BioTrain: Sub-MB, Sub-50mW On-Device Fine-Tuning for Edge-AI on Biosignals

Run Wang*, Victor J. B Jung*., Philip Wiese*, Sebastian Frey*, Giusy Spacone*

Francesco Conti‡, Alessio Burrello†, Luca Benini* ‡

*Integrated Systems Laboratory, ETH Zürich

‡ DEI, University of Bologna

†DAUIN, Politecnico di Torino

PULP Platform

Open Source Hardware, the way it should be!

@pulp_platform

pulp-platform.org

youtube.com/pulp_platform

Motivation: Why Adapt On-Device?

Biosignals drift → deployed model decays

- **Cross-subject variability**
- **Non-stationarity**
  - electrode–skin impedance, sweat, sensor shift over time
- **Result:** severe domain shift, degraded post-deployment accuracy

On-device adaptation is the answer

- **Personalize** to each user and over time, in situ
- **Privacy** raw biosignals never leave the device
- **Reliability** low-latency, no cloud dependency

The adaptation lifecycle

Day 1

New-user calibration: cold-start fine-tuning on the first labeled session

Day N

Longitudinal adaptation: periodic tuning to track physiological drift

The catch(es)

- Both phases need real gradient-based training
- Wearable MCUs have limited (KBs) fast on-chip memory and power budget (mWs)

The Challenge: Full Backpropagation Doesn't Fit on an MCU

The memory wall of training

- Full-network backpropagation stores all intermediate activations + gradients
- Scales with batch size and sequence length

8-10x Memory Footprint of Inference

vs A few **MB** on-chip memory

Cannot run on-chip!

Previous SotA approaches

Last-layer / LP
Low cost, but limited to small distribution shifts

Sparse / pruned BP
Hand-tuned and brittle; limited capacity

Small batch size
Lower memory, but slower and less stable

The bottleneck is memory management
Missing: compiler-driven tiling and static allocation of the training graph.

Related Work: On-Device Training Frameworks

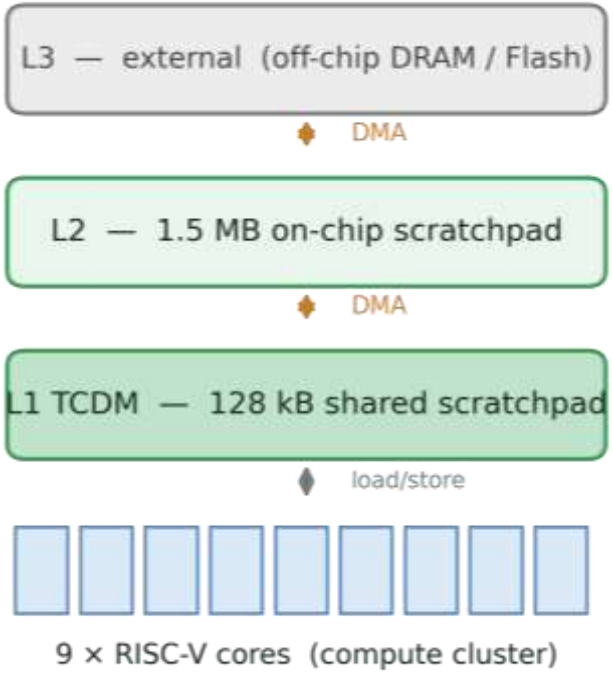
Framework	Algorithm	Depth	Batch>1	Acc. vs. offline	Compiler
TinyOL	Last-layer LP	Last layer	✗	↓ 5–10%	✗
MiniLearn	Prune + FT	Sparse	✓	↓ 2–10%	✗
TTE	Sparse BP	Sparse	✗	≈	✗
TinyTTA	Early-exit TTA	Shallow	✗	N/A	✗
AIfES	Full BP	Full	✓	≈ (tiny nets)	≈
BioTrain	**Full BP**	**Full**	✓	**≈ offline**	✓

THE GAP Prior work trades away depth, batch size, or accuracy to fit memory.
BioTrain keeps all three and adds the compiler.

Acc. vs. offline: relative to offline full-network backpropagation.

Target Hardware

Wearable MCU-class SoC[9]: parallel RISC-V cluster with an explicitly managed scratchpad hierarchy.



