(l1, 'event_unit_%d' % i, event_unit, 'demux_in_%d' % i)
    self.bind(l1, 'event_unit_alias_%d' % i, event_unit, 'demux_in_%d' % i)

self.bind(l1, 'cluster_ico', cluster_ico, 'input')

# Cores
for i in range(0, nb_pe):
    self.bind(pes[i], 'data', l1, 'data_pe_%d' % i)
    self.bind(pes[i], 'data_debug', l1, 'data_pe_%d' % i)
    self.bind(pes[i], 'fetch', icache, 'input_%d' % i)
    self.bind(pes[i], 'irq_ack', event_unit, 'irq_ack_%d' % i)
    self.bind(self, 'halt_pe%d' % i, pes[i], 'halt')

    for j in range(first_external_pcer, first_external_pcer+5):
        self.bind(pes[i], 'ext_counter[%d]' % j, l1, 'ext_counter_%d[%d]' % (i, j))

# Icache
self.bind(icache, 'refill', cluster_ico, 'input')

# Event unit
for i in range(0, nb_pe):
    self.bind(event_unit, 'clock_%d' % i, pes[i], 'clock')
    self.bind(event_unit, 'irq_req_%d' % i, pes[i], 'irq_req')
```

Example: cluster

Location:
pulp/pulp/chips/pulp_open/cluster.py

Components have in/out ports that must be connected (*bindings*)

Copyright © 2025

Python generator: Connection with C++ modeling

```python
import gvsoc.systree

class Memory(gvsoc.systree.Component):
    def __init__(self, parent: gvsoc.systree.Component, name: str, size: int, width_log2: int=2,
            stim_file: str=None, power_trigger: bool=False,
            align: int=0, atomics: bool=False, latency=0, memcheck_id: int=-1, memcheck_base: int=0,
            memcheck_virtual_base: int=0, memcheck_expansion_factor: int=5):

        super().__init__(parent, name)

        self.add_sources(['memory/memory.cpp'])

        # Since atomics are slowing down the model, this is better to compile the support only
        # if needed. Note that the framework will take care of compiling this model twice
        # if both memories with and without atomics are instantiated.
        if atomics:
            self.add_c_flags(['-DCONFIG_ATOMICS=1'])

        self.add_properties({
            'size': size,
            'stim_file': stim_file,
            'power_trigger': power_trigger,
            'width_bits': width_log2,
            'align': align,
            'latency': latency,
            'memcheck_id': memcheck_id,
            'memcheck_base': memcheck_base,
            'memcheck_virtual_base': memcheck_virtual_base,
            'memcheck_expansion_factor': memcheck_expansion_factor,
        })
```

Example: memory

Location:
core/models/memory/memory.py

- Link to CPP model (memory.cpp)
- Parametrizable arguments

Copyright © 2025

What can I do with GVSoC?

- Creating a new platform with available components
- Modelling new components
- Developing SW for prototype architectures
- HW/SW codesign tasks
- Integration with other tools (e.g., cyber-physical system modelling tools)
- …

Copyright © 2025

Thanks

Questions?

Contacts:

- Daniele Jahier Pagliari: daniele.jahier@polito.it
- Alessio Burrello: alessio.burrello@polito.it
- Giuseppe Tagliavini: giuseppe.tagliavini@unibo.it

Copyright © 2025