TU Delft

Delft University of Technology

Document Version
Final published version

Licence
CC BY

Citation (APA)
Liu, H., Suresh, S., & Zuniga, M. (2026). WiP Paper: eODL: Energy-Aware On-Device Training on Microcontrollers. In *EWSN '26: Proceedings of the 2026 International Conference on Embedded Wireless Systems and Networks* (Vol. 2026, pp. 1-6). (International Conference on Embedded Wireless Systems and Networks). https://doi.org/10.3217/j1ep-fx24

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

TU Delft

Delft University of Technology

Document Version
Final published version

Licence
Dutch Copyright Act (Article 25fa)

Citation (APA)
Liu, H., Suresh, S., & Zuniga, M. (2026). WiP Paper: eODL: Energy-Aware On-Device Training on Microcontrollers. *International Conference on Embedded Wireless Systems and Networks*, *2026*, 1-6.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

WiP Paper: *e*ODL: Energy-Aware On-Device Training on Microcontrollers

Hao Liu
h.liu-8@tudelft.nl
Delft University of Technology

Shashwath Suresh
shashwath2000@live.com
Delft University of Technology

Marco Zuniga
m.a.zunigazamalloa@tudelft.nl
Delft University of Technology

Abstract

Deep learning models can be trained on microcontrollers (MCUs) for privacy-sensitive applications, but such training is severely constrained by tiny SRAM capacity and limited battery energy. Existing on-device learning methods optimise memory usage by selectively freezing layers so that training fits within SRAM budgets, yet none explicitly account for training energy. A strategy that is feasible in memory may therefore still consume significantly more energy than necessary, shortening battery life on energy-constrained devices.

We present *e*ODL, an energy-aware on-device learning framework. *e*ODL exploits two observations: (1) counts of multiply-accumulate operations (MACs) predict training energy with high fidelity ($R^2 = 0.993$), so energy can be estimated; and (2) energy-aware layer-update selection can be solved efficiently with tabular dynamic programming. Using these, *e*ODL selects which layers to update under a hard SRAM budget while balancing energy against accuracy.

We implement *e*ODL end to end on STM32 MCUs and evaluate it on MCUNetV2 fine-tuned on the Flowers-102 dataset. Under a 50 kB training-memory budget, *e*ODL matches state-of-the-art accuracy while reducing measured on-device energy by 22.6% per epoch.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Embedded software**.

Keywords

On-device training, microcontrollers

1 Introduction

Deep learning models can run inference on resource-constrained microcontrollers (MCUs), enabling applications such as image, audio, and gesture recognition [6, 8, 9, 16]. But training these models requires user-specific data. Sending user data to centralized servers raises privacy concerns and incurs communication overhead, making cloud-based personalization impractical [4, 7, 18].

To resolve this tension between personalization and privacy, recent research has explored on-device learning (ODL) [18]. In this paradigm, a model is first pre-trained on a large-scale server-side dataset and then deployed to the MCU, where it is further trained locally using private user data.

Limitation. Recent ODL systems train models on memory-constrained MCUs, but they leave energy largely unaddressed. Methods such as Tiny Training [7] selectively freeze layers to fit SRAM budgets below 320 KB, typical of STM32-class MCUs. However, these methods optimise only memory, not energy. Memory and energy are governed by different factors: SRAM is dominated by the intermediate activations stored for backpropagation [7], whereas training energy is dominated by the large number of multiply-accumulate operations (MACs) in the backward pass [18]. A training strategy that satisfies the SRAM budget may still consume far more energy than necessary, significantly shortening battery life on energy-constrained deployments.

Challenges. Bridging this gap requires addressing two key challenges. First, energy modeling: on MCUs, the energy cost of training is difficult to measure directly, making it hard to reason about the trade-off between accuracy and energy. Second, strategy planning: choosing which layers to update under strict memory constraints while balancing accuracy and energy is inherently combinatorial. Existing planners focus on memory feasibility, but none explicitly models and penalises on-device training energy under MCU constraints [7].

