POLITECNICO DI TORINO
Repository ISTITUZIONALE

A System-Level Performance Analysis of On-Device Learning on an Ultra-Low-Power Edge System

Original
A System-Level Performance Analysis of On-Device Learning on an Ultra-Low-Power Edge System / Rusci, M., Hamdi, M.A., Jahier Pagliari, D., Conti, F., Burrello, A.. - (2026), pp. 210-213. (23rd ACM International Conference on Computing Frontiers, CF 2026 Catania (ITA) May 19 - 21, 2026) [10.1145/3801487.3806371].

Availability:
This version is available at: 11583/3016396 since: 2026-10-07T12:22:25Z

Publisher:
Association for Computing Machinery

Published
DOI:10.1145/3801487.3806371

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

27 November 2026

A System-Level Performance Analysis of On-Device Learning on an Ultra-Low-Power Edge System

Manuele Rusci
KU Leuven
Leuven, Belgium
manuele.rusci@kuleuven.be

Mohamed Amine Hamdi
Politecnico di Torino
Torino, Italy
mohamed.hamdi@polito.it

Daniele Jahier Pagliari
Politecnico di Torino
Torino, Italy
daniele.jahier@polito.it

Francesco Conti
Università di Bologna
Bologna, Italy
f.conti@unibo.it

Alessio Burrello
Politecnico di Torino
Torino, Italy
alessio.burrello@polito.it

Abstract

Training Deep Neural Networks (DNNs) on ultra-low-power processing systems is often hindered by the memory-intensive nature of backpropagation. This paper examines the trade-offs between memory and computational costs when mapping DNN training onto an ultra-low-power RISC-V-based System-on-Chip (SoC) with an 8-core accelerator, 1.5 MB on-chip memory, and a 32 MB off-chip memory. To this end, we develop a framework that automates the deployment of complete training graphs across heterogeneous SoCs. The generated code interleaves memory transfer operations with layer-wise processing functions, which are executed either on the host CPU or, for convolutional layers, on the multi-core accelerator. Overall, our work is the first to provide a full system-level analysis of end-to-end DNN training on OS-less, memory-constrained (32 MB of off-chip memory) edge systems, and we show that computation is the primary bottleneck by accounting for up to 91% of the total energy consumption.

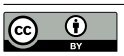
CCS Concepts

• **Computer systems organization** → **Embedded software**; • **Computing methodologies** → *Machine learning*; • **Software and its engineering** → *Development frameworks and environments.*

Keywords

On-Device Learning, Heterogeneous Processing Systems, Ultra-low power training, Training on-device

ACM Reference Format:
Manuele Rusci, Mohamed Amine Hamdi, Daniele Jahier Pagliari, Francesco Conti, and Alessio Burrello. 2026. A System-Level Performance Analysis of On-Device Learning on an Ultra-Low-Power Edge System. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers (CF '26), May 19–21, 2026, Catania, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3801487.3806371



This work is licensed under a Creative Commons Attribution 4.0 International License.
CF '26, Catania, Italy
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2568-5/26/05
https://doi.org/10.1145/3801487.3806371

1 Introduction

Training DNNs on ultra-low power processing (wearable) nodes, namely On-Device Learning (ODL), is crucial for safeguarding personal data confidentiality and facilitating on-site DNN personalization [15, 16]. However, the backpropagation algorithm, which remains the prevailing method for DNN training, poses major challenges due to its substantial memory requirements. In addition to requiring ~3× more computation than inference-only execution, training must retain the intermediate activations generated during the forward pass, which for TinyML-scale models can require up to 3.6 MB of memory per sample [2].

This work is the first to adopt a system-level methodology to map ODL tasks onto an ultra-low-power heterogeneous platform. The considered system features the GAP9 SoC, which includes a host RISC-V CPU, 1.5 MB of on-chip memory, and an 8-core accelerator, and it is connected via octo-SPI to an external ultra-low-power 32 MB RAM (AP Memory's APS256XXN). In contrast to previous on-device learning frameworks [3, 7, 9, 14], which are constrained by limited on-chip memory and neither exploit hardware accelerators nor external memory, our methodology enables full on-device training by offloading compute-intensive operations and using external memory to manage intermediate results.

