On-Device Training at the Extreme Edge

Young D. Kwon
University of Cambridge
United Kingdom
ydk21@cam.ac.uk

ABSTRACT

On-device training is critical for user privacy and customisation. However, deploying this on IoT devices and microcontrollers presents significant challenges due to restricted memory, limited computational resources, and insufficient labelled user data. Existing approaches either ignore data scarcity problems, demand prohibitively long training time (*e.g.* a few hours), or suffer from severe accuracy degradation (≥10%). This paper introduces TinyTrain, an on-device training framework that significantly reduces training time by selectively updating parts of the model while explicitly addressing data scarcity. TinyTrain employs a task-adaptive sparse-update strategy that intelligently selects layers and channels using a multi-objective optimisation criterion. This criterion simultaneously considers available user data, memory constraints, and computational capabilities of the deployment device, enabling superior performance on novel tasks with minimal resource consumption. TinyTrain exceeds standard full-network training by 3.6-5.0% in accuracy, reducing backward-pass memory and computational costs by up to 1,098× and 7.68×, respectively. When deployed on commonly used edge devices, TinyTrain achieves 9.5× faster and 3.5× more energy efficient training over existing methods, plus 2.23× reduced memory usage versus SOTA techniques, all while operating within the 1 MB memory envelope of MCU-grade platforms.

KEYWORDS

Tiny Machine Learning, On-device Training, Cross-Domain Few-Shot Learning, Personalisation, Microcontrollers

1 Introduction

Motivation: On-device training of deep neural networks (DNNs) on edge devices has the potential to enable diverse real-world applications to *dynamically adapt* to new tasks and different (*i.e.* cross-domain/out-of-domain) data distributions from users (*e.g.* personalisation) [1], without jeopardising privacy over sensitive data (*e.g.* healthcare) [9].

Challenges: Despite its benefits, several challenges hinder the broader adoption of on-device training. Firstly, labelled user data are neither abundant nor readily available in real-world IoT applications [2]. Secondly, tiny edge devices are characterised by severely limited memory [4]. With the *forward* and *backward passes* of DNN training being significantly memory-hungry. Even network architectures that are tailored to microcontroller units (MCUs), such as MCUNet [7], require almost 1 GB of training-time memory (see Table 2), which far exceeds the RAM size of the widely used embedded devices, such as Raspberry Pi Zero 2 (512 MB), and commodity MCUs (1 MB). Lastly, on-device training is limited by the constrained processing capabilities of edge devices; with training requiring at least 3× more computation (*i.e.* multiply-accumulate (MAC) count) than inference. This places an excessive burden on tiny edge devices that host less powerful CPUs, compared to the server-grade CPUs or GPUs.

Prior Works: Recently, many on-device training works have been proposed in the literature. For instance, (1) *fine-tuning* only the last layer [6] leads to considerable accuracy loss (>10%) that far exceeds the typical drop tolerance. (2) memory-saving techniques by means of *recomputation* that trade off performing more computations for lower memory usage incur significant computation overhead, further increasing the already excessive on-device training time. (3) *sparse-update* methods selectively update only a subset of layers and channels during on-device training, effectively reducing both the memory and computation load. However, the performance of this approach drops dramatically (up to 7.7%) when applied at the extreme edge where data availability is low. Moreover, it requires running *a few thousands of* computationally heavy search [8] processes on powerful GPUs to identify important layers/channels for each target dataset, unable to adapt to the properties of the user data on the fly.