To address these challenges, we propose *e*ODL, an energy-aware on-device training framework. *e*ODL has two offline components: a model profiler that estimates training energy from MAC counts, and a training strategy generator that selects which layers to update under a hard SRAM budget with energy penalised directly in the objective. Both run on the host before deployment; the MCU only executes the resulting strategy. The main contribution of *e*ODL is to integrate platform-calibrated energy estimation and an energy-aware objective into existing sparse-update training, making energy an explicit factor in strategy selection under a hard SRAM budget.

In summary, this paper makes the following contributions:

- **Integration of energy estimation for on-MCU training.** We calibrate a linear MAC-to-energy proxy on the STM32 board ($R^2 = 0.993$, median relative error 2.3%) and integrate it into sparse-update selection, providing the energy estimates required for joint memory–energy–accuracy optimisation.
- **Energy-aware strategy planner.** Existing planners optimise only memory [7], with none penalising energy. We introduce an energy penalty directly into the objective and solve the resulting problem with tabular dynamic programming, which jointly optimises across all reachable layers within the memory budget. A lightweight greedy variant is also provided for fast offline search.
- **End-to-end implementation and evaluation:** We implement *e*ODL as a fully deployable end-to-end system on STM32 microcontrollers and release the complete artifact at https://github.com/haoliu001/eODL/. Experiments show that *e*ODL achieves accuracy on par with Tiny Training while reducing per-epoch physical energy consumption by 22.6% under a 50 kB memory budget.

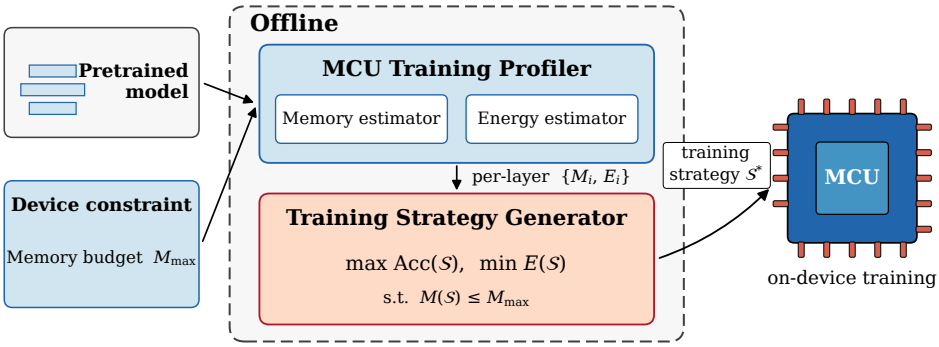


Figure 1: Overview of *e*ODL.

2 Background and Motivation

2.1 From inference to training on MCUs

MCUs can now support deep learning inference for applications such as image, audio, and gesture recognition, despite having extremely limited hardware resources, often with less than 320 KB of SRAM. This progress has been enabled by advances in TinyML, including model compression and searching techniques such as pruning, low-bit quantization and neural architecture search[5, 8, 10–12].

Training on MCUs, however, is significantly more challenging than inference [18]. Inference requires only a forward pass, whereas training additionally requires backpropagation and the storage of intermediate activations for gradient computation. For a network such as MCUNetV2 at 176×176 input, the activation memory needed by backpropagation is on the order of megabytes [7], while a typical STM32-class MCU has only 256–320 KB of SRAM—two orders of magnitude less. These transient activations can quickly exceed the SRAM budget of an MCU, making on-device training fundamentally harder to support [3].

2.2 Memory-aware training and the unaddressed energy cost

Recent systems close the gap between megabyte-scale activations and kilobyte-scale SRAM by restricting what is updated. They take several forms: last-layer-only updates [17], bias-only updates [1], sparse backward passes [13], layer- or tensor-level update selection [7], and rematerialization plus paging [15]. Despite their different mechanisms, all share a single objective: minimise the SRAM required by the backward pass.

