TrainDeeploy: Hardware-Accelerated Parameter-Efficient Fine-Tuning of Small Transformer Models at the Extreme Edge

Run Wang*, Victor J.B. Jung*, Philip Wiese*, 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, lbenini}@iis.ee.ethz.ch, f.conti@unibo.it, alessio.burrello@polito.it

***Abstract*—On-device tuning of deep neural networks enables long-term adaptation at the edge, while keeping data fully private and secure. However, the high computational demand of backpropagation remains a challenge for ultra-low-power, memory-constrained extreme-edge devices. Attention-based models further exacerbate this challenge, given their complex architecture and scale. We present *TrainDeeploy*, a novel framework that unifies efficient inference with on-device training on heterogeneous ultra-low-power System-on-Chips (SoCs). TrainDeeploy is the first complete on-device training pipeline for extreme edge SoCs supporting both Convolutional Neural Networks (CNNs) and Transformer models, as well as multiple training techniques, such as selective layer-wise fine-tuning and Low-Rank Adaptation (LoRA). On a RISC-V-based heterogeneous SoC, we demonstrate the first end-to-end fine-tuning of a complete Transformer, Compact Convolutional Transformer (CCT), achieving 11 trained images per second. We show that LoRA on-device leads to a 23% reduction in dynamic memory usage, $15\times$ reduction in trainable parameters and gradients, and $1.6\times$ reduction in memory transfer compared to full backpropagation. TrainDeeploy achieves up to 4.6 FLOP/cycle on CCT (0.28M Param, 71–126M FLOPs) and leading-edge performance up to 13.4 FLOP/cycle on Deep-AE (0.27M Param, 0.8M FLOPs), while simultaneously widening the scope compared to state-of-the-art frameworks to support both CNNs and Transformers with parameter-efficient tuning.**

***Index Terms*—On-device Training, Edge AI, Hardware Acceleration, LoRA, Heterogeneous Platforms**

I. Introduction

Edge computing has emerged as a key enabler of intelligent Internet of Things systems, enabling data processing and decision-making directly on end devices [1]. With the rapid integration of Artificial Intelligence (AI), this paradigm has extended from mobile devices to Microcontroller Unit (MCU)-based System-on-Chips (SoCs) powering sensors and wearables, bringing deep learning capabilities to ultra-low-power hardware [2]. Considerable progress has been made in deploying Convolutional Neural Networks (CNNs) and Transformers on such platforms, demonstrating that even highly resource-constrained devices can support efficient inference [3]. However, the potential of AI at the edge extends beyond inference. Growing demand for personalization, privacy, and continual adaptation motivates on-device training [4], which typically relies on backpropagation to fine-tune pretrained models locally without reliance on the cloud. Alternative approaches, such as gradient-free [5], [6] or meta-learning [7] methods, have been explored, but they lack the generality of backpropagation and, in many cases, cannot achieve competitive accuracy results.

On the other hand, training CNNs and Transformer models with backpropagation is highly demanding in terms of both computation and memory, particularly when facing the hardware constraints of extreme edge devices. Even compact networks typically require 10^7–10^9 floating-point operations per training step, as the workload is dominated by large General Matrix Multiplication (GEMM) operations [8]. Moreover, storing activations for gradient computation requires more than 10^7 bytes of memory [1], [8], which often exceeds the capacity of embedded platforms.

Current embedded machine learning training frameworks address these challenges via compute-focused optimization, parameter-efficient methods, or lightweight CNN-centric designs, but each has clear limitations. For example, PULP-TrainLib [9] primarily targets compute performance optimization; however, it does not provide an end-to-end flow with operator tiling and memory allocation across multiple hierarchies, leading to inefficient memory usage. Pruning and sparse training techniques such as MiniLearn [10], TTE [8], Sparse Backpropogation [11] reduce the memory demand, but they remain largely CNN-centric and are tailored to single-core MCUs.

To address these challenges, we introduce *TrainDeeploy*, a compilation and execution flow that enables Transformer training on heterogeneous ultra-low-power SoCs. Building on Deeploy [3], which was created for energy-efficient inference on heterogeneous MCUs, TrainDeeploy extends the flow with automatic differentiation and training-oriented passes, making backpropagation feasible on heterogeneous resource-constrained platforms. We demonstrate effective on-device Transformer fine-tuning on a heterogeneous SoC with a GEMM accelerator, using Low-Rank Adaptation (LoRA) [12] to reduce the memory footprint. This establishes the first unified pipeline for both inference and training of Transformers on ultra-low-power heterogeneous SoCs.

Specifically, our contributions are summarized as follows:

- We present TrainDeeploy, a compilation and execution flow that enables Transformer training on heterogeneous ultra-low-power SoCs, leveraging on-chip accelerators to

978-3-9826741-1-7/DATE26/© 2026 EDAA

speed up training.

- We demonstrate the first complete end-to-end on-device fine-tuning of a Compact Convolutional Transformer (CCT) on a heterogeneous SoC platform [13].
- We implement on-device LoRA training as a technique to reduce on-device training workload, making it feasible on low-power extreme-edge devices.

