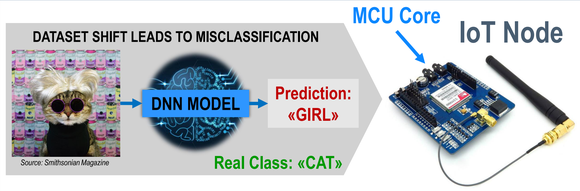
On-Device Learning (ODL) on RISC-V Multicore MCUs

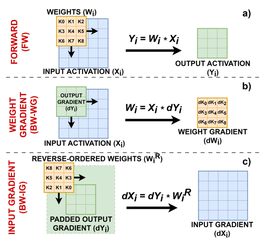
Davide Nadalini[1,2], Manuele Rusci[3], Luca Benini[2,4], and Francesco Conti[2]

[1]Politecnico di Torino, Torino, Italy, [2]Università di Bologna, Bologna, Italy, [3]Katholieke Universiteit Leuven, Leuven, Belgium, [4]ETH Zurich, Zurich, Switzerland

Motivation



On-Device Learning (ODL): the ability to "update a model *without* data leaving your users' devices" (Tensorflow Lite)

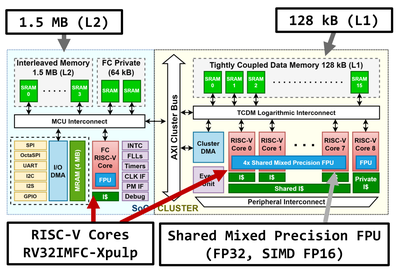


Many DNNs rely on **Convolutions** trained with **Floating-Point BackPropagation**

On-Device Inference & Adaptation **limited by MCU** performances / resources

Our Goal: enabling ODL on ultra-low-power MCUs

Our target: the PULP Platform[1]



PULP (Parallel Ultra-Low-Power): computational platform for energy-efficient and scalable edge computing based on **RISC-V cores**.

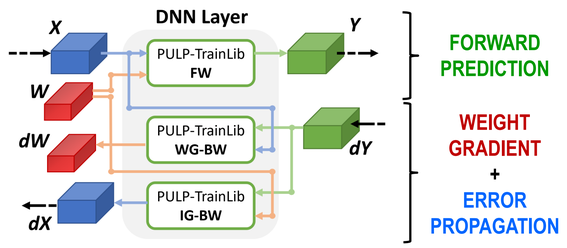
1) PULP-TrainLib: Enabling On-Device Training for RISC-V Multi-Core MCUs through Performance-Driven Autotuning[2]

Davide Nadalini[1,2], Manuele Rusci[2], Giuseppe Tagliavini[2], Leonardo Ravaglia[2], Luca Benini[2,3], and Francesco Conti[2]

[1]Politecnico di Torino, [2]Università di Bologna, [3]ETH Zurich

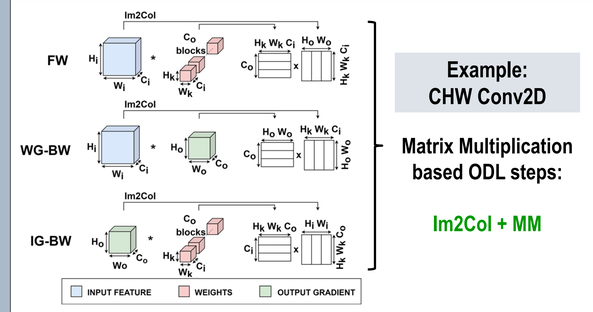
1) PULP-TrainLib, the first **open-source[3] training library for RISC-V multicore MCUs** with Matrix-Multiplication (MM)-based performance-tunable Floating-Point (FP) primitives.