Energy, however, is absent from all of these objectives. Per-epoch training energy is dominated by backward-pass MACs and the data movement they induce [18], yet none of the above methods model or control MACs. A training strategy that satisfies the SRAM budget may therefore still be energy-expensive—which matters directly for battery-powered IoT, where unnecessary computation shortens device lifetime.

2.3 From memory-feasible to energy-aware

Closing this energy gap requires two ingredients that existing MCU training stacks lack. First, an *energy model* that predicts training energy from software-level metrics, without measuring each configuration on hardware. Second, a *planner* that chooses the strategy jointly against memory, energy, and accuracy, rather than memory alone. The rest of the paper develops and evaluates these two components.

3 *e*ODL Framework

We propose *e*ODL, an energy-aware on-device learning (ODL) framework for MCUs. As shown in Figure 1, *e*ODL consists of two offline components: an MCU training profiler and a training strategy generator. Both run once offline on the host; the MCU only executes the resulting update plan, so the planner itself adds no runtime cost on the device. The profiler estimates the layer-wise memory footprint and training energy of each candidate update. Given these estimates, the strategy generator solves a constrained optimization problem that maximizes model accuracy while minimizing energy consumption under strict memory budgets. *e*ODL handles the two resources asymmetrically: the SRAM budget is enforced as a *hard* constraint, whereas energy enters the objective as a *soft* penalty weighted by γ (Section 3.2).

3.1 MCU training profiler

To assess whether a training configuration is deployable on an MCU, the profiler estimates both memory usage and energy cost. For memory estimation, *e*ODL adopts the memory predictor of prior sparse-update work [7]. Building on this foundation, *e*ODL further extends the profiler to model training energy by using Multiply-and-Accumulate (MAC) counts as an energy proxy.

3.1.1 Memory estimation. Following [7], the profiler estimates the additional memory introduced by training beyond inference. For a candidate update on layer l, we write

$$M_{\text{extra}}^{(l)} = M_{\text{act}}^{(l)} + M_{\text{param}}^{(l)}.$$

where $M_{\text{act}}^{(l)}$ is the additional activation memory retained for backpropagation and $M_{\text{param}}^{(l)}$ is the SRAM needed to store the trainable weights or biases. Bias gradients depend only on the output of the layer, so M_{act} is negligible. Weight gradients additionally require storing the input to the layer, which is much larger and dominates the memory cost.

3.1.2 Energy estimation. Direct energy measurement on MCUs is implementation-specific, making it unsuitable for rapid strategy search. We therefore use multiply-and-accumulate (MAC) counts as a proxy for training energy. For a training step, we write the

total computational cost as

$$K_{\text{train}} = K_{\text{fw}} + K_{\text{bp}} + K_{\text{up}},$$

where K_{fw} is the forward-pass MAC count, K_{bp} is the cost of propagating the error signal backward through the layer, and K_{up} is the additional cost of computing weight gradients on the updated channels only. For a convolutional layer we write $C_{\text{in}}, C_{\text{out}}, C_{\text{update}}$ for the input/output/trainable channel counts, $H_{\text{out}}, W_{\text{out}}$ for the output spatial dimensions, and K_H, K_W for the kernel sizes. The forward-pass cost of a standard convolution is

$$K_{\text{fw}} = H_{out} W_{out} \cdot K_H K_W \cdot C_{in} C_{out}.$$

The backpropagation cost is the mandatory overhead of propagating the error signal backward through the layer. For a standard convolution this amounts to a transposed convolution with the same kernel, so the MAC count equals the forward pass:

$$K_{bp} = H_{out} W_{out} \cdot K_H K_W \cdot C_{in} C_{out}.$$

The weight-update cost is optional and scales with the number of selected trainable channels:

$$K_{up} = (H_{out} W_{out} \cdot K_H K_W \cdot C_{in}) \cdot C_{update}.$$

Accordingly, we model training energy as a linear function of the total MAC count:

$$\hat{E} = a \cdot K_{\text{train}} + b,$$

where a and b are obtained by linear regression on the target platform (§4.3).

