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 (IIS), ETH Zurich, Switzerland
†Department of Electrical, Electronic and Information Engineering (DEI), University of Bologna, Italy
‡Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, Italy
{runwang, jungvi, wiesep, sebastian.frey, gspacone, lbenini}@iis.ee.ethz.ch, f.conti@unibo.it, alessio.burrello@polito.it

***Abstract*—Biosignals exhibit substantial cross-subject and cross-session variability, inducing severe domain shifts that degrade post-deployment performance for small, edge-oriented AI models. On-device adaptation is therefore essential to both preserve user privacy and ensure system reliability. However, existing sub-100 mW Microcontroller Unit (MCU)-based wearables platforms can only support shallow or sparse adaptation schemes due to the prohibitive memory footprint and computational cost of full Backpropagation (BP). In this paper, we propose BioTrain, a framework enabling full-network fine-tuning of state-of-the-art biosignal models under milliwatt-scale power and sub-megabyte memory constraints. We validate BioTrain using both offline and on-device benchmarks on Electroencephalogram (EEG) and Electrooculogram (EOG) datasets, covering Day-1 new-subject calibration and longitudinal adaptation to signal drift. Experimental results show that full-network fine-tuning achieves accuracy improvements of up to 35% over non-adapted baselines and outperforms last-layer update by approximately 7% during new-subject calibration. Furthermore, on the GAP9 MCU platform, BioTrain enables efficient on-device training throughput of 17 samples/s for EEG and 85 samples/s for EOG models with a power envelope below 50 mW. In addition, BioTrain's efficient memory allocator and network topology optimization enable the use of a large batch size, thereby reducing peak memory usage. For a completely on-chip BP on GAP9, BioTrain reduces the memory footprint by 8.1× from 5.4 MB to 0.67 MBcompared to conventional full-network fine-tuning using batch normalization with batch size 8.**

***Index Terms*—TinyML, On-Device Learning, Continual Learning, Biomedical Edge Computing, Personalized Healthcare, EEG, EOG**

I. Introduction

Biosignal-driven edge-AI is rapidly emerging in transformative wearable applications, ranging from Brain-Computer Interface (BCI) to longitudinal health monitoring [1]. Biosignals exhibit substantial inter-subject variability and non-stationarity during long-term use, driven by factors such as fluctuating electrode–skin impedance, perspiration, and sensor displacement [1], [2]. These effects induce distribution shifts between training and deployment, degrading performance in wearable-grade, compact AI models, under both cross-subject transfer and long-term temporal drift. To address this challenge, on-device adaptation enables continuous model updates based on biosignals acquired in situ, allowing personalization under non-stationary sensing conditions [3]. When cloud-trained models are deployed on edge devices for biosignal-based wearables, on-device adaptation typically consists of an initial calibration phase for new users, followed by continuous adaptation to long-term signal drift. By avoiding reliance on remote servers, this step naturally preserves data privacy and supports stable, low-latency operations that do not depend on a remote connection.

However, the practical deployment of on-device training techniques is constrained by limited computational, memory, and energy resources in wearable devices. These constraints make robust deep adaptation difficult and have traditionally forced aggressive memory-reduction strategies, such as sparse updates or last-layer training, also known as Linear Probing (LP). Considering both on-device adaptation phases, restricting training to only partially update the network's weights has been shown to be insufficient to handle substantial signal variations [4]. Enabling deeper on-device Backpropagation (BP) is therefore desirable, but it is fundamentally constrained by the large memory footprint of BP and the complexity of executing its computation under tight resource budgets. In the absence of ecosystem and compiler support for automated memory orchestration, such deployments rely heavily on manual optimization, severely limiting the scalability and robustness of on-device adaptation.

To address these challenges, we propose BioTrain, a compiler framework that automatically generates efficient bare-metal C code to perform on-device training on resource-constrained Microcontroller Unit (MCU) platforms, enabling full-network BP under tight memory budgets. BioTrain is built on Deeploy, a domain-specific compiler originally developed for energy-efficient inference on MCUs [5], and extends its inference-oriented compilation pipeline to support training workloads. In particular, BioTrain leverages Deeploy's static memory allocation and tiling mechanisms and extends them to gradient computations by integrating optimized gradient kernels from PULPTrainLib [6], enabling efficient BP for long time-series workloads. The main contributions of this work are as follows:

- We develop BioTrain, a deployment framework that support BP code generation on resource-constrained devices by integrating and optimizing Convolutional Neural Network (CNN) gradient kernels tuned for tiling from PULPTrainLib.