2 Method

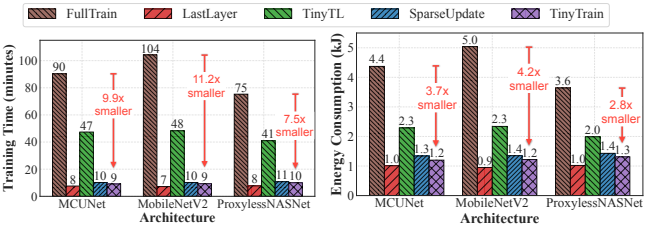
To address the aforementioned challenges and limitations, we present *TinyTrain* [5], the first approach that fully enables data-, compute-, and memory-efficient on-device training on tiny edge devices. Similar to *SparseUpdate*, instead of updating the whole model, TinyTrain leverages a sparse-update method to selectively train only part of the architecture. However, TinyTrain departs from the static configuration of the sparse-update policy, *i.e.* the subset of layers and channels to be fine-tuned being fixed, and introduces *task-adaptive sparse update*. Specifically, at run time, TinyTrain dynamically adapts the sparse update policy based on both the properties of the user data, and the memory and processing capacity of the target device. Moreover, we introduce a new multi-objective criterion to guide the layer/channel selection process that captures both the importance of channels and their computational and memory cost. Contrary to *SparseUpdate*'s server-based evolutionary search, our criterion can be efficiently run on very constrained edge devices and enables us to adapt the layer/channel selection in a task-adaptive manner, leading to better on-device adaptation and higher accuracy. To further address the drawbacks of *SparseUpdate* under scarce data availability, TinyTrain enhances the conventional on-device training pipeline by means of a few-shot learning (FSL) pre-training scheme; this step meta-learns a reasonable global representation that allows on-device training to be sample-efficient and reach high accuracy despite the limited user data.

3 Evaluation

We comprehensively evaluate TinyTrain and the baselines based on three DNN architectures and four cross-domain datasets by conducting 200 trials for each dataset. Table 1 shows that TinyTrain

Table 1: Top-1 accuracy results of TinyTrain and the baselines. TinyTrain achieves the highest accuracy.

Model	Method	Traffic	Omniglot	Aircraft	Flower	Avg.
MCUNet	None	35.5	42.3	42.1	73.8	48.4
	FullTrain	**82.0**	72.7	75.3	90.7	80.2
	LastLayer	55.3	47.5	56.7	83.9	60.8
	TinyTL	78.9	73.6	74.4	88.6	78.9
	SparseUpdate	72.8	67.4	69.0	88.3	74.4
	TinyTrain (Ours)	79.3	**73.8**	**78.8**	**93.3**	**81.3**
Mobile NetV2	None	39.9	44.4	48.4	81.5	53.5
	FullTrain	75.5	**69.1**	68.9	84.4	74.5
	LastLayer	58.2	55.1	59.6	86.3	64.8
	TinyTL	71.3	69.0	68.1	85.9	73.6
	SparseUpdate	77.3	**69.1**	72.4	87.3	76.5
	TinyTrain (Ours)	**77.4**	68.1	**74.1**	**91.6**	**77.8**
Proxyless NASNet	None	42.6	50.5	41.4	80.5	53.8
	FullTrain	78.4	73.3	71.4	86.3	77.3
	LastLayer	57.1	58.8	52.7	85.5	63.5
	TinyTL	72.5	**73.6**	70.3	86.2	75.7
	SparseUpdate	76.0	72.4	71.2	87.8	76.8
	TinyTrain (Ours)	**79.0**	71.9	**76.7**	**92.7**	**80.1**



(b) Training Time **(c) Energy Consumption**

Figure 1: The end-to-end latency and energy consumption of the on-device training methods across three architectures.

achieves the highest accuracy, with gains of 3.6-5.0% over fine-tuning the entire DNN, denoted by FullTrain. On the compute front, Table 2 shows that TinyTrain significantly reduces the memory footprint and computation required for backward pass by up to 1,098× and 7.68×, respectively. TinyTrain further outperforms the SOTA *SparseUpdate* method in all aspects, yielding: (a) 2.6-7.7% accuracy gain across four datasets; (b) 1.59-2.23× reduction in memory; and (c) 1.52-1.82× lower computation costs. Finally, we demonstrate how our work makes important steps towards efficient training on very constrained edge devices by deploying TinyTrain on a Pi Zero 2 embedded device and achieving an end-to-end on-device training in 10 minutes, an order of magnitude speedup over the two-hour training of FullTrain (see Figure 1). For TinyTrain, we also include the time and energy to dynamically select layers/channels for our task-adaptive sparse update, which takes around 20-35 seconds, accounting for only 3.4-3.8% of the total training time of TinyTrain. We further compare TinyTrain and SOTA on two embedded devices, Pi Zero 2 and Jetson Nano, showing TinyTrain yields 1.08-1.12x and 1.3-1.7x faster on-device training than SOTA, respectively.

Table 2: The memory footprint and computation costs for a backward pass between TinyTrain and the baselines.