The frontend of our toolchain takes a DNN model and derives the corresponding backward graph via auto-differentiation. The backend then employs the TVM-based MATCH code generator [6], which we extend to manage the temporal placement of inputs, outputs, and intermediate tensors across internal and external memories. Forward and backward layers are executed either on the host CPU or, for compute-intensive convolutions, offloaded to an 8-core hardware accelerator.

In summary, this paper makes the following contributions: *(i)* we provide a vertical stack to deploy end-to-end ODL tasks on heterogeneous SoCs coupled with non-trivial memory hierarchies made of on-chip and external memories, [1] and *(ii)* we report a system-level memory-compute analysis through measurements on a state-of-the-art HW prototype. Our findings show that ODL tasks on this class of systems are largely compute-bound, requiring 65–91% of total training energy. While advanced integration technologies may further reduce memory energy, a more impactful direction is the

[1]The code is available at: https://github.com/mrusci/on_device_learning_on_mcu_deployment.

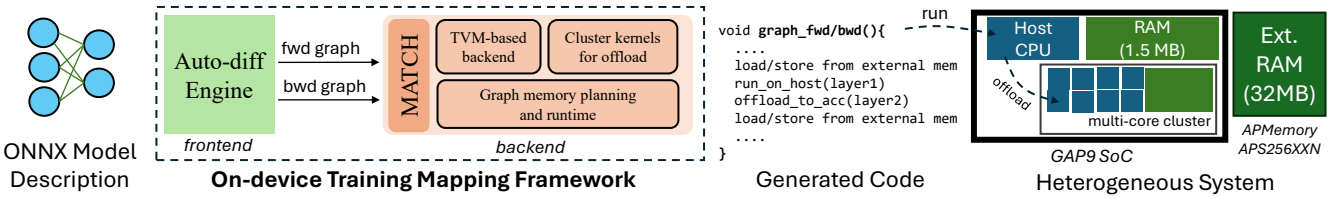


Figure 1: Automatic deployment of DNN training on an ultra-low power system composed of a heterogeneous processor and an external RAM memory.

energy-efficient acceleration of non-convolution operations such as backward normalization, supported by appropriate tooling.

2 Related Work

One of the earliest on-device learning frameworks, *TinyOL* [14], deploys a trainable classification layer on top of a frozen DNN backbone on an Arduino Nano 33 BLE MCU. The *AIfES* [18] training framework for ARM Cortex-M MCUs defines all intermediate and parameter tensor allocations at compile time within the on-chip RAM, implicitly limiting the maximum DNN size it can support. The *Tiny Training Engine* (TTE) [9] framework combines 8-bit training with a sparse update logic to skip the gradient computation for less relevant layers. At compile-time, their code generator produces a fully integer training graph via automatic differentiation and then generates software for single-core ARM Cortex-M CPUs, leveraging pre-optimized compute kernels. The allocation plan for a single-level memory is defined with a first-fit rule. Deutel et al. [4] introduced a dynamic sparse scheme for on-device training in which the subset of weights to be learned can change at every epoch. The proposed approach was evaluated on multiple single-core ARM Cortex-M CPUs, and the analysis was limited to small DNNs that fit within the on-chip memory. Still targeting MCUs, the *TinyTTA* engine is built upon TensorFlow Lite Micro (TFLM), to deploy a frozen model with multiple trainable heads for early-exit [7]. Their code generator works for 1-sample batches and was tested on a ARM Cortex M7 without the use of external memory. With respect to these works: *(i)* we target a more advanced heterogeneous platform with a host CPU coupled to a flexible HW accelerator and an external memory; *(ii)* we derive insights on the memory-compute costs of training, while previous works only reported the aggregated latency and (sometimes) energy statistics; *(iii)* we analyze complete training workloads using larger DNNs (enabled by the use of external memory) with normalization layers, also supporting batch sizes greater than 1.