3.2 Training Strategy Generator

The Training Strategy Generator determines which layers and channels should be updated on the MCU. Its objective is to improve model accuracy while minimizing training energy under strict memory constraints.

3.2.1 Objective and Efficiency Score. To compare candidate updates, *e*ODL defines an Efficiency Score (η) that balances accuracy benefit against energy cost:

$$\eta = \Delta\text{acc} - \gamma \cdot \frac{E}{E_{\max}},$$

where Δacc denotes the estimated accuracy benefit of a candidate update, E is its profiler-estimated energy cost, $E_{\max}$ is the highest energy cost among all candidate updates (used to normalise the energy term to $[0, 1]$), and γ controls the penalty on energy consumption. A larger η indicates a more favorable benefit–cost trade-off.

The energy term E is obtained from the profiler (Section 3.1), while the accuracy-benefit term Δacc is described next.

3.2.2 Δacc. To estimate how much each candidate update improves accuracy, *e*ODL adopts the accuracy-benefit signal Δacc from prior work [7]. For each layer, Δacc is measured once on the host by training with only that update enabled and recording the resulting accuracy. The values are stored in a lookup table and reused by the strategy generator.

Δacc reflects two sources of accuracy gain: (i) extending bias updates to earlier layers, and (ii) updating a fraction $r \in \{1/8, 1/4, 1/2, 1\}$ of the weights at a given layer. As Figure 2 shows, bias-only adaptation saturates quickly, whereas sparse weight updates yield larger gains at later layers and higher update ratios. This motivates *e*ODL's layer- and channel-granularity update selection.

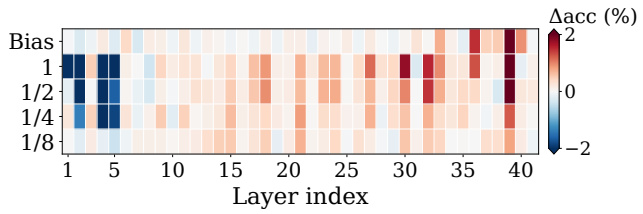


Figure 2: Marginal accuracy gain (Δacc) per layer when sparsely updating a pre-trained MCUNetV2 on Flowers.

3.2.3 Update selection. Given the efficiency score and the profiler estimates, the strategy generator must decide which layers and channels to update under the memory budget. The challenge is that layer updates are not independent: to update a deep layer, gradients must propagate through all layers above it, and this cost is paid even if none of those intermediate layers are updated. *e*ODL decomposes this coupled problem into two steps:

Step 1: Fix a candidate backpropagation depth k. Here, k denotes the number of last layers through which gradients are allowed to flow. For example, $k = 10$ means gradients flow back through the last 10 layers, so only those 10 layers can be updated. Once a candidate k is fixed, the backpropagation memory for these k layers is known, and Step 2 uses the remaining budget for weight updates.

Step 2: Choose what to update within those k layers. With k fixed, each layer's update decision is independent of the others, so the problem becomes a standard bounded knapsack. Based on this bounded-knapsack formulation, we design two alternative solvers: an exact DP solver when optimality is preferred, or a lightweight greedy solver when faster selection is needed.

DP solver (Algorithm 1). The algorithm builds a table $M[i][b]$ that records the best total Efficiency Score achievable by considering only the first i layers under a memory budget of b bytes. It fills the table one layer at a time, from the first reachable layer to the last. For each layer i and each possible budget b, the algorithm proceeds as follows. It first assumes the layer is skipped, inheriting the score from the previous row (Lines 6–7). Then it considers every candidate update variant v for that layer—updating 1/8, 1/4, 1/2, or all channels. For each variant, it checks whether the memory cost fits within the current budget b (Line 9). If so, it looks up the best score achievable by the previous $i-1$ layers using the remaining budget $b - \text{cost}(v)$, and adds the variant's own score $\eta(v)$ (Line 10). If this total exceeds the current best for $M[i][b]$, the table entry is updated (Lines 11–13). Once the table is fully filled, a traceback through the choice table C recovers which variant was selected at each layer, producing the final update strategy (Line 19).