BackPropagation-based ODL primitives of **CNN models**



[1]PULP Platform: https://pulp-platform.org/

[2]D. Nadalini, M. Rusci, G. Tagliavini, L. Ravaglia, L. Benini, and F. Conti, "PULP-TrainLib: Enabling On-Device Training for RISC-V Multi-Core MCUs through Performance-Driven Autotuning"



Example: CHW Conv2D

Matrix Multiplication based ODL steps:

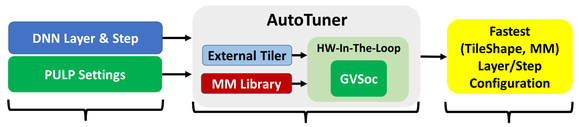
Im2Col + MM

Workhorse: FP32 **Unrolled & Parallel MM** Library

TABLE 1: PULP-TRAINLIB'S OPTIMIZED MM ALGORITHMS

MM Type	Unrolling Factor	Parallelism	Description
mm		N or M	Naive MM
mm_uJ	$J = 2$	N or M	Unroll J inner products in K
mm_unroll_1xV	$U = 2, 4, 8$	N or M	Unroll U columns of C
mm_unroll_Ux1	$V = 2, 4, 8$	N or M	Unroll V rows of C
mm_unroll_UxV	$U, V = 2, 4$	N or M	Unroll U,V rows and columns of C

2) AutoTuner, an **HW-in-the-loop (HIL)-based tool** to optimize PULP-TrainLib's primitives according to the **DNN layer and step** and the target **PULP Platform settings**



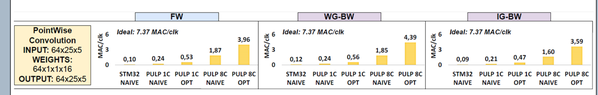
Layer type/size & step, **PULP** SoC **Setup ($L1_{size}$, N_{cores})**

Layer is tiled if exceeds L1, **exhaustive search with HIL (PULP GVSoC simulator)** for optimal setup

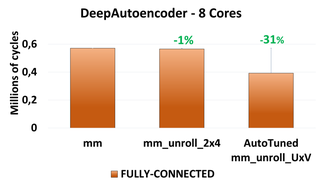
Optimal **tile size & MM algorithm** for layer/step

3) A detailed analysis of PULP-TrainLib and AutoTuner on an **8-Core PULP Platform**

PULP GVSoC Setup: $L1_{SIZE} = 64\ kB$, $N_{CORES} = \{1, 8\}$

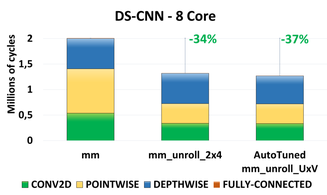


- Up to **2.4x** speedup (**autotuned vs one-size-fits-all**, **4.39 MAC/clk** on 8 RISC-V cores)
- **36.6x less latency** vs unoptimized **STM32**



PULP SoC @400 MHz: DeepAutoencoder < 1 ms DS-CNN < 4 ms

Ideal estimation of a single training step of two **TinyML Perf models**[4] (with tiling and AutoTuner)



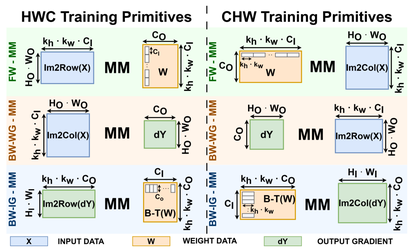
[3]**PULP-TrainLib** repository: https://github.com/pulp-platform/pulp-trainlib

[4]TinyMLPerf Benchmarks https://github.com/mlcommons/tiny/tree/master/benchmark

2) Reduced Precision Floating-Point Optimization for Deep Neural Network On-Device Learning on MicroControllers[5]

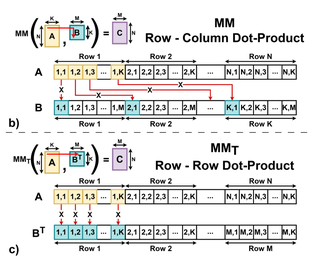
Davide Nadalini[1,2], Manuele Rusci[3], Luca Benini[1,4] and Francesco Conti[1]

[1]Università di Bologna, [2]Politecnico di Torino, [3]Katholieke Universiteit Leuven, [4]ETH Zurich



Main Idea: Row-Column MMs are inefficient with SIMD (Sparse load-stores)

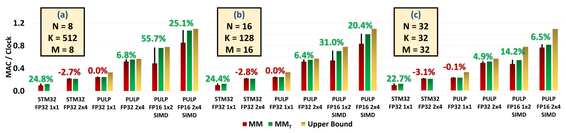
1) Row-Column MM is replaced by MM_T (SIMD row-row MM)