To address the constraints imposed by limited on-chip memory, several works such as *DaCapo* [8] employ *checkpointing* techniques to store only a subset of the intermediate tensors produced during the forward pass [5, 13, 17]. This strategy requires recomputing the missing activations during the backward pass from the stored tensors. In our system, this optimization is less energy-efficient than loading the intermediate tensors directly from external memory and is therefore not adopted. Unlike *DaCapo*, which targets a single-core MCU and relies on on-chip FLASH memory for storing intermediate results, our system uses ultra-low-power RAM and a heterogeneous CPU, resulting in over 20× lower energy consumption on similar workloads.

3 ODL Mapping Methodology

This section describes our flow to deploy DNN training workloads on an ultra-low-power system. To avoid the batch-size dependency introduced by BatchNorm layers, which require storing batch-wide statistics and enforcing batch processing, we replace them with *GroupNorms*. A GroupNorm computes normalization independently for each sample, enabling memory-efficient sample-by-sample training (with gradient accumulation) while reducing the activation-buffer size by up to one to two orders of magnitude. Despite this simplification, our experiments (Section 4) show that GroupNorm with small effective batches maintains accuracy comparable to BatchNorm trained with much larger batches.

The on-device training deployment targets a board with a state-of-the-art *GAP9* processor[2], connected via OctoSPI to an external RAM memory (AP Memory's *APS256XXN*). The GAP9 SoC couples a RISC-V host CPU with a cluster of 8 general-purpose RISC-V CPUs (identical to the host) as the on-chip accelerator, and a 128 kB scratchpad (non-cached) memory. The host CPU is equipped with a larger on-chip memory (1.5 MB) that serves the overall execution. The target processor is representative of a family of novel MCUs that integrate a general-purpose CPU with on-chip HW accelerators [1], and we selected this device in particular for its best-in-class flexibility, provided by the programmable 1+8 cores, and ultra-low power consumption. On the other hand, the external memory has a capacity of 32 MB with a read/write bandwidth of 370 M bytes per second, which is set by the maximum processor clock speed (370 MHz). This type of RAM is suited for ultra-low-power applications because it does not require a power-hungry IO interface commonly found on the processor side to communicate with external DRAM.

Given a DNN graph description, our deployment flow (Fig. 1) produces optimized `C` functions for the forward and backward passes, interleaving calls to layer-wise processing functions (executed on the host CPU or offloaded to the cluster accelerator) and memory copies between the external and the on-chip memory. Internally, the framework works in two steps. The front end performs automatic differentiation (*auto-diff*) and generates both the forward and backward computation graphs of the DNN model. In our work, we build on the auto-diff infrastructure of the TTE compiler [9] and extend it by enriching the set of backward functions and their hardware-aware definitions.

The backend relies on the MATCH compiler [6] and converts the graphs into executable code. For every conv2d or conv2d_transpose operator, MATCH: *(i)* defines a temporal schedule, i.e., tiling of the

[2]designed by *GreenWaves Technologies*

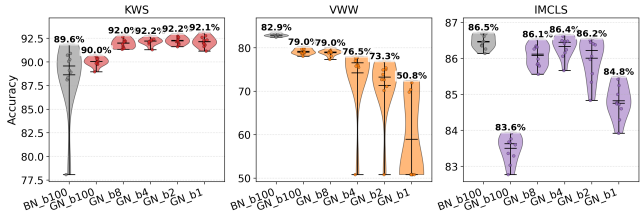


Figure 2: Accuracy with respect to the batch size (*b*). BN and GN stand for BatchNorm and GroupNorm, respectively. The box plot statistics are based on 10 training runs. The mean values are reported above the boxes.

Table 1: Memory (kB) of ODL tasks running on the host CPU.

		kws		vww		imcls	
		fw	bw	fw	bw	fw	bw
ext mem	params	93.4	78.3	837.0	822.8	305.1	291.4
	activ.	1189.8	1189.8	3727.0	3727.0	1260.3	1260.3
	grads		93.4		837.0		305.1
	total	**1283.1**	**1361.4**	**4564.0**	**5386.8**	**1565.5**	**1856.9**
on-chip mem		159.8	276.1	577.0	813.0	336.3	528.0

layers' operands and loop ordering, using a model-based engine (ZigZag [10]) and *(ii)* selects one of the hand-optimized convolutional C functions for the 8-core cluster adapted from PULP-Trainlib [11]. All tensors and computations are in `float32`. In addition, we developed a custom graph executor with a memory planner that manages external memory, replacing TVM's *AOT executor*. The planner computes a static memory layout at compile time using a greedy first-fit strategy based on tensor lifetimes: tensors are sorted by decreasing lifetime (and size), then placed in the first available region covering their entire lifetime. If a tensor cannot be held in on-chip memory for its full duration, it is temporarily offloaded to external memory when unused. Parameter and I/O tensors always reside in external memory and are loaded on demand, freeing on-chip memory after use.