- 9-core RISC-V compute cluster
- 3-level memory: L1 TCDM 128 kB · L2 1.5 MB on-chip · L3 external
- Scratchpads, not hardware caches no automatic caching
- **So software must explicitly tile to L1/L2, schedule DMA, and statically allocate**

Our Work: From Deeploy to BioTrain

BioTrain is the CNN extension of TrainDeeploy: our line of work on the Deeploy compiler.

Deeploy[1]

Inference compiler
tiling + static alloc + C codegen

Onnx4Deeploy

ONNX autograd builds the training graph

TrainDeeploy[2]

Transformer training, LoRA PEFT

BioTrain

★ **this work**

Full-network CNN BP for biosignals on GAP9

BioTrain extends TrainDeeploy's stack from Transformer LoRA/PEFT to **full-network CNN backpropagation** Deeploy's tiling + static alloc, now for CNN gradient kernels.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Contributions: What BioTrain Delivers

1 A training compiler

Extends Deeploy / TrainDeeploy to CNNs. Bare-metal C for forward + backward, with tiled CNN gradient kernels (PULP-TrainLib).

2 Memory that fits on chip memory

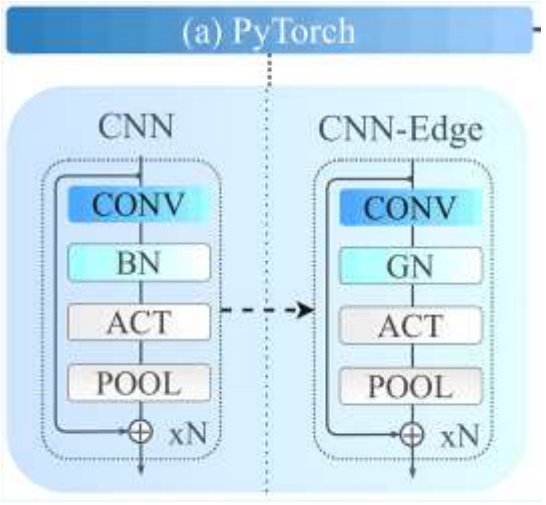
Compiler optimized memory allocation. Gradient Accumulation.

3 Real-world validation

EEG + EOG on GAP9: +35% vs. no-adapt, +7% vs. last-layer. 17 / 85 samples·s^{-1}, < 50 mW.

BioTrain Framework Overview

BioTrain extends the Deeploy compiler (via TrainDeeploy) from inference to full-network CNN training.



a) PyTorch
Standard CNN; swap BN→GN for edge

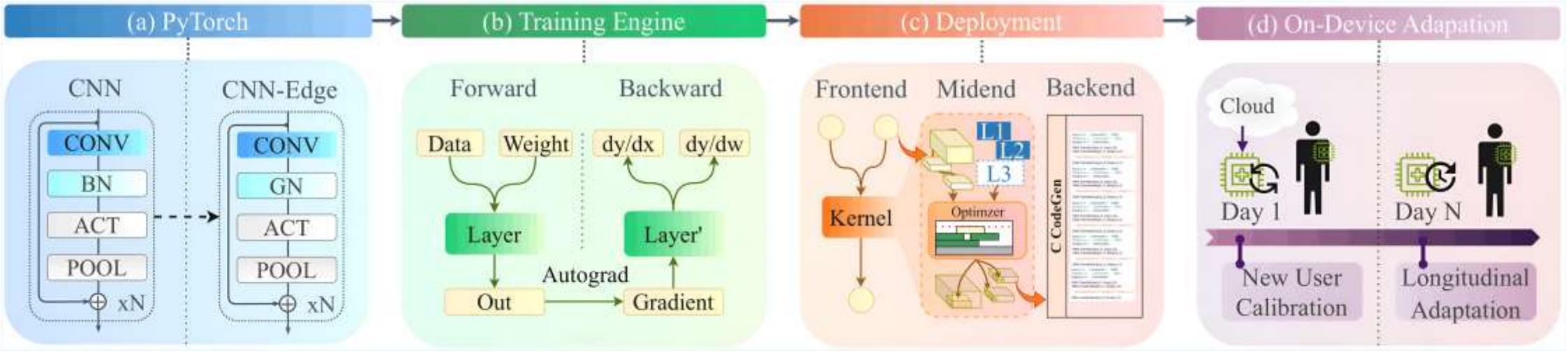
b) Training engine
Autograd builds forward + backward graph

c) Deployment
Frontend→midend→back end → bare-metal C

d) On-device
Day-1 calibration, then longitudinal adaptation

BioTrain Framework Overview

BioTrain extends the Deeploy compiler (via TrainDeeploy) from inference to full-network CNN training.