- We demonstrate two real-world on-device biosignal adaptation scenarios on Electroencephalogram (EEG) [4] and Electrooculogram (EOG) [7] datasets. By modifying the topology of the baseline CNN architectures to support on-chip mini-batch processing, BioTrain achieves up to an 8× peak activation memory reduction, from 5.4 MB to 0.67 MB, with respect to conventional full-network fine-tuning using Batch Normalization (BN) with batch size 8, enabling full-network on-chip BP while maintaining comparable or improved performance under both Day-1 new-user calibration and longitudinal adaptation.
- We evaluate on GAP9 MCU showing up to 35% accuracy improvement over non-adapted baselines and 7% over LP across both Day-1 Calibration and Longitudinal Adaptation scenarios. BioTrain achieves full BP training throughput of 17 samples/s (EEG) and 85 samples/s (EOG) within a 50 mW power envelope. Energy analysis shows that a standard 320 mAh battery supports 211 (EEG) or 951 (EOG) complete training sessions with a training configuration of 40 epochs over 200 samples, enabling practical repeated on-device personalization and adaptation.

BioTrain is released open source at `https://github.com/pulp-platform/Deeploy`.

II. Background and Related Works

A. On-device Adaptation

On-device adaptation has become a key capability for edge biosignal-based AI systems operating in non-stationary conditions, where user behavior, sensor placement, and environmental factors induce distribution shift after deployment. Prior work spans (i) supervised personalization via parameter-efficient fine-tuning (e.g., low-rank adaptation or only updating selected layers) [8], and (ii) inference-time unsupervised/semi-supervised test-time adaptation (TTA) based on self-supervision (e.g., entropy minimization, self-distillation, pseudo-labeling), typically updating only a small subset of parameters to fit edge constraints [9], [10]. Although BP-free or statistics-only updates reduce memory usage, they inherently limit adaptation capacity when task-specific supervision is available, as is common during short calibration sessions on biosignal-based wearable devices. BioTrain therefore targets true gradient-based on-device learning to match offline optimization and generalizing across model architectures without hand-crafted update rules.

B. AI Deployment Compiler Frameworks

Edge deployment differs from general-purpose DNN stacks (e.g., PyTorch/JAX) due to tight on-chip memory and limited runtime support. Commercial and open-source compiler toolchains provide efficient MCU-centric inference (e.g., TFLM, X-CUBE-AI, NXP's eIQ, and compiler ecosystems such as MLIR/TVM for lowering and code generation) [11]. On the academic side, Deeploy [5] and MATCH [12] demonstrate compiler-driven tiling and static memory allocation for memory-constrained inference using optimized operator libraries such as PULP-NN. TrainDeeploy [13] extends Deeploy to adapter-style PEFT of small transformers at the edge, but no existing stack supports end-to-end full-network on-device training. Hence, BioTrain extends Deeploy from inference to on-device adaptation, enabling end-to-end BP for CNNs.

TABLE I: Representative on-device training frameworks.

Work	Algorithm	Depth	BS >1	Acc. (rel.)†	Compiler
TinyOL [14]	Last-layer	LP	✗	↓ 5–10%	✗
MiniLearn [15]	Prune + FT	Sparse	✓	↓ 2–10%	✗
TTE [16]	Sparse BP	Sparse	✗	≈	✗
TinyTTA [10]	Early-exit TTA	Shallow	✗	N/A	✗
AIfES [17]	Full BP	Full	✓	≈*	≈‡
BioTrain	Full BP	Full	✓	≈	✓

† Accuracy relative to offline full-network backpropagation training.
‡ Supports on-device training but is currently limited to very small CNN and DNN models due to the lack of tiling and memory orchestration.
* Only tested on simple MNIST and IRIS datasets.

C. On-device Training Frameworks

Early research on on-device learning primarily addressed memory constraints by avoiding full network adaptation. To reduce activation and gradient storage, these works restricted training to shallow or sparse configurations [10], [14]–[16]. As shown in Table I, TinyOL [14] updates only the final classification layer, while MiniLearn [15] and the Tiny Training Engine (TTE) [16] rely on pruning and sparse BP to reduce memory footprints. Similarly, TinyTTA [10] employs memory-aware early-exit ensembles for lightweight adaptation. To further limit activation storage, these approaches typically operate with a batch size of one. However, this comes at the cost of a reduced achieved final accuracy.