Model	Method	Memory	Ratio	Compute	Ratio
MCUNet	FullTrain	906 MB	1,013×	44.9M	6.89×
	LastLayer	2.03 MB	2.27×	1.57M	0.23×
	TinyTL	542 MB	606×	26.4M	4.05×
	SparseUpdate	1.43 MB	**1.59**×	11.9M	**1.82**×
	TinyTrain (Ours)	**0.89 MB**	1×	**6.51M**	1×
Mobile NetV2	FullTrain	1,049 MB	987×	34.9M	7.12×
	LastLayer	1.64 MB	1.54×	0.80M	0.16×
	TinyTL	587 MB	552×	16.4M	3.35×
	SparseUpdate	2.08 MB	**1.96**×	8.10M	**1.65**×
	TinyTrain (Ours)	**1.06 MB**	1×	**4.90M**	1×
Proxyless NASNet	FullTrain	857 MB	**1,098**×	38.4M	**7.68**×
	LastLayer	1.06 MB	1.36×	0.59M	0.12×
	TinyTL	541 MB	692×	17.8M	3.57×
	SparseUpdate	1.74 MB	**2.23**×	7.60M	**1.52**×
	TinyTrain (Ours)	**0.78 MB**	1×	**5.00M**	1×

4 Conclusion

We have developed the first realistic on-device training framework, TinyTrain, solving practical challenges in terms of data, memory, and compute constraints for extreme edge devices. TinyTrain opens the door, for the first time, to on device training with acceptable performance on a variety of low resources devices such as MCUs embedded in IoT frameworks.

Limitations & Societal Impacts. Our evaluation is currently limited to CNN-based architectures on vision tasks. As future work, we hope to extend TinyTrain to different architectures (*e.g.* Transformers, RNNs) and applications (*e.g.* audio or biological data [3]). In addition, while on-device training avoids the excessive electricity consumption and carbon emissions of centralised training, it has thus far been a significantly draining process for the battery life of edge devices. Nevertheless, TinyTrain paves the way towards alleviating this issue as demonstrated in Figure 1c.

REFERENCES

[1] Jagmohan Chauhan, Young D. Kwon, Pan Hui, and Cecilia Mascolo. 2020. ContAuth: Continual Learning Framework for Behavioral-Based User Authentication. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4, 4, Article 122 (Dec. 2020).

[2] Young D. Kwon, Jagmohan Chauhan, Hong Jia, Stylianos I. Venieris, and Cecilia Mascolo. 2023. LifeLearner: Hardware-Aware Meta Continual Learning System for Embedded Computing Platforms. In *Proc. of SenSys '23.*

[3] Young D Kwon, Jagmohan Chauhan, Abhishek Kumar, Pan Hui, and Cecilia Mascolo. 2021. Exploring System Performance of Continual Learning for Mobile and Embedded Sensing Applications. In *Proc. of SEC '21.*

[4] Young D. Kwon, Jagmohan Chauhan, and Cecilia Mascolo. 2022. YONO: Modeling Multiple Heterogeneous Neural Networks on Microcontrollers. In *proc. of IPSN '22.* 285–297. https://doi.org/10.1109/IPSN54338.2022.00030

[5] Young D. Kwon, Rui Li, Stylianos I. Venieris, Jagmohan Chauhan, Nicholas D. Lane, and Cecilia Mascolo. 2023. TinyTrain: Deep Neural Network Training at the Extreme Edge. arXiv:2307.09988 [cs.LG]

[6] Seulki Lee and Shahriar Nirjon. 2020. Learning in the Wild: When, How, and What to Learn for On-Device Dataset Adaptation. In *Proc. of AIChallengeIoT '20.*

[7] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. 2020. MCUNet: Tiny Deep Learning on IoT Devices. In *Proc. of NeurIPS '20.*

[8] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-Device Training Under 256KB Memory. In *Proc. of NeurIPS '22.*

[9] Nhat Pham, Hong Jia, Minh Tran, Tuan Dinh, Nam Bui, Young Kwon, Dong Ma, Phuc Nguyen, Cecilia Mascolo, and Tam Vu. 2022. PROS: An Efficient Pattern-Driven Compressive Sensing Framework for Low-Power Biopotential-Based Wearables with on-Chip Intelligence. In *Proc. of MobiCom '22.*