4 Experimental Result

Our experimental analysis aims to answer the question: *"is on-device learning memory-bound on the target ultra-low power system?"*. To address this, we applied our methodology to the three largest DNNs from the MLPerf Tiny Benchmark suite [2]: DS-CNN for keyword spotting with 23.3 k trainable parameters (indicated as *kws* in the following), MobileNet for visual wake words (*vww*, 209.3 k params) and ResNet8 for image classification (*imcls*, 76.2 k params).

Fig. 2 illustrates the accuracy measured on the benchmarked DNNs after replacing the BatchNorm (BN) with GroupNorm (GN) layers and by varying the batch size (b). We follow [3] and set the group size to 8 channels. The models are trained from scratch with the hyperparameter settings defined by the original codebase [12]. The baseline is identified by BN_b100 in the plot. For *kws* and *imcls*, the GN layer does not introduce significant accuracy drops with respect to the baseline up to a batch size of 1 (a few outliers were observed within the *kws* baseline). Conversely, for the larger *vww*, we observe a decrease in accuracy and an increase in variance for small batches, likely due to the higher sensitivity of deeper models to noisy normalization statistics computed on a single sample.

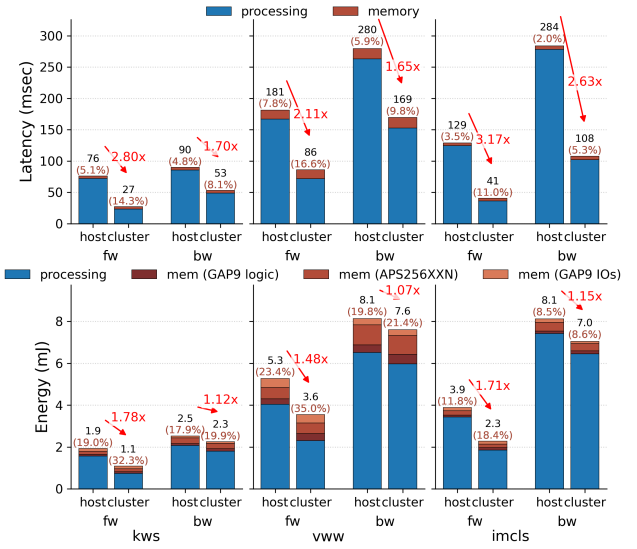


Figure 3: Latency and energy measurements when running ODL on the host CPU (*host*) or on the *cluster* accelerator. Processing and memory costs are shown separately for the *fw* and *bw* tasks. The total values and the memory usage ratio (in parentheses) are indicated at the top of each bar.

Following these results, and consistent with prior findings [3], we identify a batch size of 8 with GN as an optimal configuration, achieving accuracy on par with the baseline.

Tab. 1 reports the external and on-chip memory occupation for the (single-sample) forward and the backward passes. The storage of the activation feature maps (output of the forward and input of the backward) dominates the requirement. The memory for parameters (only a subset of them is loaded during backward) and gradients constitutes only the 12% of the total for *kws*, increasing to up to the 32% for *imcls*. For the larger *vww*, the memory usage reaches 5.26 MB, which fits within the available external memory. Less than 1.8 MB of memory is instead required for training *kws* and *imcls*. Remarkably, thanks to the batch-agnostic GN layers, this memory footprint does not linearly scale with respect to the batch size.

Conversely, the on-chip memory footprint is up to 276 kB, 813 kB, and 528 kB for respectively *kws*, *vww* and *imcls*, which depends on the peak memory requirement among DNN layers. Note that the requirement for the backward pass is, on average, 1.6× that of the forward. This is mainly due to the extra temporary buffer required by conv2d_transpose operators.