Crucially, these design choices are not driven by computational infeasibility, as full backpropagation is viable for a wide range of small-scale networks on edge devices [4]. Instead, the fundamental limitation lies in memory management during training. Supporting full backpropagation on embedded platforms requires storing intermediate activations and gradients that frequently exceed scratchpad memory capacity, making compiler-driven tiling and static memory allocation essential. The only approach that partially supports full BP with an automatic compiler infrastructure to produce end-to-end code is AIfES [17]. However, it lacks systematic mechanisms for tiling and memory orchestration, effectively restricting deployments to small networks that fit entirely within on-chip memory. BioTrain addresses all these limitations by building on Deeploy's [5] compiler-supported tiling and static memory allocation, and by combining gradient accumulation with Group Normalization (GN) [18] to enable batch sizes larger than one, thereby improving the robustness of edge learning algorithms under tight memory budgets and maintaining the same accuracy of offline training.

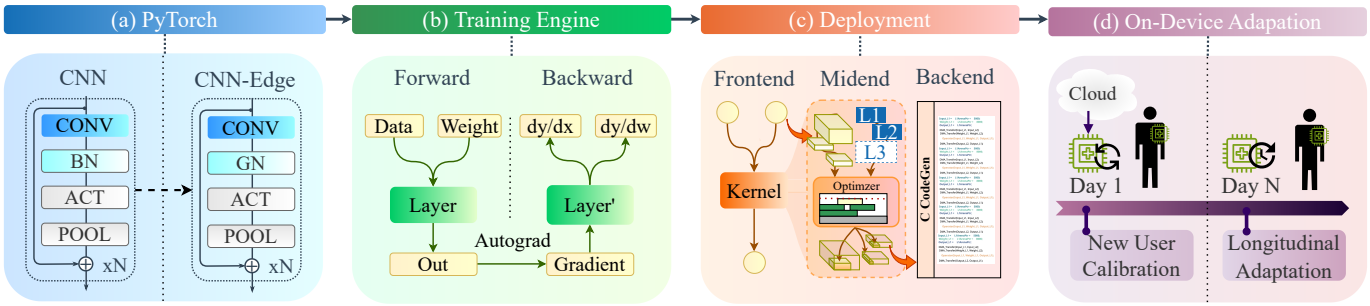

Fig. 1: Overview of the BioTrain framework for on-device model adaptation. (a) The framework interfaces with PyTorch to support common neural network architectures. (b) The training engine executes forward and backward passes, exposing intermediate activations and gradients through an autograd mechanism to enable parameter updates. (c) The deployment stack lowers training operators through a frontend–midend–backend compiler pipeline, generating optimized kernels and bare-metal C code with explicit memory management. (d) In the target on-device learning scenario, a cloud-pretrained model is first initialized and calibrated for a new user, followed by continual on-device adaptation over time.

III. BioTrain Framework

A. Compilation Workflow

Figure 1 summarizes our workflow: in stage (a), we train the model in PyTorch and replace BN with GN to avoid cross-sample dependencies and maintain stable optimization under small effective batch sizes. Combined with gradient accumulation, this reduces the peak gradient/activation memory footprint and enables full end-to-end training within MCU memory budgets. In stage (b), we export the trained model via an enhanced ONNX Runtime training API that synthesizes missing gradient subgraphs for unsupported operators, yielding a complete training graph with end-to-end BP. In stage (c), we compile this ONNX training graph to a target executable by extending Deeploy to support BP, generating code for forward and backward passes, and orchestrating memory allocation. Finally, stage (d) covers in-field operation: the device performs a Day-1 subject calibration upon first use and then runs periodic on-device fine-tuning to preserve accuracy under longitudinal and temporal distribution shifts in the biosignals.