Memory Accesses: adjacent for A, sparse for B

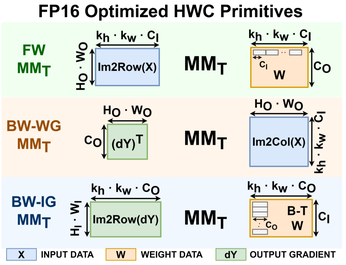
Memory Accesses: adjacent for A and B

FP16 SIMD MM_T vs MM: loop unrolling on 1 Core



On 8 cores, up to 7.89 MAC/clk for MM_T (1.91x FP32 MM)

2) Primitive-level optimization of ODL training steps (BackPropagation-based)

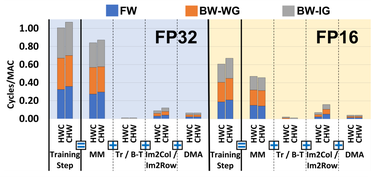


E.g. FP16 Conv2D:

- **Weights** stored in **transposed** form
- Primitive **matrix expression reshaped** to exploit MM_T kernels

E.g: Conv2D (8x8x16) input, (3x3) kernel, (8x8x16) output

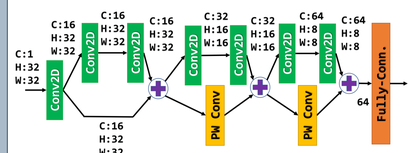
- FP16 **HWC 11% faster** than CHW
- FP16 **1.66x faster** than FP32



[5]D. Nadalini, M. Rusci, L. Benini, F. Conti, "Reduced Precision Floating-Point Optimization for Deep Neural Network On-Device Learning on MicroControllers"

3) FP16 optimization of full TinyML[2] models on a GreenWaves Technologies' GAP9 SoC @370MHz, 8 Cores

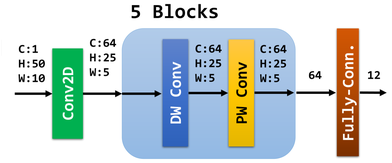
More accurate estimate: DMA transfers, Im2Col/Im2Row, real SoC (shared FPUs)



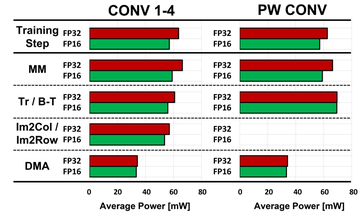
ResNet8 for Image Classification (FP16) **17.1 ms/sample (1.88x wrt FP32)**

5 Blocks

DS-CNN for Keyword Spotting (FP16) **6.4 ms/sample (1.8x wrt FP32)**



4) Power Analysis of ODL Primitives on the **GAP9 SoC** (FP32 vs FP16)



- Up to **63.6 mW** (MM-dominated)
- 4 – 25.4 uJ / training step (FP32)
- **3.8 – 17.4 uJ / training step (FP16)**

5) Real-time Continual Learning on MCU Targets (comparison with SoA)

Table 7: Latency and Energy Evaluation for Continual Learning on MCUs

	Platform	Latency (s)		Energy (J)	
Aifes[30]	STM32L4 @ 80 MHz	DW21	236655	DW21	7739
		DW23	142157	DW23	4649
		LIN27	102.4	LIN27	3.35
PULP-TrainLib[31]	Greenwaves GAP9 @ 370 MHz	DW21	504.1	DW21	32.24
		DW23	303.8	DW23	19.43
		LIN27	0.89	LIN27	0.06
This Work	Greenwaves GAP9 @ 370 MHz	DW21	308.5	DW21	18.68
		DW23	185.9	DW23	11.26
		LIN27	0.89	LIN27	0.06

Continual Learning[6] on a **MobileNet**: new class learned in 3-5 minutes with FP16 on GWT GAP9 (retraining the last 5-10 layers)

Conclusion

- Enabling ODL on MCUs allows **real-time BackProp-based learning** on IoT end nodes (Continual, Online, …, Learning)
- Reduced Precision enables **fast (1.66x *FP32*) backend for ODL** with high enough precision
- **Power consumption < 100 mW** on Multi-Core RISC-V MCUs

[6]Latent Replay for Real-Time Continual Learning: https://arxiv.org/abs/1912.01100