Our results, targeting an extreme-edge RISC-V-based heterogeneous SoC with on-chip GEMM acceleration, show that LoRA acceleration yields 23% lower dynamic memory usage, 15× fewer trainable parameters and gradients, 1.6× fewer memory transfers, and speeds up the training by 2.3-3.5x compared to non-accelerated execution, while also outperforming state-of-the-art CNN training frameworks. We show that end-to-end CCT fine-tuning can be performed at a throughput of up to 11 gradient updates per second in a single-sample training setting, fine-tuning all transformer layers. To the best of our knowledge, TrainDeeploy is the first end-to-end on-device fine-tuning deployment framework targeting both CNNs and Transformers.

II. Background

A. On-Device Training

On-device training imposes substantially higher demands compared with inference in terms of computational complexity, memory footprint, and numerical precision. Beyond forward propagation, training requires backpropagation to compute gradients for both inputs and weights. The most important operation is the GEMM, given its massive utilization in attention layers, Multi-Layer Perceptrons (MLPs), and CNNs. For a generic GEMM $Y = WX$, the backward pass yields $\frac{\partial L}{\partial X} = W^\top \Delta$ and $\frac{\partial L}{\partial W} = \Delta X^\top$, where $\Delta = \frac{\partial L}{\partial Y}$ is the upstream gradient. Each forward GEMM, therefore, induces two additional GEMMs in the backward pass. In addition to computation, intermediate activations must be stored until the backward pass, significantly stressing the few hundred KB to a few MB of SRAM available in typical MCU-class devices. Furthermore, while inference commonly tolerates low-precision integer quantization [1], training generally requires higher-precision floating point (FP) arithmetic (e.g., FP16, FP32) to ensure stable gradient updates [8].

B. Parameter-Efficient Fine-Tuning and LoRA

Parameter-Efficient Fine-Tuning (PEFT) techniques, which update only a small subset of model parameters while retaining accuracy and adaptability, have been introduced to cope with the prohibitive compute and memory requirements of billion-parameter large language models. Despite the orders-of-magnitude difference in scale, such techniques are also highly relevant but remain largely unexplored in the context of MCU-scale, edge-oriented platforms. Among these methods, LoRA is particularly effective [12]. Instead of updating the full weight $W_0 \in \mathbb{R}^{d\times k}$, LoRA introduces a low-rank decomposition $W = W_0 + BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and $r \ll \min(d, k)$. As shown in Figure 1(a), W_0 is frozen and only the two small matrices A and B are trained, reducing the number of trainable parameters from dk to $r(d + k)$. Figure 1(b) illustrates how

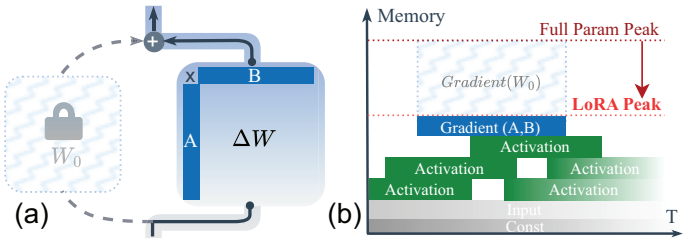

Fig. 1: Low-Rank Adaptation (LoRA). (a) Frozen pre-trained weight W_0 with trainable low-rank matrices A and B. (b) Comparison of memory footprint over time between full-parameter fine-tuning and LoRA. The stacked areas illustrate how tensors are allocated and released during execution. By reducing gradient storage, LoRA further lowers the peak memory footprint, enabling training within tight on-chip memory budgets.

this translates into a substantial reduction in memory footprint compared to full-parameter fine-tuning, with activation and weight memory remaining unchanged while gradient storage is drastically reduced. Also, since gradients are computed only for A and B, LoRA leaves the activation-gradient computation unchanged but significantly reduces the weight-gradient and parameter-update costs.

C. Heterogeneous HW Platforms

A recent trend in the extreme-edge AI domain is the proliferation of heterogeneous SoCs equipped with specialized accelerators. Recent platforms increasingly combine a general-purpose host with neural processing units (NPUs), digital signal processors (DSPs), or fixed-function GEMM engines. For example, the STM32N6 family from STMicroelectronics integrates the Arm Cortex-M55 core with the in-house Neural-ART accelerator [14]. The MAX78000 from Analog Devices features an Arm Cortex-M4 control core coupled with a dedicated CNN accelerator [15]. The GAP9 from GreenWaves Technologies consists of a RISC-V fabric controller and a compute cluster with nine RISC-V cores plus the NE16 deep neural network accelerator [16]. Finally, Arm Ethos-U55 NPUs are integrated in commercial edge SoCs such as the Alif Ensemble family, Infineon PSoC Edge, and Himax WiseEye2 [17]. This heterogeneous organization enables collaborative execution: control and irregular tasks remain on the MCU core, while specific compute-intensive tasks are offloaded to accelerators.

III. Related Work

Prior work on on-device training for extreme-edge platforms has explored four complementary directions. Model-side methods (e.g., TinyTL [18], MiniLearn [10]) reduce memory by pruning or restricting updates, but sacrifice adaptability and remain CNN-centric. Compiler-level techniques (e.g., TTE [8], POET [19]) mitigate memory pressure via operator reordering, sparsity, quantization, paging, or gradient checkpointing, but require nontrivial graph modifications and some tricks incur latency–memory trade-offs. System-level frameworks such as PULP-TrainLib [9] offer optimized computational primitives for on-device RISC-V learning, but mainly demonstrate benefits on small networks and lack memory-aware compilation support. Hardware design for extreme edge training (e.g., MINOTAUR [20], Chameleon [7]) achieves energy-efficiency

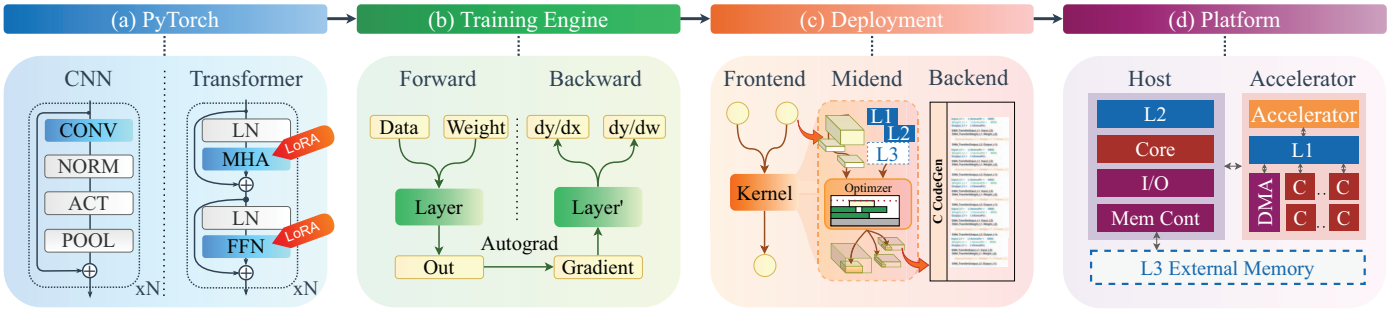

Fig. 2: Overview of the TrainDeeploy framework. (a) Models are defined in PyTorch and exported through Open Neural Network Exchange (ONNX). (b) The training engine augments the forward graph with the backward graph via autograd, producing a full training graph. (c) Deeploy extends its inference compiler to training with a frontend (ONNX parsing), a midend (memory optimizer integrating tiling and static allocation across full forward-backward graph), and a backend (C code generation). (d) The generated code is deployed on the heterogeneous SoCleveraging hierarchical memory mapping (Level-1 Tightly-Coupled Data Memory (TCDM) (L1)–L3 External Memory (e.g., HyperRAM)) and an on-board accelerator for GEMMs.

via novel formats or gradient-free schemes, but limits generality and portability. Overall, these approaches often *(i)* compromise accuracy, *(ii)* optimize latency or memory in isolation, *(iii)* remain CNN-centric, or *(iv)* depend on specialized formats and hardware. To the best of our knowledge, no work has demonstrated an end-to-end Transformer/CNN training pipeline within the energy and memory budgets of ultra-low-power devices, one of the core contributions of this work. Furthermore, none of these tools support the efficient deployment on highly heterogeneous devices, as the ones described in Sec. II-C. TrainDeeploy is the first end-to-end Transformer/CNN training pipeline for heterogeneous ultra-low-power devices.

IV. TrainDeeploy Framework

Starting from PyTorch models and training strategies, TrainDeeploy performs training graph construction, operator tiling, and static memory allocation at compile time and lowers the graph into C code, which can then be compiled using a regular platform compiler (e.g., GCC, LLVM). To realize this, we build on Deeploy [3], a domain-specific compiler aimed at energy-efficient inference of DNNs on heterogeneous MCUs. TrainDeeploy exploits and extends Deeploy's static memory allocation and tiling principles, broadening the flow from inference to training. We also designed our framework to take advantage of accelerators that are increasingly available in emerging edge devices. Within this context, a central contribution is the joint handling of memory and compute bottlenecks: LoRA reduces trainable state and gradient pressure to fit the HW memory, while we offload GEMM workloads to on-chip accelerators to speed up training workloads.

a) Target platform: As Figure 2(d) illustrates, the target platform we consider is a standard heterogeneous SoC organization, consisting of a host processor that manages peripherals and orchestrates execution, and heterogeneous programmable accelerators. Specifically, we adopt a generic accelerator paradigm that might contain one or more processor cores, a local L1 memory, and potentially a fixed-function hardware unit (e.g., to accelerate GEMM). The memory hierarchy consists of on-chip memories, i.e., an accelerator-dedicated L1, a host L2, and an external L3, with data movement managed with DMA.

b) Pipeline: The pipeline starts with the input models and training strategies, which are defined in PyTorch (Figure 2(a)). While our framework supports general architectures such as CNNs and Transformers, we highlight that Transformer networks can optionally be augmented with PEFT modules such as LoRA. More broadly, TrainDeeploy can integrate other model-level training optimizations (e.g., structured sparsity, layer-wise training, or alternative PEFT methods). The training configuration, loss functions, and optimizers are specified, and the models are then exported in ONNX format to provide a hardware-agnostic graph representation that serves for subsequent symbolic differentiation.

Figure 2(b) illustrates the construction of a full training graph from its forward counterpart. Using an automatic differentiation engine [21], the forward graph is traversed in reverse topological order starting from the loss node. For each operator, a predefined differentiation rule specifies the corresponding backward operator, which is instantiated and linked to form explicit gradient paths. The resulting structure integrates the forward and backward dataflows into an operator-level automatic differentiation graph. Subsequently, optimizer update rules are incorporated as dedicated subgraphs, yielding a complete training representation. The final outcome is a static ONNX training graph that jointly encodes inference and gradient flows in a hardware-agnostic intermediate form. Unlike dynamic autograd execution, the static representation exposes the complete forward–backward structure of the training step. This global view enables the subsequent memory optimizer to operate over the entire computation, thereby enlarging the search space.

Figure 2(c) shows the graph compiler, the core of TrainDeeploy, which processes the training graph with awareness of the memory hierarchy of heterogeneous SoCs. The frontend firstly parses and validates operators to corresponding kernels before lowering them into a static intermediate representation. In the Midend, memory scheduling is performed in conjunction with operator tiling through a unified constraint-programming formulation. Tiling constraints, derived from kernel-specific requirements and hardware limits, define the feasible search space for tile sizes. At the same time, the scheduler applies liveness analysis to model tensor lifetimes. These two aspects

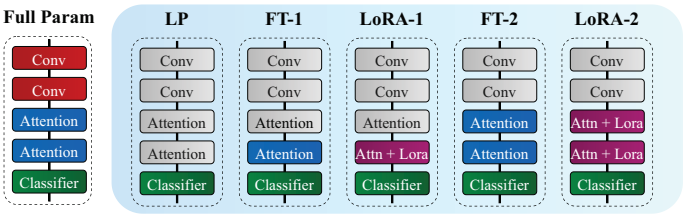

Fig. 3: Illustration of fine-tuning strategies evaluated on the CCT-2 model for on-device training. The convolutional tokenizer (**Conv**) is frozen in all strategies, while different subsets of transformer encoder blocks (**Attn**) and the classifier head are adapted. Five representative strategies are considered: **LP** (linear probing), only the classifier head is trained; **FT-1**, full fine-tuning of the last attention block; **LoRA-1**, low-rank adaptation (LoRA, rank $r = 4$) applied to the last attention block; **FT-2**, full fine-tuning of the last two attention blocks; and **LoRA-2**, LoRA (rank $r = 4$) applied to the last two attention blocks.

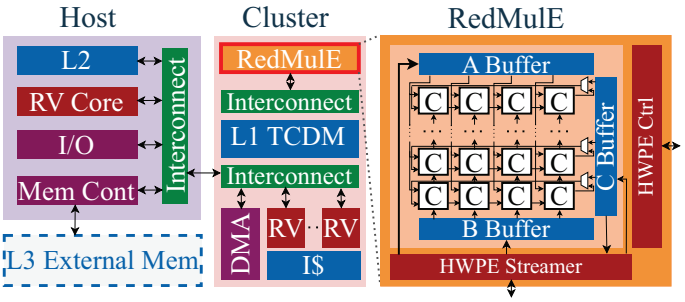

Fig. 4: Hardware setup of the PULP-based SoC modeled in GVSoC. The system consists of a host and an 8-core compute cluster with shared multi-banked L1 TCDM (128 KB), a hierarchical memory with L2 SRAM (2 MB) and external L3 HyperRAM (32 MB), and the Reduced Multiplier Engine (FP GEMM accelerator) (RedMulE) floating-point GEMM accelerator integrated with direct low-latency access to L1.

are solved jointly as a 2D bin-packing problem, producing a static allocation schedule using TetriSched [22] that minimizes peak memory usage across the hierarchy while ensuring feasibility for all operators of the full forward-backward graph. By default, L1 stores active tiles, while L2 holds weights, inputs, activations, and gradients; when the L2 budget is exceeded, tensors spill to L3. LoRA low-rank matrices are treated as standard GEMMs integrated seamlessly into the pipeline. Finally, the backend generates C code with optimized FP kernels for execution on the target platform, as illustrated in Figure 2(d), with accelerator-specific extensions discussed in the following paragraph.

c) Accelerator Support: TrainDeeploy extends Deeploy with end-to-end compiler support for on-board FP GEMM accelerators. The frontend identifies GEMMs and convolutional layers that can be supported by the accelerator, the midend applies accelerator-aware tiling strategies to respect L1 constraints, and the backend generates runtime kernels that manage synchronization with CPU operators and perform data layout transformations. By mapping GEMM-heavy kernels from both native GEMMs and lowered convolutions in forward and backward passes to the accelerator, TrainDeeploy effectively exploits on-chip compute units and alleviates the bottleneck of GEMM-dominated training computations.

V. TrainDeeploy Testing Scenario

A. Network Overview and Fine-Tuning Strategies

We adopt the Compact Convolutional Transformer (CCT-2/3x2) [23] as the target model to show the performance of TrainDeeploy. This lightweight vision transformer contains two convolutional tokenizer layers and two transformer encoder blocks (2 heads, 128-dimensional embedding, 128-dimensional hidden MLP), followed by attention-based sequence pooling. Overall, it has 0.28 M parameters (≈1.12 MB in FP32), requires only 67 MFLOPs per inference, and achieves 89.75% accuracy on CIFAR-10 and 66.93% on CIFAR-100.

To evaluate the configurability of our framework, we freeze the convolutional tokenizer and consider five representative fine-tuning strategies, as shown in Figure 3. The first is linear probing (LP), where all transformer blocks remain frozen and only the classifier head is trained, serving as the most lightweight baseline. The second one, FT-1, unfreezes the last attention block for full fine-tuning, while the third one, LoRA-1, applies rank-4 low-rank adaptation to the last attention block. Similarly, FT-2 unfreezes the last two attention blocks, and LoRA-2 applies rank-4 low-rank adaptation to these two blocks. Table I summarizes the trainable components, LoRA usage, accuracy, and cost of each strategy.

All training experiments are deployed on the hardware platform using Stochastic Gradient Descent (SGD) with a batch size of 1 and single-step parameter updates, with all computations (weights, activations, gradients, and optimizer states) in FP32.

B. On-Device Fine-Tuning Hardware Setup

As a practical representative of the class of heterogeneous SoCs discussed in Section IV, we selected a PULP instance simulated using the event-based GVSoC simulator [24]. As shown in Figure 4, a host leverages a RISC-V core, while the accelerator cluster includes 8 RISC-V processors and a specialized GEMM unit based on the RedMulE architecture [25]. The RISC-V cores implement the `RV32IMFCXpulp` ISA and share four Floating Point Units (FPUs), all connected to a 128 KB multi-banked L1, called Tightly-Coupled Data Memory (TCDM), via a logarithmic interconnect for single-cycle, low-latency access. The memory hierarchy comprises 2 MB L2 SRAM and 32 MB external L3 HyperRAM accessible through a HyperBus interface. Since the combined peak memory usage of CCT model weights, activations, and gradients exceeds the L2 capacity, L3 serves as the primary storage, with tensors staged through L2 and tiled into L1 at runtime. RedMulE is a FP GEMM engine based on a 12×4 systolic array of FPUs with a three-stage pipeline within a sub-100 mW power envelope. It is originally optimized for FP16/FP8 inference workloads. In this work, the datapath is extended to FP32 to ensure stable gradient accumulation during training. RedMulE is tightly coupled to the RISC-V cluster through the L1 TCDM. Based on the silicon prototype reported in [13], we assume a target frequency of 360 MHz for the whole SoC.

TABLE I: Comparison of fine-tuning strategies on CCT-2 and cost metrics. Accuracy is reported for 50-shot transfer (mean ± std over 30 runs). For reference, training from scratch yields 99.7% on MNIST and 94.0% on EuroSAT.

Strategy	Training Components	LoRA	Accuracy (50-Shot Transfer)		Cost	
			CIFAR-10→MNIST	CIFAR-10→EuroSAT	FLOPs (M)	Trained Param(MB)
Full FT	Entire model	✗	92.83 ± 0.91	64.85 ± 3.11	201	1.12
LP	Classifier head	✗	88.34 ± 0.42	76.70 ± 0.55	71	0.005
FT-1	Last attention block	✗	93.54 ± 0.38	78.94 ± 0.44	96	0.38
LoRA-1	Last attention block	✓	95.38 ± 0.29	77.00 ± 0.47	86	0.026
FT-2	Last 2 attention blocks	✗	94.62 ± 0.33	**81.52 ± 0.36**	126	0.76
LoRA-2	Last 2 attention blocks	✓	**96.00 ± 0.27**	80.50 ± 0.41	104	0.05

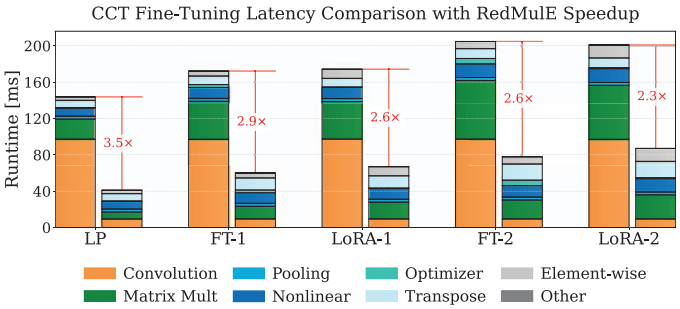

Fig. 5: End-to-end training latency across fine-tuning strategies. For each strategy, the left bar shows runtime using 8 cores without RedMulE acceleration, while the right bar shows runtime with RedMulE acceleration. In the accelerated LoRA-2 and FT-2 configurations, the measured latency corresponds to a peak throughput of up to 11 gradient updates per second under single-sample, end-to-end fine-tuning.

VI. Results

A. Transfer Learning Results

1) Fine-Tuning Setup: To show that our setup is credible for real-world applications, we report fine-tuning accuracy on two representative few-shot tasks: CIFAR-10 → MNIST and CIFAR-10 → EuroSAT, each under a 50-shot setting following the benchmark in [26]. Each model is trained for 100 epochs using SGD with a batch size of 8, an initial learning rate of 0.01, and cosine annealing down to 0.0005. The convolutional tokenizer is frozen, as it mainly provides generic low-level features, and unfreezing adds cost with little benefit. Final accuracy is reported as the average over the last five epochs, and each few-shot experiment is repeated 30 times for statistical robustness.

2) Fine-Tuning Accuracy: Figure 3 and Table I highlight the trade-off between accuracy and trainable footprint. As expected, linear probing yields the lowest accuracy, while adapting a single Transformer block (FT-1) already improves transferability. Applying LoRA to the same block (LoRA-1) further boosts MNIST accuracy to 95.4%, showing that low-rank adaptation can enhance generalization while reducing parameters. Extending to two blocks (FT-2) provides the highest EuroSAT accuracy (81.5%), whereas LoRA-2 achieves the best MNIST accuracy (96.0%) with only 0.05 MB trainable parameters, about 15× fewer than FT-2 for just 1% accuracy gap on EuroSAT. For reference, training from scratch yields 99.7% on MNIST and 94.0% on EuroSAT. Under the 50-shot setting, full fine-tuning of the entire model achieves 92.8% ± 0.9 on MNIST and 64.8% ± 3.1 on EuroSAT, exhibiting a substantial accuracy drop compared to configurations with a frozen convolutional tokenizer. Therefore, in the following experiments, we freeze the convolutional tokenizer and focus fine-tuning on the transformer layers.

3) Training Cost: As shown in Table I, LP is computationally the cheapest, but it is insufficient in accuracy. Full layer fine-tuning incurs a steep cost (0.38–0.76 MB), while LoRA keeps parameters small (0.026–0.05 MB) and operations lower than full fine-tuning. This operation decrease stems from reducing gradient and optimizer states while leaving activations unchanged. Overall, LoRA achieves nearly the same accuracy as full fine-tuning at a fraction of the cost, making Transformer adaptation increasingly practical within the tight memory and energy budgets of ultra-low-power SoCs, even when considering future extensions to more complex optimizers and larger batch sizes.

B. End-to-End Latency and Memory Footprint

Figure 5 shows the end-to-end training runtime per sample per step at 360 MHz for the five fine-tuning strategies listed in Table I. Each bar compares execution on the 8-core cluster (left) with RedMulE acceleration (right). On the baseline system, runtimes range from 143 ms to 200 ms. LP is the fastest strategy since it updates only the last layers, while runtime increases as more layers are tuned. For the same number of updated layers, FT-1 and FT-2 are slower than their LoRA counterparts because they update more parameters.

With RedMulE acceleration, runtimes drop to 41–87 ms, corresponding to a 2.3–3.5× speedup. LP achieves the highest gain (3.5×) as it has the highest ratio of GEMM and convolution operations, while LoRA-2 shows the smallest improvement (2.3×). Although runtime grows with the number of updated layers, LoRA can be slightly slower than FT after acceleration, due to its many small matrix multiplications limiting accelerator utilization and frequent low-rank transfers adding overhead.

Figure 6 profiles the dynamic peak memory usage and off-chip data movements across the five fine-tuning strategies. Figure 6 (a) shows the peak dynamic allocation in L3, accounting for activations and gradients but excluding weights and inputs. The reported values reflect the outcome of allocation optimizations performed by the `MiniMalloc` allocator in Deeploy, where the reduced gradients introduced by LoRA are stacked with activations. As training depth increases, the dynamic demand rises from less than 1 MB in LP to nearly 1.8 MB in FT-2, while LoRA lowers this footprint by 19–23% and reduces the overall off–chip data transfer volume to 0.62× that of full fine-tuning, i.e., a 1.6× reduction, as shown in Figure 6(b).

TABLE II: Comparison with related ultra–low-power on-device training work.

System	Hardware	Model	Memory Layout[2]	Params	FW+BW FLOP[1]	FLOP/cyc	Training Memory[2]
PULP-TrainLib [9]	PULP-SoC	Deep-AE / DS-CNN	64 KB	270K / 52.5K	0.8M / 7.6M	5.6	64 KB
POET [19]	NRF-52840	ResNet-18	256 KB + 32 GB[3]	11M	4.5G	N/A	<256 KB + N/A[4]
MiniLearn [10]	NRF-52840	CNNs	256 KB + 1 MB	79K / 47K / 12K	1.9M / 2.9M / 0.11M[5]	0.59 / 1.52 / 0.09	196 KB + 439 KB
TTE [8]	STM32F7	MCUNet	320 KB + 1 MB	0.48M	<69M[6]	<0.43	173 KB
Ours	**PULP-SoC**	**CCT**	**128 KB + 32 MB**	**0.28M**	**71M–126M**	**4.6**	**128 KB + 2.5 MB**
Ours	**PULP-SoC**	**Deep-AE**	**128 KB + 32 MB**	**270K**	**0.8M**	**13.4**	**128 KB**

[1] FW+BW = forward + backward operations (FLOP).
[2] On-chip SRAM + off-chip memory (e.g., external Flash or DRAM).
[3] POET leverages a 32 GB SD card/Flash to enable activations to be stored off-chip.
[4] POET's training scheme reduces peak SRAM usage but increases dependence on external Flash. The exact external Flash footprint has not been reported.
[5] For MiniLearn, convolution layers are pruned to 75% sparsity, which reduces operations but incurs up to 10% accuracy drop.
[6] For TTE, FLOPs are reported as an upper bound, since which layers are updated with sparsity or the exact sparsity ratios are not reported.

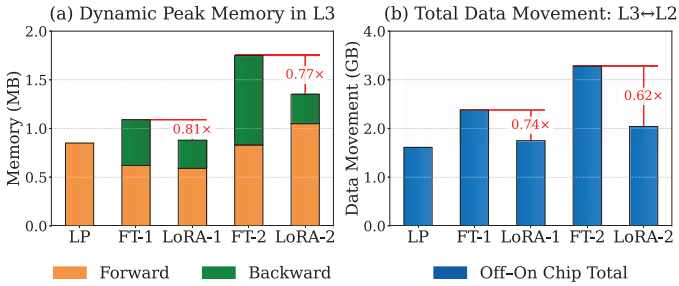

Fig. 6: (a) Dynamic peak L3 memory usage during forward and backward passes, including activations and gradients but excluding the fixed 1.12 MB weights and input. For each configuration, the reported peak memory corresponds to the maximum memory footprint over its training graph lifetime. (b) Aggregate off-chip data movement (L3↔L2), reported as the total transferred bytes across five strategies.

C. State-of-the-Art Comparison

We compare our framework with on-device training approaches targeting ultra–low-power platforms. Table II summarizes the key differences in hardware targets, model size, FLOP/cycle, and memory footprint. In addition to our Transformer results (CCT), we also include a small-network baseline (Deep-AE) to align with prior CNN/MLP-centric work.

a) PULP-TrainLib [9]: PULP-TrainLib is evaluated on the same PULP cluster configuration as ours (8 RISC-V cores with 4 FPUs), but is confined to L1-only execution and therefore limited to small autoencoders and CNNs, e.g., Deep-AE with 270K parameters and 0.8M forward+backward FLOPs or DS-CNN with 52.5K parameters and 7.6M FLOPs. It achieves a higher throughput of 5.6 FLOP/cycle, thanks to its highly optimized computational primitives. By contrast, our CCT fine-tuning involves a substantially larger model with 0.28M parameters and 71–126M FLOPs, i.e., about two orders of magnitude higher compute demand, which necessarily introduces off-chip/on-chip transfer overheads. Even under this heavier load, CCT still sustains 4.6 FLOP/cycle with the aid of the RedMulE GEMM accelerator. For a fair comparison, we benchmarked the same Deep-AE, which reaches 13.4 FLOP/cycle, 2.4× higher than PULP-TrainLib, highlighting the benefit of offloading GEMM-dominated kernels to the accelerator while maintaining scalability to larger models.

b) POET [19]: POET relies on paging and checkpointing to trade off SRAM and latency, enabling ResNet-18 training under a 256 KB SRAM + 32 GB Flash setup. However, this introduces extensive off-chip traffic and recomputation. The paper does not report end-to-end latency or FLOP/cycle, preventing a direct throughput comparison. In contrast, our CCT training fits within 128 KB on-chip SRAM + 2.5 MB external memory, reducing off-chip reliance by more than an order of magnitude while still sustaining 4.6 FLOP/cycle throughput.

c) MiniLearn [10] & TTE [8]: MiniLearn demonstrates pruning-driven training with CNNs, reporting 0.59–1.52 FLOP/cycle depending on network setup, but at the cost of up to 10% accuracy degradation. TTE combines pruning and quantization-aware sparsity to fit MCUNet within 320 KB SRAM and 1 MB Flash, but reaches less than 0.43 FLOP/cycle. Compared with these CNN-focused methods, our CCT fine-tuning sustains 4.6 FLOP/cycle (3–10× higher) without sacrificing accuracy (50-shot 64.9% Full-BP vs. 80.5% LoRA-2 for EuroSAT), as shown in Table I.

d) Comparison: Unlike prior MCU training approaches that trade accuracy for sparsity or rely on paging/recomputation, TrainDeeploy enables Transformer training by jointly addressing memory and compute. Our results show 71–126M forward+backward FLOPs handled within 128 KB on-chip SRAM + 2.5 MB external memory, while achieving up to 96% accuracy with LoRA fine-tuning. Despite the additional overhead of L3–L2 transfers, our framework sustains 4.6 FLOP/cycle thanks to the RedMulE accelerator, at least 3× more compute-efficient than sparsity-based methods and with lower external memory dependence than paging-based approaches. Moreover, TrainDeeploy is orthogonal to sparsity and pruning techniques, and can incorporate them to further reduce training cost.

VII. Conclusion and Future Work

This work presented TrainDeeploy, a framework that enables Transformer training on ultra-low-power heterogeneous edge devices. Testing on a SoA SoC, using LoRA, we reduce trainable parameters and stored gradients by 15× and cut off-chip transfers by 1.6×. Offloading GEMMs to RedMulE yields the first hardware-accelerated LoRA training at the edge, achieving 2.3-3.5× speedups over an 8-core RISC-V cluster. TrainDeeploy provides a robust and unified toolchain for on-device learning at the extreme edge.

VIII. Acknowledgment

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

REFERENCES

[1] J. Lin, L. Zhu, W. Chen, W. Wang, and S. Han, "Tiny Machine Learning: Progress and Futures [Feature]," *IEEE Circuits and Systems Magazine*, vol. 23, no. 3, pp. 8–34, Oct. 2023.

[2] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "AI and ML Accelerator Survey and Trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2022, pp. 1–10.

[3] M. Scherer, L. Macan, V. J. B. Jung, P. Wiese, L. Bompani, A. Burrello, F. Conti, and L. Benini, "Deeploy: Enabling Energy-Efficient Deployment of Small Language Models On Heterogeneous Microcontrollers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 11, pp. 4009–4020, Nov. 2024.

[4] S. Zhu, T. Voigt, F. Rahimian, and J. Ko, "On-device Training: A First Overview on Existing Systems," *ACM Transactions on Sensor Networks*, vol. 20, no. 6, pp. 1–39, Nov. 2024.

[5] G. Hinton, "The Forward-Forward Algorithm: Some Preliminary Investigations," Dec. 2022. [Online]. Available: http://arxiv.org/abs/2212.13345

[6] Y. Zhang, P. Li, J. Hong, J. Li, Y. Zhang, W. Zheng, P.-Y. Chen, J. D. Lee, W. Yin, M. Hong, Z. Wang, S. Liu, and T. Chen, "Revisiting Zeroth-Order Optimization for Memory-Efficient LLM Fine-Tuning: A Benchmark," May 2024. [Online]. Available: http://arxiv.org/abs/2402.11592

[7] D. d. Blanken and C. Frenkel, "Chameleon: A MatMul-Free Temporal Convolutional Network Accelerator for End-to-End Few-Shot and Continual Learning from Sequential Data," May 2025, arXiv:2505.24852.

[8] J. Lin, L. Zhu, W. Chen, W. Wang, C. Gan, and S. Han, "On-Device Training Under 256KB Memory," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Nov. 2022, pp. 22 941–22 954.

[9] 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: 22nd International Conference, SAMOS 2022, Samos, Greece, July 3–7, 2022, Proceedings*, Jul. 2022, p. 200–216.

[10] C. Profentzas, M. Almgren, and O. Landsiedel, "MiniLearn: On-Device Learning for Low-Power IoT Devices," in *Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks*, Jan. 2023, p. 1–11.

[11] F. Paissan, D. Nadalini, M. Rusci, A. Ancilotto, F. Conti, L. Benini, and E. Farella, "Structured Sparse Back-propagation for Lightweight On-Device Continual Learning on Microcontroller Units," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 2172–2181.

[12] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," Oct. 2021, arXiv:2106.09685.

[13] A. S. Prasad, M. Scherer, F. Conti, D. Rossi, A. Di Mauro, M. Eggimann, J. T. Gómez, Z. Li, S. S. Sarwar, Z. Wang, B. De Salvo, and L. Benini, "Siracusa: A 16 nm Heterogenous RISC-V SoC for Extended Reality with At-MRAM Neural Engine," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 7, pp. 2055–2069, Jul. 2024.

[14] STMicroelectronics. (2024) STM32N6 Series Microcontrollers with Neural-ART Accelerator. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32n6-series.html

[15] Analog Devices, Inc. (2022) MAX78000 Ultra-Low Power Microcontroller with CNN Accelerator. [Online]. Available: https://www.analog.com/en/products/max78000.html

[16] GreenWaves Technologies. (2020) GAP9 AIoT Ultra-Low-Power Application Processor. [Online]. Available: https://www.semi.org/en/greenwaves-gap9-ai-iot-ulp-app-processor-on-gfs-22fdx

[17] Arm Limited. (2020) Arm Ethos-U55 MicroNPU. [Online]. Available: https://developer.arm.com/Processors/Ethos-U55

[18] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Dec. 2020, pp. 11 285 – 11 297.

[19] S. G. Patil, P. Jain, P. Dutta, I. Stoica, and J. Gonzalez, "POET: Training Neural Networks on Tiny Devices with Integrated Rematerialization and Paging," in *Proceedings of the 39th International Conference on Machine Learning*, Jun. 2022, pp. 17 573–17 583.

[20] K. Prabhu, R. M. Radway, J. Yu, K. Bartolone, M. Giordano, F. Pedding-haus, Y. Urman, W. Khwa, Y. Chih, M. Chang, S. Mitra, and P. Raina, "MINOTAUR: A Posit-Based 0.42–0.50-TOPS/W Edge Transformer Inference and Training Accelerator," *IEEE Journal of Solid-State Circuits*, vol. 60, no. 4, pp. 1311–1323, Apr. 2025.

[21] Microsoft, "ONNX Runtime," https://github.com/microsoft/onnxruntime, 2025.

[22] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "TetriSched: Global Rescheduling with Adaptive Plan-ahead in Dynamic Heterogeneous Clusters," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 1–16.

[23] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, "Escaping the Big Data Paradigm with Compact Transformers," Jun. 2022, arXiv:2104.05704.

[24] GVSoC, "GVSoC Simulator Project," https://github.com/gvsoc/gvsoc.

[25] Y. Tortorella, L. Bertaccini, L. Benini, D. Rossi, and F. Conti, "RedMule: A Mixed-Precision Matrix-Matrix Operation Engine for Flexible and Energy-Efficient On-Chip Linear Algebra and TinyML Training Acceleration," *Future Generation Computer Systems*, vol. 149, pp. 122–135, Dec. 2023.

[26] Y. Guo, N. C. Codella, L. Karlinsky, J. V. Codella, J. R. Smith, K. Saenko, T. Rosing, and R. Feris, "A Broader Study of Cross-Domain Few-Shot Learning," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, Aug. 2020, p. 124–141.