a) PyTorch
Standard CNN; swap BN→GN for edge

b) Training engine
Autograd builds forward + backward graph

c) Deployment
Frontend→midend→back end → bare-metal C

d) On-device
Day-1 calibration, then longitudinal adaptation

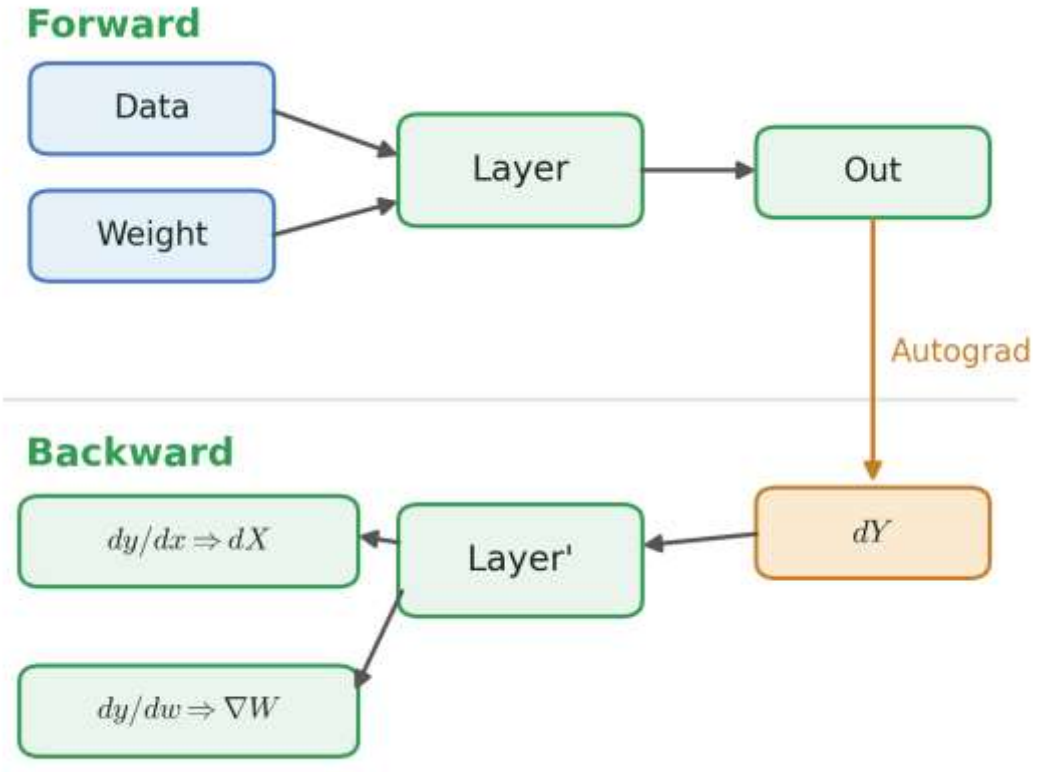
Training Engine: Config & Graph

Training configuration

- Full-network backprop: all layers trainable (not last-layer only)
- **Accumulation:** effective batch > 1 at batch-size-1 peak memory