B. On-Chip Memory Reduction for CNN Training

The first challenge addressed is reducing the peak memory footprint of end-to-end training. In biosignal workloads, typical mini-batches range from 8 to 32. These lead to a proportional increase in activation memory compared to inference, making on-chip training prohibitive under tight memory budgets. While reducing the batch size minimizes the memory usage, it often slows convergence and destabilizes optimization, leading to lower accuracy. To address this, we employ gradient accumulation, which executes multiple single-sample forward/backward passes and applies weight updates only after reaching the target effective batch size, thereby retaining the optimization benefits of larger batches with a low peak memory footprint. However, from a compiler perspective, BN requires cross-sample reductions to compute batch-level statistics, introducing global synchronization that is incompatible with gradient accumulation. We therefore replace BN with GN, since it has been shown to provide stable optimization when batch sizes are small [18], and it depends only on per-sample statistics, avoiding cross-sample synchronization. This enables full-network fine-tuning with gradient accumulation under strict on-chip memory constraints.

C. Deployment Stack for End-to-End On-MCU Training

The second challenge addressed is the development of a deployment stack that produces a final, runnable executable for the target MCU, encompassing the complete end-to-end training pipeline in FP32. To this end, we extend Deeploy with three main contributions, summarized in Fig. 2.

1) Frontend: We enhance the frontend to support CNN BP operators, including GN. We further introduce two frontend passes: (1) gradient decomposition, which separates parameterized operator gradients into input and weight gradients (i.e., gradient-x and gradient-w) to accommodate different tiling constraints; and (2) normalization decomposition, which splits normalization operators into statistic computation (i.e., mean and standard deviation) and normalization computation, for handling long biosignals that cannot be fully reduced in L1 memory.

2) Midend: We extend Deeploy's midend to support training graphs, including BP and weight-update subgraphs. Deeploy's midend is responsible for tiling and memory allocation across all nodes of the input graph. Given kernel-specific tiling constraints, it uses OR-Tools to solve a constrained optimization problem that determines a scratchpad-aware tiling configuration with high on-chip reuse and minimal off-chip

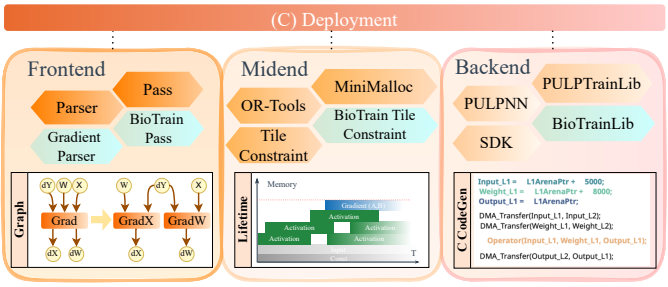

Fig. 2: Overview of the deployment stack for end-to-end on-MCU training. In light blue, our novel inserted blocks.

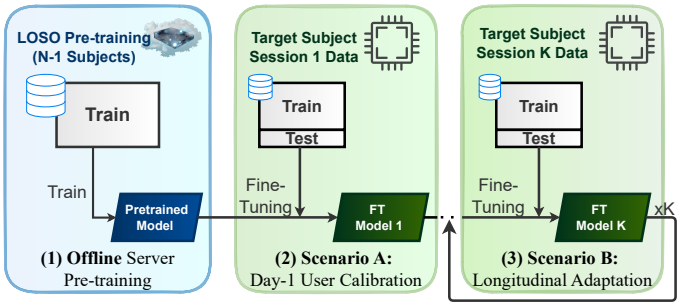

Fig. 3: Overview of the evaluation protocols. The pipeline includes: (1) Offline Server Pre-training using a Leave-One-Subject-Out (LOSO) scheme ($N-1$ subjects); (2) **Scenario A**: Day-1 Calibration. (3) **Scenario B**: Longitudinal Adaptation.

transfers. The framework generalizes this mechanism to end-to-end training by introducing tiling constraints tailored to CNN gradient kernels. Since biosignal workloads typically feature long temporal dimensions and relatively few channels, we adopt temporal tiling as the primary method for orchestrating data movement and reuse within the scratchpad.

For convolution weight gradients, we denote by T_k the temporal tile processed at step k. Within each tile, an `im2col` transformation is applied locally and computes a partial contribution to the weight gradient, $\nabla W^{(k)} = \mathrm{im2col}(X_{T_k})^\top \cdot dY_{T_k}$, which is accumulated into a persistent weight-gradient buffer to form the final ∇W across all tiles. For convolution input gradients, dX is tiled temporally, and the corresponding dY tiles are extended with a kernel-dependent halo, ensuring non-overlapping dX tiles and efficient DMA-compute pipelining. We manage normalization gradients by employing a cross-tile Welford aggregation scheme that computes global mean and variance from tile-local statistics, avoiding the need to retain full-length activations in scratchpad memory. Other operators, including pooling and activation-gradient kernels, follow Deeploy's default tiling strategies.

3) Backend: We extend the Deeploy backend to support CNN gradient kernels, with a new library that includes basic convolution gradient operators adapted from PULP-TrainLib [6], as well as implemented normalization, pooling, and activation gradient kernels. As PULPTrainLib does not natively support tiling, we introduce modifications to ensure compatibility between its gradient kernels and Deeploy's tiling mechanism. For the forward path, we primarily reuse existing optimized kernels from the Deeploy PULP platform backend and the PULP-NN library.

D. Evaluation Scenarios

To systematically evaluate the robustness of fine-tuning methods on edge devices, we provide infrastructure for two scenarios, as illustrated in Fig. 3.

Scenario A: Day-1 Calibration. This protocol simulates initial system deployment to address the cold-start problem for unseen users. We assume the existance of a backbone model obtained offline via Global Leave-One-Subject-Out (LOSO) Pre-training on $N-1$ subjects. For the target unseen subject, we simulate an initial deployment by fine-tuning the global model on the training set of the subject's first session (80% of session S_1) and evaluating on the test set (20% of session S_1). This corresponds to an initial calibration session on the subject where ground-truth labels are obtained using a reference setup or supervised protocol (e.g., clinical devices or controlled tasks) while the wearable device collects data.

TABLE II: Summary of the EEG and EOG datasets and models.

Property	EEG Dataset	EOG Dataset
Subjects	5	5
Sessions per subject	4	2
Signal channels	8	3 ($L/R/C$)
Classification task	2 ($Tongue$ vs. $Rest$)	11
Trial duration	3.8 s ($T = 1900$)	2.0 s ($T = 1000$)
Backbone model	MI-BMINet	EpiDeNet
Model parameters	7.9 k	4.1 k

Scenario B: Longitudinal Adaptation. We then evaluate a longitudinal adaptation protocol through sequential incremental learning. In this scenario, the model calibrated on the target subject's session S_1 (as described in Scenario A) serves as the starting point for continuous evolution. As we collect new data from the user across subsequent sessions ($S_2, S_3, \ldots, S_n$), the model continuously refines its tracking of physiological shifts. At each session, we fine-tune on the first 80% of the data (in temporal order) and test on the remaining 20%. This simulates continuous refinement to track physiological drift. In a real-world scenario, this corresponds to periodic re-calibration sessions where ground-truth measurements are again collected using reference equipment and supervised procedures to update the on-device model.

IV. Experimental Setup

A. Hardware, Biosignal Datasets, and Models

The hardware evaluation is conducted on the GAP9 MCU platform, which integrates a programmable 9-core RISC-V compute cluster and an additional core functioning as the system controller. GAP9 features a shared 128 kB user-managed L1 SRAM and 1.5 MB L2 SRAM. Data transfers between L2 and L1 are performed via a programmable DMA engine, enabling overlap between computation and memory transfers. Power consumption is measured using the Nordic Semiconductor's Power Profiler Kit II.

We evaluate BioTrain across two biosignal modalities, EEG and EOG, using data collected and models deployed via the GAPses [7] platform, which is already built on the GAP9 architecture. The dataset characteristics and model configurations are summarized in Table II. For the EEG modality, we use Dataset B from [4], which consists of recordings from 5 subjects, each collected over 4 sessions. The task is a binary classification between tongue motor movement and rest. Signals are acquired from 8 EEG channels at a sampling rate of 500 Hz, with each trial lasting 3.8 s ($T = 1900$ samples). The preprocessing pipeline applies a 50 Hz notch filter followed by a 0.5–100 Hz bandpass filter. We adopt MI-BMINet as the

TABLE III: Average classification accuracy (%) under Day-1 Calibration and Longitudinal Adaptation.

Task	Avg. Acc. (%, Mean ± Std.)			
	No-FT	LP	Full-FT	Edge-FT
EEG (Subj.)	50.8 ± 8.8	74.8 ± 11.4	80.0 ± 11.4	**86.4** ± 7.9
EEG (Sess.)	49.8 ± 3.8	76.2 ± 7.6	78.7 ± 7.3	**83.9** ± 13.3
EOG (Subj.)	78.1 ± 18.1	83.3 ± 13.0	**88.7** ± 10.3	87.7 ± 10.4
EOG (Sess.)	84.1 ± 11.2	86.5 ± 10.7	**89.1** ± 8.8	87.2 ± 7.6

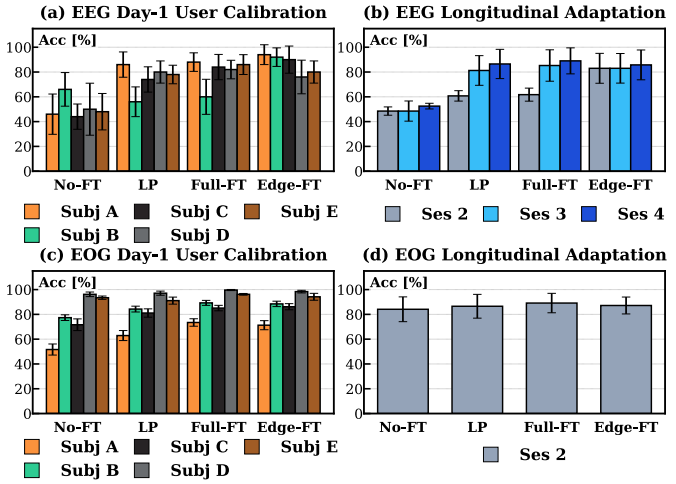

Fig. 4: Comparison of fine-tuning strategies across Day-1 Calibration (a, c) and Longitudinal Adaptation (b, d) for EEG (top) and EOG (bottom). Edge-Optimized Fine-Tuning (Edge-FT) replaces BN with GN to enable memory-efficient on-device training.

classification backbone, following the original EEG study that introduced the dataset. MI-BMINet is a lightweight CNN architecture tailored for EEG classification and employs spatial–temporal convolutions and depthwise separable convolutions to minimize computational and memory overhead. The model contains 7.9 k parameters, making it well-suited for on-chip training under tight memory constraints.

For the EOG modality, we use data collected from the GAPses smart glasses platform [7], comprising recordings from 5 subjects across 2 sessions. Signals are acquired using 3 dry electrodes positioned on the left and right nose pads and the nasal bridge (center), while bias and reference electrodes are placed on the right and left mastoids, respectively. From these channel measurements, two logical EOG channels, horizontal and vertical, are derived as $V_H = V_R - V_L$ and $V_V = V_C - \frac{V_R+V_L}{2}$ following the standard formulation in the original study. Signals are sampled at 500 Hz, with each trial lasting 2.0 s ($T = 1000$ samples). The preprocessing pipeline includes a 50 Hz notch filter, a 0.5–40 Hz bandpass filter, and a moving average filter with a 2 s window applied for real-time de-trending, consistent with the original GAPses processing pipeline. The task is an 11-class eye-movement classification problem. Following [7], we use EpiDeNet as the classification backbone, a compact CNN with 4.1 k parameters.

B. Evaluation Protocols

For both modalities (EEG and EOG) under both scenarios (Day-1 Calibration and Longitudinal Adaptation), we evaluate four adaptation strategies: Zero-Shot / No Fine-Tuning (No-FT), LP (only training the last classification layer), Full Backpropagation Fine-Tuning (Full-FT) (using the original architectures with BN layers), and the proposed edge-optimized Edge-FT, which replaces BN with GN to improve training stability under small-batch, on-device settings. For cloud pre-training, we use the Adaptive Moment Estimation with Weight Decay (AdamW) optimizer with a batch size of 64, a learning rate of 1×10^{-3}, and 40 training epochs. To accommodate the transition from cloud-scale resources to edge-constrained environments, on-device adaptation is performed using a more memory-efficient optimization setup. Specifically, we switch to Stochastic Gradient Descent (SGD) with momentum (0.9) and cosine annealing, using a reduced batch size of 8 and a learning rate of 5×10^{-3}, with weight decay set to 1×10^{-3}. The number of fine-tuning epochs is set to 30. All experiments are repeated five times with different random seeds, and results are reported as mean and standard deviation.

V. Experimental Results

A. Adaptation Accuracy

Figure 4 and Table III summarize the accuracy of the described protocols under the two introduced scenarios for both benchmarks.

Scenario A: Day-1 Calibration. Without adaptation, No-FT yields low accuracy for EEG (50.8%) and high variance for EOG (78.1% ± 18.1%), confirming that pre-trained models cannot generalize to new users. LP improves accuracy to 74.8% (EEG) and 83.3% (EOG), yet exhibits large cross-subject variability, with EEG Subject B saturating at 56.0%, indicating classifier-only updates are insufficient to cope with distribution shifts. Full-FT further improves to 80.0% (EEG) and 88.7% (EOG), demonstrating the benefit of updating the full network. Edge-FT achieves comparable or better performance, 86.4% for EEG (+6.4% over Full-FT) and 87.7% for EOG while reducing variance, showing that replacing BN with GN does not compromise adaptation quality.

Scenario B: Longitudinal Adaptation. Without adaptation, No-FT degrades to 49.8% for EEG and remains at 84.1% for EOG, confirming persistent signal drift across sessions. LP shows adaptation lag for EEG, with accuracy dropping to 62.4% at session S_2 before gradually recovering; such delayed convergence is impractical for wearables. For EOG, LP achieves 86.5%, offering only marginal improvement. Full-FT exhibits similar lag for EEG (61.6% at S_2) due to fragile representations from weak Day-1 initialization, but achieves the best EOG accuracy (89.1%), demonstrating its effectiveness when initial calibration is strong. Edge-FT maintains stable EEG performance from S_2 onward (84.8% at S_2, 85.8% at S_4), avoiding the adaptation lag observed with other methods. For EOG, Edge-FT achieves 87.2%, comparable to Full-FT. These results confirm that full-network fine-tuning

TABLE IV: System-level evaluation of on-device training on GAP9 (FP32, 1.5 MB L2), comparing LP, Full-FT, and Edge-FT.

Dataset	Strategy	Acc (Subj/Sess)	Params (Train)	FLOPs	L2 Peak Memory (MB)	Latency (ms)	Power (mW)	Energy (mJ)	Throughput (GFLOPs/s)
EEG [4]	LP	74.8 / 76.2	0.07 k	139.2 M	0.19	360.0	45.2	16.08	0.38
	Full-FT	80.0 / 78.7	7.9 k	416.8 M	5.36†	—	—	—	—
	Edge-FT	**86.4 / 83.9**	7.9 k	416.8 M	0.67	469.6	43.8	20.16	0.89
EOG [7]	LP	83.3 / 86.5	0.2 k	7.2 M	0.11	34.4	50.4	1.76	0.21
	Full-FT	**88.7 / 89.1**	4.1 k	20.8 M	2.24†	—	—	—	—
	Edge-FT	87.7 / 87.2	4.1 k	20.8 M	0.28	93.6	48.2	4.48	0.22

† Exceeds GAP9 L2 capacity (1.5 MB), thus cannot execute fully on-chip.

tracks effectively signal drift, with Edge-FT providing the most consistent longitudinal adaptation across both modalities. In both scenarios and across both benchmarks, we have demonstrated that our Edge-FT achieves comparable or better accuracy to Full-FT, while outperforming the lighter LP.

B. Memory, Performance, Energy Analysis

Table IV reports per-batch (8 samples) system-level performance on GAP9. For EEG, Edge-FT scales trainable parameters from 0.07 k (LP) to 7.9 k and per-batch compute from 139.2 M to 416.8 M FLOPs, yet caps peak L2 activation memory at 0.67 MB—enabling full-network BP entirely on-chip. A conventional full-FT with standard batch normalization would require ∼5.4 MB, exceeding GAP9's 1.5 MB L2. BioTrain sustains 0.89 GFLOPs/s under 50 mW; EOG shows a similar trend (0.28 MB, 0.22 GFLOPs/s).

With 40 epochs over 200 samples (8000 samples/session) on a 320 mAh, 3.7 V battery, BioTrain affords ∼211 EEG and ∼951 EOG on-device training sessions, demonstrating practical full-network fine-tuning within wearable-class MCU energy and memory budgets.

VI. Conclusion

We presented BioTrain, a compiler-assisted on-device adaptation framework enabling full-network fine-tuning within a sub-50 mW envelope on MCU platforms. On EEG and EOG datasets it delivers up to 35% accuracy gains over non-adapted baselines with 8× peak memory reduction (5.4 MB→0.67 MB), enabling full on-chip BP across Day-1 Calibration and Longitudinal Adaptation scenarios. Future work will extend BioTrain to quantized training and additional biosignal modalities.

Acknowledgment

This work has received funding from the Swiss State Secretariat for Education, Research, and Innovation (SERI) under the SwissChips initiative.

References

[1] S. Li, Z. Tang, M. Li, L. Yang, and Z. Shang, "A survey of neural signal decoding based on domain adaptation," *Neurocomputing*, vol. 657, p. 131653, Dec. 2025.

[2] B. Yang, F. Rong, Y. Xie, D. Li, J. Zhang, F. Li, G. Shi, and X. Gao, "A multi-day and high-quality EEG dataset for motor imagery brain-computer interface," *Scientific Data*, vol. 12, no. 1, p. 488, Mar. 2025.

[3] S. Zhu, T. Voigt, F. Rahimian, and J. Ko, "On-device training: A first overview on existing systems," *ACM Trans. Sen. Netw.*, vol. 20, no. 6, pp. 118:1–118:39, Oct. 2024.

[4] L. Mei, T. M. Ingolfsson, C. Cioflan, V. Kartsch, A. Cossettini, X. Wang, and L. Benini, "An ultra-low power wearable BMI system with continual learning capabilities," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 19, no. 3, pp. 511–522, Jun. 2025.

[5] M. Scherer, L. Macan, V. Jung, P. Wiese, L. Bompani, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling energy-efficient deployment of small language models on heterogeneous microcontrollers," Aug. 2024.

[6] 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," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, A. Orailoglu, M. Reichenbach, and M. Jung, Eds. Cham: Springer International Publishing, 2022, vol. 13511, pp. 200–216.

[7] S. Frey, M. A. Lucchini, V. Kartsch, T. M. Ingolfsson, A. H. Bernardi, M. Segessenmann, J. Osieleniec, S. Benatti, L. Benini, and A. Cossettini, "GAPses: Versatile smart glasses for comfortable and fully-dry acquisition and parallel ultra-low-power processing of EEG and EOG," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 19, no. 3, pp. 616–628, Jun. 2025.

[8] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[9] J. Song, J. Lee, I. S. Kweon, and S. Choi, "EcoTTA: Memory-efficient continual test-time adaptation via self-distilled regularization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 11 920–11 929.

[10] H. Jia, Y. Kwon, A. Orsino, T. Dang, D. Talia, and C. Mascolo, "TinyTTA: Efficient test-time adaptation via early-exit ensembles on edge devices," *Advances in Neural Information Processing Systems*, vol. 37, pp. 43 274–43 299, 2024.

[11] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han, "Tiny machine learning: Progress and futures," *IEEE Circuits and Systems Magazine*, vol. 23, no. 3, pp. 8–34, 2023.

[12] M. Amine Hamdi, F. Daghero, G. Maria Sarda, J. Van Delm, A. Symons, L. Benini, M. Verhelst, D. Jahier Pagliari, and A. Burrello, "MATCH: Model-aware TVM-based compilation for heterogeneous edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 10, pp. 3844–3857, Oct. 2025.

[13] R. Wang, V. J. B. Jung, P. Wiese, F. Conti, A. Burrello, and L. Benini, "TrainDeeploy: Hardware-accelerated parameter-efficient fine-tuning of small transformer models at the extreme edge," in *Design, Automation and Test in Europe Conference (DATE)*, 2026.

[14] H. Ren, D. Anicic, and T. A. Runkler, "TinyOL: TinyML with online-learning on microcontrollers," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[15] C. Profentzas, M. Almgren, and O. Landsiedel, "MiniLearn: On-device learning for low-power IoT devices," in *EWSN*, 2022, pp. 1–11.

[16] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 941–22 954, 2022.

[17] L. Wulfert, J. Kühnel, L. Krupp, J. Viga, C. Wiede, P. Gembaczka, and A. Grabmaier, "AIfES: A next-generation edge AI framework," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 6, pp. 4519–4533, 2024.

[18] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–19.