Fig. 3 summarizes latency and energy measurements for the per-sample execution of the forward and backward functions. We measure the power consumption of the system using a Nordic Power Profiler Kit II, and we distinguish between the consumption of the GAP9 logic, GAP9 IOs, and the APS256XXN memory. We compare two configurations: *host*, where only the host CPU is involved in the computation (the 8-core cluster is off), and *cluster*, which makes use of the accelerator. The latency and energy analyses distinguish the costs of the processing vs. the memory transfer phases, i.e., when the host CPU controls the ext. memory copies and waits for their completion.

Table 2: Comparison with SoA. MMAC only accounts for Multiply and Accumulate of convs.

DNN models		trainable params	MMAC fwd	latency (msec)	energy (mJ)	MMAC bwd	latency (msec)	energy (mJ)
DaCapo [8]	DS-CNN	36 k	n/a	2195	220	n/a	10201	1075
	MobileNet	8.7 k	n/a	1819	190	n/a	80813	8330
This Work	DS-CNN (*kws*)	23.3 k	2.7	27.1	1.1	5.0	52.9	2.3
	MobileNet (*vww*)	209.3 k	7.5	85.8	3.6	14.9	169.5	7.6
	ResNet8 (*imcls*)	76.2 k	12.5	40.7	2.3	32.4	107.9	7.0

In the *host* configuration, the memory transfers take up only up to 7.8% of the total execution time and can be as low as 3.5% and 2% for the forward and backward passes of the *imcls* model. The compute-memory balance shifts when using the accelerator. On average, the latency speedup reaches 2.7× (forward) and 2× (backward) when the convolution kernels are offloaded to the cluster. This increases the memory latency share from 5.3% (*imcls* bw) to 16.6% (*vww* fw) of the total. The energy reduction achieved by the *cluster* w.r.t. the *host* configuration is less than the reduction in latency, with an average of 39% (forward) and 10% (backward) savings. This is explained by the 1.7× higher power consumption (on average) when the cluster is active and fully loaded vs. the *host* setup, with a peak of 2.3× for *imcls* bw. So, while benefiting from the acceleration, *the compute phase remains the bottleneck for on-device training, accounting for 65% to 91% of the total energy consumption.*

In the lower part of Fig. 3, we break down the energy costs attributed to memory operations. The external memory chip accounts for only 50% of the memory energy. On average, the 23% and the 27% of the total consumption can be attributed, respectively, to the GAP9 logic during memory transfers (i.e., the CPU waiting for load/store completion) and to the physical GAP9 IOs. The latter cost could be alleviated by leveraging more advanced and compact packaging solutions. On the contrary, the use of asynchronous memory transfers could partially reduce energy consumption due to logic, while also further speeding up the application. However, fully exploiting asynchronous copies will necessitate a more sophisticated memory-scheduling strategy.

Tab. 2 compares the results of our analysis with the measurements reported in DaCapo [8], which is the only other work breaking down the latency and energy costs of ODL on MCUs. Notably, the DS-CNN and MobileNet benchmarked from DaCapo differ from the MLPerf Tiny DNNs and the lack of topology details prevents us from analyzing the same models. However, our *vww*'s MobileNet executes 21× faster and consumes 53× less energy than the MobileNet used in [8], which also features 24× fewer parameters. Of this gap, only roughly 370MHz / 180MHz ≈ 2× comes from our higher clock frequency. The majority of it can be attributed to a combination HW acceleration (they target single-core MCUs), faster and more energy-efficient memory technology (they use the MCU's internal FLASH) and a better optimized software runtime. For backward, the gap increases by more than two orders of magnitude due to DaCapo's recomputation strategy.

5 Discussion and conclusions

This work analyzed DNN on-device training acceleration on a ultra-low-power processing system, revealing that processing costs are the predominant ones. Further optimizations are needed to accelerate non-conv layers (e.g., GroupNorm linear operations), while convolution operators can also benefit from domain-specific HW tensor cores and novel mixed- and low-precision training policies, which were not used in our setup. On the memory side, additional efficiency may come from advanced integration technologies and support for asynchronous data transfers. Future work will also focus on generalizing our approach across multiple system architectures and developing more abstract performance models to analyze the gap between peak and effective performance.