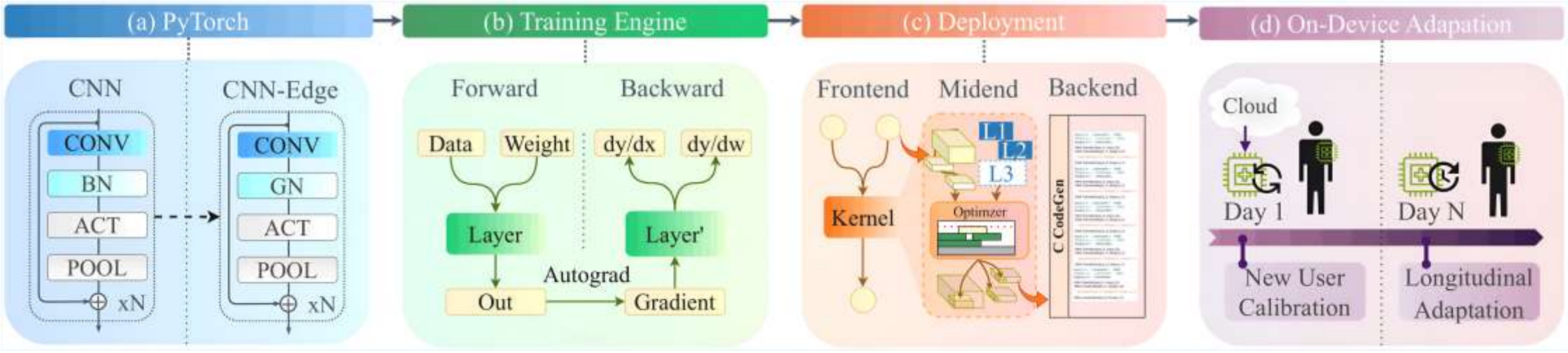
Edge-friendly choices

- **BN→GN**: per-sample stats, no cross-sample sync for small batch size
- **Precision**: FP32, batch 8 per accumulation step



BioTrain Framework Overview

BioTrain extends the Deeploy compiler (via TrainDeeploy) from inference to full-network CNN training.



a) PyTorch
Standard CNN; swap BN→GN for edge

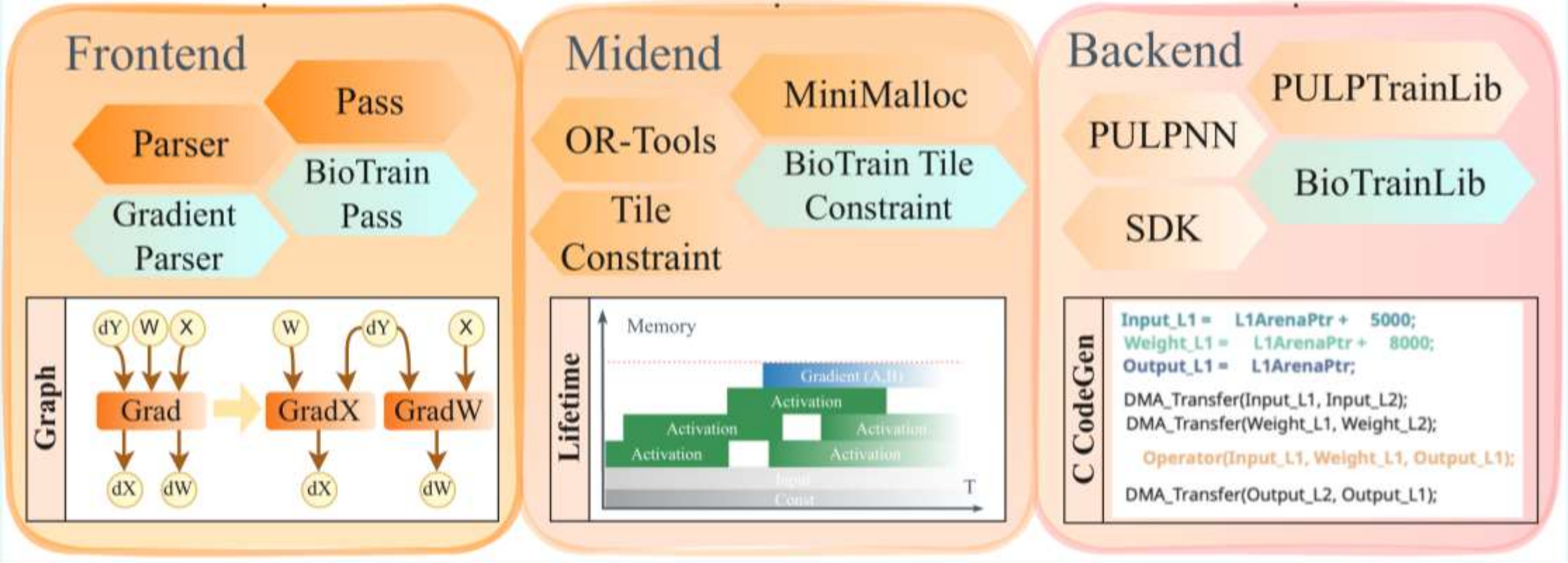
b) Training engine
Autograd builds forward + backward graph

c) Deployment
Frontend→midend→backend → bare-metal C

d) On-device
Day-1 calibration, then longitudinal adaptation

Deployment Stack: Frontend → Midend → Backend

Light-blue blocks are the new BioTrain contributions in the Deeploy pipeline.