Greedy solver (Algorithm 2). As a lightweight alternative, the greedy solver ranks all candidate updates by Efficiency Score (Line 3), then repeatedly picks the highest-scoring update that fits the remaining budget (Lines 2–4) until no feasible update remains. It does not guarantee optimality but runs with minimal overhead, making it suited for fast search or potential runtime adaptation on MCUs.

Algorithm 1 DP-based Update Selection for a Fixed Backpropagation Depth k

Require: Reachable layers $\mathcal{L}_k$, remaining memory budget B, candidate variants $\mathcal{V}_i$ for each layer i
Ensure: Update strategy S maximizing total Efficiency Score

```
/* initialize DP tables */
Let N ← |L_k|
Initialize DP table M[0...N][0...B] ← 0
Initialize choice table C[0...N][0...B] ← ∅
/* update DP states */
for i = 1 to N do
    for b = 0 to B do
        M[i][b] ← M[i − 1][b]
        C[i][b] ← skip
        for each variant v ∈ V_i do
            if cost_mem(v) ≤ b    ▷ memory cost in bytes then
                η_cand ← M[i − 1][b − cost_mem(v)] + η(v)
                if η_cand > M[i][b] then
                    M[i][b] ← η_cand
                    C[i][b] ← v
                end if
            end if
        end for
    end for
end for
/* recover selected strategy */
return Traceback(C, M)
```

Algorithm 2 Greedy Update Selection for a Fixed Backpropagation Depth k

Require: Reachable layers $\mathcal{L}_k$, remaining memory budget B, candidate variants $\mathcal{V}_i$ for each layer i
Ensure: Update strategy S

```
S ← ∅;  B_rem ← B
while ∃ v with cost_mem(v) ≤ B_rem do
    v* ← arg max_{v feasible} η(v)
    S ← S ∪ {v*};  B_rem ← B_rem − cost_mem(v*)
end while
return S
```

The full planner evaluates all candidate depths k and returns the strategy with the highest total Efficiency Score.

4 Evaluation

Following the experimental setting of study [7], we compare *e*ODL's generated training strategies against this state-of-the-art Tiny Training baseline under additional SRAM budgets of 50 kB and 100 kB, where the budget denotes the extra memory available for training beyond inference.

4.1 Experimental setup

We focus on a single representative setup: an ImageNet-pretrained MCUNetV2 backbone fine-tuned on the Flowers dataset, matching the baseline in [7]. Tiny Training maximizes estimated accuracy within the memory budget; it does not minimize memory usage once the constraint is satisfied. Similarly, *e*ODL treats memory as a hard constraint while optimizing the energy–accuracy trade-off. Cross-architecture and cross-dataset generalisation is left to future work (§6). To study how *e*ODL navigates the energy–accuracy trade-off, we sweep the energy penalty $\gamma \in \{0, 0.5, 1\}$: $\gamma = 0$ prioritises accuracy, $\gamma = 1$ prioritises energy savings, and $\gamma = 0.5$ balances the two. For each setting, we run both the Greedy and Tabular DP solvers.

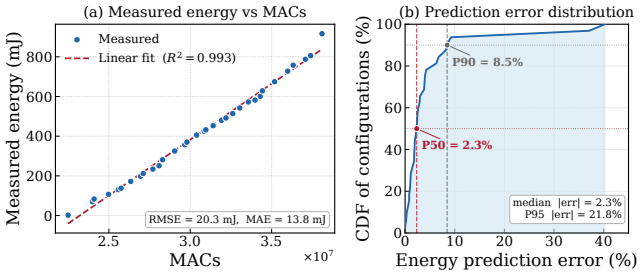


Figure 3: Our energy model. (a) Measured training energy vs. MAC count. (b) CDF of relative prediction error.