References

[1] Lennart Bamberg, Filippo Minnella, Roberto Bosio, Fabrizio Ottati, Yuebin Wang, Jongmin Lee, Luciano Lavagno, and Adam Fuks. 2025. eIQ Neutron: Redefining edge-AI inference with integrated npu and compiler innovations. *arXiv preprint arXiv:2509.14388* (2025).

[2] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. 2021. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597* (2021).

[3] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems* 33 (2020), 11285–11297.

[4] Mark Deutel, Frank Hannig, Christopher Mutschler, and Jürgen Teich. 2024. On-Device Training of Fully Quantized Deep Neural Networks on Cortex-M Microcontrollers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[5] In Gim and JeongGil Ko. 2022. Memory-efficient DNN training on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 464–476.

[6] Mohamed Amine Hamdi, Francesco Daghero, Giuseppe Maria Sarda, Josse Van Delm, Arne Symons, Luca Benini, Marian Verhelst, Daniele Jahier Pagliari, and Alessio Burrello. 2025. MATCH: Model-Aware TVM-based Compilation for Heterogeneous Edge Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2025).

[7] Hong Jia, Young Kwon, Alessio Orsino, Ting Dang, Domenico Talia, and Cecilia Mascolo. 2024. TinyTTA: Efficient Test-time Adaptation via Early-exit Ensembles on Edge Devices. *Advances in Neural Information Processing Systems* 37 (2024), 43274–43299.

[8] Osama Khan, Gwanjong Park, and Euiseong Seo. 2023. DaCapo: An On-Device Learning Scheme for Memory-Constrained Embedded Systems. *ACM Transactions on Embedded Computing Systems* 22, 5s (2023), 1–23.

[9] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256kb memory. *Advances in Neural Information Processing Systems* 35 (2022), 22941–22954.

[10] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. 2021. ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators. *IEEE Trans. on Comput.* 70, 8 (2021), 1160–1174.

[11] Davide Nadalini, Manuele Rusci, Giuseppe Tagliavini, Leonardo Ravaglia, Luca Benini, and Francesco Conti. 2022. PULP-TrainLib: Enabling on-device training for RISC-V multi-core MCUs through performance-driven Autotuning. In *International Conference on Embedded Computer Systems*. Springer, 200–216.

[12] Daniele Jahier Pagliari, Matteo Risso, Beatrice Alessandra Motetti, and Alessio Burrello. 2023. Plinio: a user-friendly library of gradient-based methods for complexity-aware DNN optimization. In *2023 Forum on Specification & Design Languages (FDL)*. IEEE, 1–8.

[13] Shishir G Patil, Paras Jain, Prabal Dutta, Ion Stoica, and Joseph Gonzalez. 2022. POET: Training Neural Networks on Tiny Devices with Integrated Rematerialization and Paging. In *Int. Conf. on Machine Learning*. PMLR, 17573–17583.

[14] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[15] Manuele Rusci, Francesco Paci, Marco Fariselli, Eric Flamand, and Tinne Tuytelaars. 2024. Self-learning for personalized keyword spotting on ultra-low-power audio sensors. *IEEE Internet of Things Journal* (2024).

[16] Andrea Soltoggio, Eseoghene Ben-Iwhiwhu, Vladimir Braverman, Eric Eaton, Benjamin Epstein, Yunhao Ge, Lucy Halperin, Jonathan How, Laurent Itti, Michael A Jacobs, et al. 2024. A collective AI via lifelong learning and sharing at the edge. *Nature Machine Intelligence* 6, 3 (2024), 251–264.

[17] Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, and Xuanzhe Liu. 2022. Melon: Breaking the memory wall for resource-efficient on-device machine learning. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*. 450–463.

[18] Lars Wulfert, Johannes Kühnel, Lukas Krupp, Justus Viga, Christian Wiede, Pierre Gembaczka, and Anton Grabmaier. 2024. AIfES: A next-generation edge AI framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 6 (2024), 4519–4533.