Frontend

New BP operators;
Gradient decomposition

Midend

OR-Tools tiling + MiniMalloc static allocation

Backend

Codegen with gradient kernels

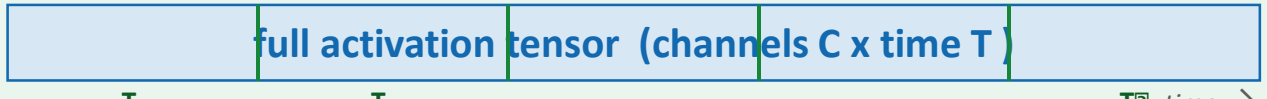
Inside the Midend: Tiling + Static Allocation

Biosignals are wide in time span but narrow in number of channels → tile along the temporal axis to orchestrate DMA ↔ compute on-chip.

Tiling the tensors: stream the temporal axis through L1

L2 · 1.5 MB on-chip (slow)

split along time → tiles T_1 … T⍰ (each fits L1)



DMA load: tile → L1

DMA store: result → L2

✗ **full tensor ≫ 128 kB can't fit L1**

L1 · 128 kB · fast

T⍰ ✓ **fits L1**

Multiple- RISC-V cluster

compute gradient on tile

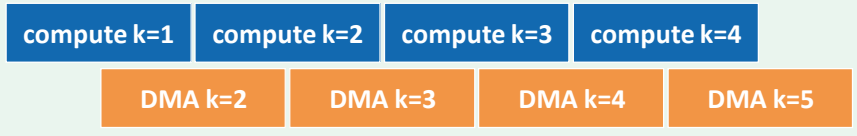
∇W , dX , norm

Why tile? Overlap DMA with compute

Naive (no double buffer):

Serial: PEs idle during every transfer *time →*

Tiled + double buffering:



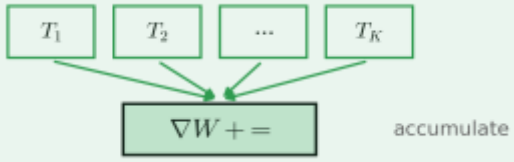
load k+1 while computing k → DMA hidden under compute

peak L1 ≤ 128 kB ✓ · fits 1.5 MB L2 ✓ · near-full PE use

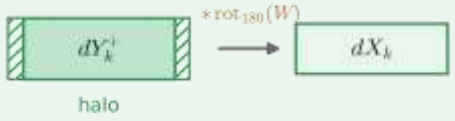
Inside the Midend: Tiling + Static Allocation

Biosignals are wide in time span but narrow in number of channels → tile along the temporal axis to orchestrate DMA ↔ compute on-chip.

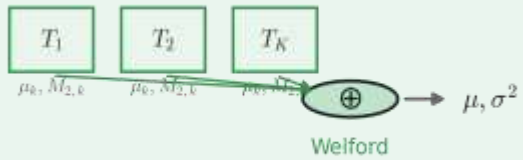
Weight gradient ∇W



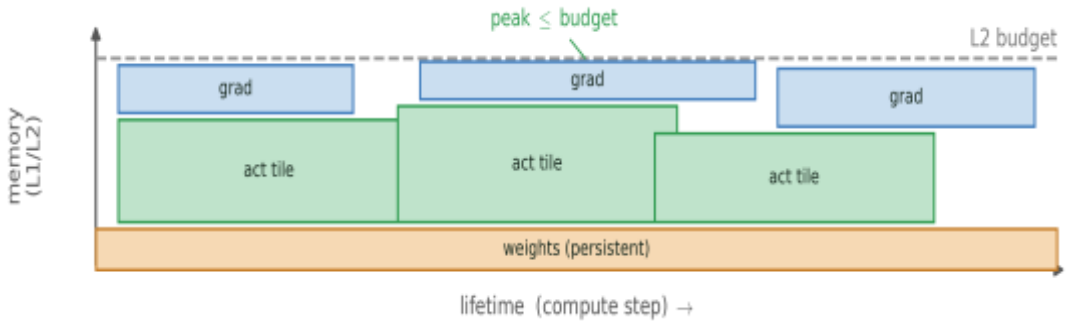
Input gradient dX



Normalization grad



Static allocation: MiniMalloc



Lifetime-aware 2-D packing of tiled tensors and full tensors into fixed L1/L2 offsets → peak memory minimized for L1/L2, no runtime malloc.