4.2 Methodology and metrics

We report three metrics to connect optimisation results with real deployment behaviour. *1) Accuracy:* top-1 accuracy after on-device training, evaluated with the Tiny Training Engine (TTE), Quantisation-Aware Scaling (QAS), SGD (no momentum, no weight decay), batch size 64, cosine learning-rate schedule (base LR 0.025, 5-epoch warmup), and 50 epochs, matching [7]. All results are from a single seed. *2) Computational cost (MACs):* total multiply-and-accumulate operations per training step, our hardware-agnostic energy proxy. *3) Energy consumption:* measured energy per epoch on an STM32F746G-Discovery board via an nRF Power Profiler Kit 2 [14].

4.3 Validating energy estimates

To verify that MACs can serve as a cheap proxy for on-device training energy, we measure the actual energy on the STM32 board across back-propagation depths from the last-1 to the last-33 layers and weight-update ratios from $\frac{1}{8}$ to full. These configurations cover the design space the strategy generator explores.

Figure 3(a) shows that the relationship is essentially linear: a two-parameter fit captures 99.3% of the variation ($R^2 = 0.993$), with a typical deviation of about 20 mJ over a measured range of 1.7 to 919 mJ. In practical terms, the MAC count alone is enough to predict on-device energy with high fidelity.

Figure 3(b) reports the relative prediction error per configuration. Half are predicted within 2.3% of their true energy, and 90% within 8.5%. The few outliers above 20% all correspond to very low-energy configurations (below 30 mJ), whose absolute error stays under 5 mJ—too small to reorder the candidate strategies.

In summary, MACs are a reliable energy proxy: the strategy generator can score thousands of candidates purely in software, without running each one on the MCU.

Method	$\gamma = 0$	$\gamma = 0.5$	$\gamma = 1$
50 kB Budget			
Tiny Training [7]		86.70 / 32.69	
Greedy	87.18 / 29.97	87.20 / 28.67	87.20 / 28.45
Tabular DP	87.33 / 29.67	86.65 / 28.41	87.04 / 28.38
100 kB Budget			
Tiny Training [7]		87.77 / 35.09	
Greedy	88.10 / 36.79	87.64 / 32.07	85.43 / 26.83
Tabular DP	87.43 / 37.60	88.01 / 28.86	87.04 / 28.38

Table 1: Accuracy (%) / MACs cost (million MACs) under two memory budgets.

4.4 Energy–accuracy trade-off

We compare *e*ODL against Tiny Training [7], which fits SRAM but does not penalise energy.

Accuracy and computational cost. Table 1 reports accuracy and MACs jointly. Since MACs are a validated energy proxy (§4.3, $R^2 = 0.993$), they allow us to compare the expected training energy cost of different strategies before measuring energy on hardware.

50 kB budget. At $\gamma = 0$, the planner does not explicitly penalise energy, so this setting serves as an accuracy-oriented reference point. Both solvers match or slightly exceed Tiny Training in accuracy (Greedy 87.18%, DP 87.33% vs. 86.70%), showing that *e*ODL preserves accuracy under the same SRAM budget. When γ increases to 1, energy is explicitly considered in the objective: accuracy remains almost unchanged (Greedy 87.20%, DP 87.04%), while MACs drop to 28.38M, corresponding to a 13.2% reduction over the baseline. Although DP finds the global optimum for the estimated Efficiency Score within the defined search space, it does not directly optimize the final measured accuracy. Since the accuracy-gain estimates are approximate, Greedy may occasionally match or slightly exceed DP in test accuracy.

100 kB budget. At $\gamma = 0$, the larger memory budget mainly provides a high-accuracy reference point. Greedy reaches the highest accuracy of 88.10%, while DP achieves 87.43%. At $\gamma = 0.5$, DP reaches 88.01% accuracy with only 28.86M MACs (17.7% reduction). At $\gamma = 1$, Greedy pushes MACs down to 26.83M (23.5% reduction) but accuracy drops to 85.43%, while DP maintains 87.04%/28.38M. This gap arises because Greedy commits to locally best updates without considering later layers, so it can over-allocate budget to early, low-accuracy layers. DP avoids this by jointly optimising across all reachable layers. In practice, this gap only matters when γ is high and the budget is large; under 50 kB, the two solvers produce nearly identical results.

An interesting case is DP at $\gamma = 1$, which produces the same result (87.04%/28.38M) under both budgets. This is because the strong energy penalty favours a compact strategy that uses less than 50 kB, so the extra memory at 100 kB is simply not needed. Across all settings, increasing γ consistently reduces MACs for both solvers, confirming that γ provides a predictable way to trade accuracy for energy.

Method	$\gamma = 0$	$\gamma = 0.5$	$\gamma = 1$
Tiny Training [7]		1.053	
Greedy	0.912	0.848	0.815
Tabular DP	0.905	0.814	0.815

Table 2: Measured on-device energy (J/epoch) under the 50 kB budget.

Overall, Table 1 shows that increasing γ generally reduces MACs, allowing *e*ODL to control the trade-off between training energy and accuracy under the same memory budget.

Hardware validation. To confirm that the MACs reductions in Table 1 translate to real energy savings, Table 2 reports measured energy on the STM32 board for the 50 kB setting. Even at $\gamma = 0$, energy drops from 1.053 J to 0.912 J (Greedy) and 0.905 J (DP), a 13–14% reduction with no accuracy loss. At $\gamma = 1$, both solvers converge to 0.815 J, a 22.6% reduction in energy. These results confirm that our planner yields tangible energy savings on the target MCU.

5 Conclusion

We present *e*ODL, an energy-aware framework for on-device training on microcontrollers. Unlike prior sparse-update methods that only fit the SRAM budget, *e*ODL explicitly trades energy against accuracy under the same hard memory constraint. It predicts on-device training energy from MAC counts with a simple linear model ($R^2 = 0.993$ on an STM32 board), and selects an update strategy with a lightweight greedy or dynamic-programming solver. In our experiments, *e*ODL matches state-of-the-art accuracy under tight memory budgets while cutting measured on-device energy by 22.6% at a 50 kB budget.

6 Discussion and Related Work

Generalisation across models and datasets. Our evaluation focuses on MCUNetV2 fine-tuned on Flowers, which rovides an initial validation of the core methodology but does not cover the full space of edge learning workloads. A natural next step is to test *e*ODL on other architectures as well as on additional datasets.

Limitations of MAC-Based Energy Estimator. Our current evaluation validates the MAC–energy correlation only on the STM32F746G-Discovery board. Since the mapping from MAC count to energy is platform-specific, deploying *e*ODL on a new MCU requires recalibrating and validating the estimator. On accelerator-equipped devices or architectures such as transformers, heterogeneous operators and data movement may weaken the correlation between MAC counts and energy. Such deployments may therefore require a more detailed, platform- and architecture-specific energy model. A systematic evaluation across different MCU platforms and architectures is left to future work.

Compatibility with Other Training Strategies. The current implementation of *e*ODL is built on the sparse layer- and channel-update design of Tiny Training and does not directly support other memory-aware training mechanisms. Extending *e*ODL to such approaches would require redefining the candidate strategy space and adapting the corresponding memory, energy, and accuracy estimators. We leave this extension to future work.

Limitations of the current evaluation. *e*ODL currently targets on-device fine-tuning for image recognition using pretrained

lightweight CNNs, such as MCUNetV2, on SRAM-constrained STM32-class MCUs. It assumes that partial training through sparse layer and channel updates is feasible within the device memory budget, and does not aim to support full training of large models. Extending *e*ODL to other architectures, devices, and application domains is left to future work.

Dependence on labeled data. The Δacc lookup table is constructed offline using representative labeled data and is specific to the model and task. Applying *e*ODL to a new domain therefore requires repeating this profiling step. When sufficient labeled data are unavailable, the current method cannot reliably estimate accuracy gains; developing label-efficient or transferable estimators is left to future work.

System-level energy impact. *e*ODL reduces the computation energy of on-device training, but does not optimize the energy consumed by cameras, sensors, radios, or other peripherals. Therefore, the overall system-level saving depends on how much of the total energy is attributable to model training. In applications dominated by peripheral or communication costs, the end-to-end benefit of *e*ODL may be limited.

Related Work: ODL on MCUs. On-device learning has also been explored on more capable MPU-class platforms running Linux, such as ElasticTrainer [2] dynamically select which tensors to update at runtime, and TinyTrain [4] performs task-adaptive layer/channel selection using a Fisher-based multi-objective criterion over update importance and memory. MCU-based ODL, however, faces fundamentally tighter constraints, including kilobyte-scale SRAM, the absence of an operating system, and bare-metal execution. Recent work on ODL for MCUs [3, 18] shares the goal of fitting backpropagation into this limited SRAM but differs in *how* updates are restricted: bias-only adaptation [1], sparse backward passes [13], layer- or tensor-granularity update selection [4, 7], and rematerialisation with paging [15]. All four families share a single objective—minimise the SRAM footprint of the backward pass.

Acknowledgment: This work is partly supported by the NWO *LuxSenz* project, funded through the TOP Grant (Module 1) under project number 612.001.854.

References

[1] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce memory, not parameters for efficient on-device learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*.

[2] Kai Huang, Boyuan Yang, and Wei Gao. 2023. Elastictrainer: Speeding up on-device training with runtime elastic tensor selection. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services*. 56–69.

[3] Jacob Huckelberry, Yuke Zhang, Allison Sansone, James Mickens, Peter A. Beerel, and Vijay Janapa Reddi. 2024. TinyML Security: Exploring Vulnerabilities in Resource-Constrained Machine Learning Systems. arXiv:2411.07114

[4] Young D. Kwon, Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane, and Cecilia Mascolo. 2024. TinyTrain: Resource-aware task-adaptive sparse training of DNNs at the data-scarce edge. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*.

[5] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. 2021. μnas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*.

[6] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny deep learning on IoT devices. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*. 11711–11722.

[7] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256KB memory. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*.

[8] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, and Song Han. 2023. Tiny Machine Learning: Progress and Futures [Feature]. *IEEE Circuits and Systems Magazine* (2023).

[9] Hao Liu, Qing Wang, and Marco Zuniga. 2025. SolarML: Optimizing Sensing and Inference for Solar-Powered TinyML Platforms. In *Proceedings of the Design, Automation & Test in Europe Conference (DATE)*.

[10] Hao Liu, Qing Wang, and Marco Zuniga. 2026. InstMeter: An Instruction-Level Method to Predict Energy and Latency of DL Model Inference on MCUs. *arXiv preprint arXiv:2603.04134* (2026).

[11] Gaurav Menghani. 2023. Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better. *Comput. Surveys* (2023).

[12] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A White Paper on Neural Network Quantization.

[13] Francesco Paissan, Davide Nadalini, Manuele Rusci, Alberto Ancilotto, Francesco Conti, Luca Benini, and Elisabetta Farella. 2024. Structured Sparse Backpropagation for Lightweight On-Device Continual Learning on Microcontroller Units. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*.

[14] Chanho Park, Do Yun Park, Haohui Zhang, Da Som Yang, Catherine R Redden, Seonggwang Yoo, Tae Wan Park, Mingyu Zhou, Jae-Young Yoo, Youngmin Sim, et al. 2025. Adaptive electronics for photovoltaic, photoluminescent and photometric methods in power harvesting for wireless wearable sensors. *Nature communications* (2025).

[15] 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 *Proceedings of the International Conference on Machine Learning (ICML)*.

[16] Visal Rajapakse, Ishan Karunanayake, and Nadeem Ahmed. 2023. Intelligence at the extreme edge: A survey on reformable TinyML. *Comput. Surveys* (2023).

[17] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. 2021. TinyOL: TinyML with Online-Learning on Microcontrollers. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*.

[18] Shuai Zhu, Thiemo Voigt, Fatemeh Rahimian, and Jeonggil Ko. 2024. On-device Training: A First Overview on Existing Systems. *ACM Transactions on Sensor Networks* (2024).