Backend: C Code Generation

```
// static L1/L2 arenas (MiniMalloc)
Weight_L1 = L1ArenaPtr + 0;       // resident
Input_L1  = L1ArenaPtr + 5000;
Output_L1 = L1ArenaPtr + 8000;

DMA_Transfer(Weight_L1, Weight_L2);   // load once

for (k = 0; k < N_TILES; k++) {       // tile loop
  DMA_Transfer(Input_L1, Input_L2+k);  // load tile
  DMA_Wait();
  Operator(Input_L1, Weight_L1, Output_L1);
  DMA_Transfer(Output_L2+k, Output_L1);  // store tile
}
```

Backend (c): bare-metal C

- Static L1/L2 arena allocation (MiniMalloc)

Backend: C Code Generation

```
// static L1/L2 arenas (MiniMalloc)
Weight_L1 = L1ArenaPtr + 0;       // resident
Input_L1  = L1ArenaPtr + 5000;
Output_L1 = L1ArenaPtr + 8000;

DMA_Transfer(Weight_L1, Weight_L2);   // load once

for (k = 0; k < N_TILES; k++) {       // tile loop
  DMA_Transfer(Input_L1, Input_L2+k);  // load tile
  DMA_Wait();
  Operator(Input_L1, Weight_L1, Output_L1);
  DMA_Transfer(Output_L2+k, Output_L1);  // store tile
}
```

Backend (c): bare-metal C

- Static L1/L2 arena allocation (MiniMalloc)
- DMA transfer & Tiling loop

Backend: C Code Generation

```
// static L1/L2 arenas (MiniMalloc)
Weight_L1 = L1ArenaPtr + 0;       // resident
Input_L1  = L1ArenaPtr + 5000;
Output_L1 = L1ArenaPtr + 8000;

DMA_Transfer(Weight_L1, Weight_L2);   // load once

for (k = 0; k < N_TILES; k++) {       // tile loop
  DMA_Transfer(Input_L1, Input_L2+k);  // load tile
  DMA_Wait();
  Operator(Input_L1, Weight_L1, Output_L1);
  DMA_Transfer(Output_L2+k, Output_L1);  // store tile
}
```

Backend (c): bare-metal C

- Static L1/L2 arena allocation (MiniMalloc)
- DMA transfer & Tiling loop
- Gradient kernels from PULP-TrainLib (made tiling-compatible) + PULP-NN forward

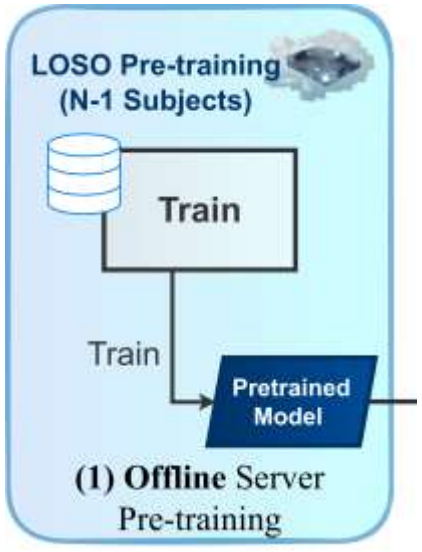
Backend: C Code Generation

```
// static L1/L2 arenas (MiniMalloc)
Weight_L1 = L1ArenaPtr + 0;       // resident
Input_L1  = L1ArenaPtr + 5000;
Output_L1 = L1ArenaPtr + 8000;

DMA_Transfer(Weight_L1, Weight_L2);   // load once

for (k = 0; k < N_TILES; k++) {       // tile loop
  DMA_Transfer(Input_L1, Input_L2+k);  // load tile
  DMA_Wait();
  Operator(Input_L1, Weight_L1, Output_L1);
  DMA_Transfer(Output_L2+k, Output_L1);  // store tile
}
```

Backend (c): bare-metal C

- Static L1/L2 arena allocation (MiniMalloc)
- DMA transfer & Tiling loop
- Gradient kernels from PULP-TrainLib (made tiling-compatible) + PULP-NN forward
- Optimizer step folded in after backpropagation

Evaluation Setup



Scenario A · Day-1

Cold-start: fine-tune the LOSO backbone on 80% of a new subject's first session; test on 20%.

Scenario B · Day-N

Sequential sessions ; fine-tune on first 80% (in time), test on last 20%.

Datasets & target

	EEG [5]	EOG [6]
Backbone	MI-BMINet	EpiDeNet
Params	7.9 k	4.1 k
Subjects	5	5
Sessions	4	2
Task	2-class	11-class
Seq. len T	1900	1000

GAP9 MCU · 9-core RISC-V · 128 kB L1 TCDM · 1.5 MB L2

Results: Adaptation Accuracy

No-FT: No Fine-Tuning
LP: Last Linear Layer Fine-Tuning
Full-FT: Offline Full Model Fine-Tuning
Edge-FT: On Edge Full Model Fine-Tuning

Average accuracy (%)

Task	No-FT	LP	Full-FT	Edge-FT

Results: System Efficiency

0.67 MB
EEG peak L2 memory
vs. 5.36 MB
fits 1.5 MB budget

< 50 mW
training power
43.8 (EEG)
48.2 (EOG)

17 / 85
samples s^{-1}
EEG / EOG full-network BP

211 / 951
sessions / charge
320 mAh
40 ep × 200 samples

Per-batch (8 samples) on GAP9 — FP32, 1.5 MB L2

Dataset	Strategy	Peak L2 (MB)	Latency (ms)	Power (mW)	GFLOP/s
EEG	LP	0.19	360	45.2	0.38
EEG	Full-FT	**5.36 ✗**	—	—	—
EEG	**Edge-FT**	**0.67**	470	43.8	**0.89**
EOG	**Edge-FT**	**0.28**	94	48.2	0.22

✗ Full-FT exceeds GAP9 L2 — cannot run on-chip.

Bottom line
Full Training runs on chip with 8min/1.5min for EEG and EOG

Conclusion

- **First end-to-end compiler** for full-network CNN backpropagation on constrained MCUs — extends Deeploy from inference to training.

Future work: quantized (low-bit) on-device training · additional biosignal modalities · more training new methods

Open source: github.com/pulp-platform/Deeploy

References

[1] M. Scherer et al., “Deeploy: Enabling Energy-Efficient Deployment of Small Language Models on Heterogeneous Microcontrollers,” IEEE TCAD, 2024.

[2] R. Wang, V. J. B. Jung, P. Wiese, F. Conti, A. Burrello, L. Benini, “TrainDeeploy: Hardware-Accelerated Parameter-Efficient Fine-Tuning of Small Transformer Models at the Extreme Edge,” DATE, 2026.

[3] D. Nadalini et al., “PULP-TrainLib: Enabling On-Device Training for RISC-V Multi-core MCUs through Performance-Driven Autotuning,” SAMOS, 2022.

[4] Y. Wu and K. He, “Group Normalization,” ECCV, 2018.

[5] L. Mei et al., “An Ultra-Low Power Wearable BMI System with Continual Learning Capabilities,” IEEE TBioCAS, 2025.

[6] S. Frey et al., “GAPses: Versatile Smart Glasses for Comfortable, Fully-Dry Acquisition and Ultra-Low-Power Processing of EEG and EOG,” IEEE TBioCAS, 2025.

[7] J. Lin et al., “On-Device Training Under 256KB Memory,” NeurIPS, 2022.

[8] M. A. Hamdi et al., “MATCH: Model-Aware TVM-Based Compilation for Heterogeneous Edge Devices,” IEEE TCAD, 2025.

[9] S. Frey et al., "“GAPses: Versatile smart glasses for comfortable and fully-dry acquisition and parallel ultra-low-power processing of EEG and EOG", IEEE TBCS 2025.

BioTrain: Sub-MB, Sub-50mW On-Device Fine-Tuning

Thank you

Questions welcome

runwang@iis.ee.ethz.ch · github.com/pulp-platform/Deeploy

TrainDeeploy, DATE 2026 (TransformerTraining at the ExtremeEdge)

IJCNN TinyML 2026, (Extended Work on CNN and on-board)

RISC-V Europe Submit Demo 2026 (Youtube)

Deeploy

https://github.com/pulp-platform/Deeploy

Onnx4Deeploy

https://github.com/pulp-platform/Onnx4Deeploy

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA