Comparison of Two Microcontroller Boards for On-Device Model Training in a Keyword Spotting Task

Nil Llisterri Giménez*, Felix Freitag*, JunKyu Lee†, Hans Vandierendonck†
Department of Computer Architecture. Technical University of Catalunya. Barcelona, Spain
nil.llisterri@estudiantat.upc.edu, felix@ac.upc.edu
† The Institute of Electronics, Communications and Information Technology. Queen's University Belfast. UK
junkyu.lee, h.vandierendonck@qub.ac.uk

2022 11th Mediterranean Conference on Embedded Computing (MECO) | 978-1-6654-6828-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/MECO55406.2022.9797171

***Abstract*—Machine learning applications on resource-constrained devices such as microcontroller units often use models trained externally on more powerful devices. This approach, however, limits a later adaptation of the machine learning model in the device to changing data. Differently, on-device training allows the model to be updated for new datasets, but the training process needs to take into account the resource limitations of the device. This paper compares on-device training performance for a keyword spotting task using two popular microcontroller boards, Arduino Nano 33 BLE Sense and Arduino Portenta H7, in terms of inference accuracy, training latency, and current consumption. We use feedforward neural networks having a single hidden layer for models. The inference accuracy has been significantly improved using the Portenta H7 board by employing more neurons fitted to its memory budget, compared to the Nano board. With a neural network having 25 neurons for a hidden layer, the 5.0 $\times$ inference and 4.2 $\times$ training speedups are achieved using the Arduino Portenta H7 board, compared to the Arduino Nano 33 BLE Sense. While the memory of the Arduino Nano 33 BLE Sense is capable to train a neural network for the keyword spotting task, the Arduino Portenta H7 gives new possibilities for exploring more complex models for more complex problems thanks to a larger memory budget and adapting a model to new data in lower latency.**

***Index Terms*—TinyML, machine learning, IoT.**

I. INTRODUCTION

Machine learning (ML) is progressively being implemented in ever smaller computing devices, creating new infrastructures at the network edge. Recently, ML applications are being brought into microcontroller boards, which has been coined as TinyML [1]. TinyML mitigates concerns of privacy, inference speed, and energy consumption. For example, performing the inference tasks locally on TinyML edge devices removes the data communication between the edges and a cloud, minimizing such concerns [2]. TinyML has opened a wide range of applications where data is analyzed where it is produced and with low energy usage. Wildlife conservation is a successful area of TinyML[1]. Industrial applications include structural health monitoring, where TinyML can be applied to detect anomalies [3].

However, most of today's TinyML solutions use machine learning models that have previously been trained on powerful devices with large datasets, doing then only inference on the embedded device. One of the limitations is that it is not possible to adapt the model on the device to changing data.

There have been proposals to train models on the embedded devices, like the works of [2], [4], [5], which combine transfer learning with on-device training. In such on-device training, a neural network is stored on the device and with each new sample a specific layer of the model is incrementally trained.

In this paper we explore the training of a neural network on microcontrollers for a Keyword Spotting (KWS) task with two popular boards, the Arduino Nano 33 BLE Sense and the Arduino Portenta H7. We compare the performance of both boards for doing the KWS training task and analyze the possibilities each hardware raises.

II. RELATED WORK

Cai et al. [6] presented a memory efficient on-device learning, named Tiny-Transfer-Learning (TinyTL). The idea of TinyTL was to learn only bias modules for transfer learning while fixing the weights to save the storage of intermediate activations in backpropagation. Disabato et al. [7] discussed challenges and opportunities of on-device training using embedded systems and IoT devices with examples of an incremental learning algorithm based on transfer learning and k-nearest neighbor algorithm. Ren et al. [2] proposed an online learning on Micro-Controller Units (MCUs) that is able to be adaptive to dynamically changing data (e.g., time series data). Nil et al. [5] performed on-device training on an Arduino Nano 33 BLE Sense board to explore how different aspects of the federated learning mechanisms affected the neural network (NN) training process.

Many researches [8]–[10] leveraged neural architecture search variants [11] to generate compact networks fitted to resource constrained devices such as MCUs and mobile devices according to training data complexity, quality and quantity. Fedorov et al. [12] claimed that the traditionally Convolutionary Neural

[1]Wildlife tracker challenge winners use nordic-powered wireless connectivity. https://www.nordicsemi.com/News/2021/01/Wildlife-tracker-challenge-winners-use-Nordic-powered-wireless-connectivity

978-1-6654-6828-2/22/$31.00 ©2022 IEEE

Network (CNN) architecture is not suitable for MCUs due to restricted memory budget, and proposed a search method to seek compact sparse neural network architecture that can be deployed on MCUs. Lee et al. [13] performed a survey on model-, arithmetic-, and implementation-level optimization for deep learning that can produce compact smaller networks for resource constrained devices.

In this paper, we analyze the on-device training performance in terms of accuracy and hardware usage on both the Arduino Portenta and Nano boards, using as a case a KWS applications and compare them. We also evaluate the performance of the dual-core architecture of the Arduino Portenta board.

III. CASE STUDY: KEY WORD SPOTTING MACHINE LEARNING TASK

We created a dataset[2] for three voiced keywords, each of 180 samples, being a total of 540 samples. These keywords were taken using the Arduino Nano 33 BLE Sense built-in microphone. The audio was sampled at 16kHz and recorded for 1 second. For this audio signal 13 Mel-frequency cepstral coefficients (MFCC) were computed for 50 frames (i.e., each frame consists of signal samples for 20ms), obtaining thus 650 values.

With this data we train in the microcontroller a three layer feedforward neural network, which is stored in the device's RAM. The input layer of the network has as many nodes as MFCC coefficient are extracted, i.e 650. It is connected to a fully connected hidden layer, whose size we change throughout the experimentation according to available memory footprint on each for an Arduino Nano and an Arduino Portenta H7 board. Finally, the output layer has 3 nodes, corresponding to the number of keywords to recognize. For the training of the neural network for the KWS task we apply the implementation of [5].

IV. EXPERIMENTATION

For the experimentation we use the Arduino Nano 33 BLE Sense, which contains a microcontroller based on the Cortex M4F running at 64Mhz with 256kB SRAM. We also use the Arduino Portenta H7, which contains two cores, a Cortex M7 running at 480 MHz and a Cortex M4 running at 240 MHz. We observed that from the 1 MB of SRAM of the board, 523kB was available for the Cortex M7 and 295kB for the Cortex M4. Figure 1 shows the two types of boards we used for the experimentation.

The experimentation is divided in two parts: In section IV-A we compare the application deployed on the two boards where in the Portenta H7 we use only the M7. In section IV-B we analyse the application deployed on the Portenta H7 when the M7 and M4 microcontrollers are used.

A. Comparison of the KWS application on the Arduino Nano 33 BLE Sense and Arduino Portenta H7

Memory usage: We experimentally determined the maximum size of the hidden layer allowed by the memory constraints of the two microcontroller boards for the given feedforward neural network. For the Arduino Nano we found that this was a layer of 25 hidden neurons considering the required number of weights and activations while for the Arduino Portenta up to 70 hidden neurons were the maximum possible.

[2]The dataset and the application code is publicly accessible at https://github.com/NilLlisterri/Portenta-Dual-Core

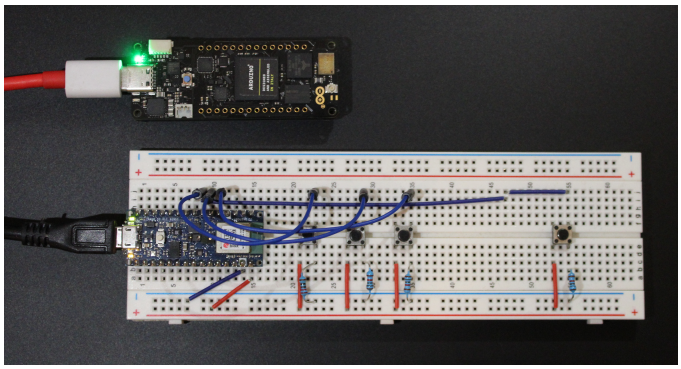


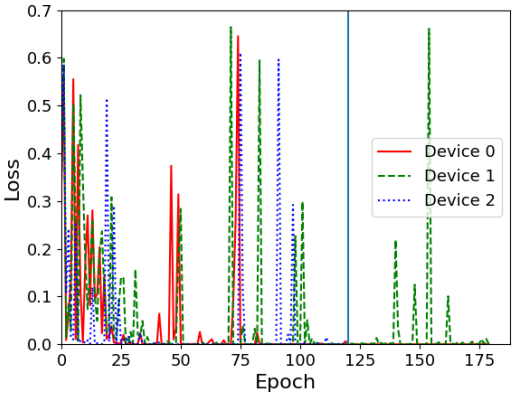
Figure 1: Arduino Nano 33 BLE Sense and Arduino Portenta H7 with Vision Shield.

Analyzing the components used by the KWS application, the audio buffer consumes 32kB, the neural network weights in the Arduino Nano and Arduino Portenta consume around 65kB and 182kB, repectively, and the same amount of memory is consumed for the gradients of the backpropagation training. Compiler information indicated that around 48kB are used for other memory allocations of code and libraries.

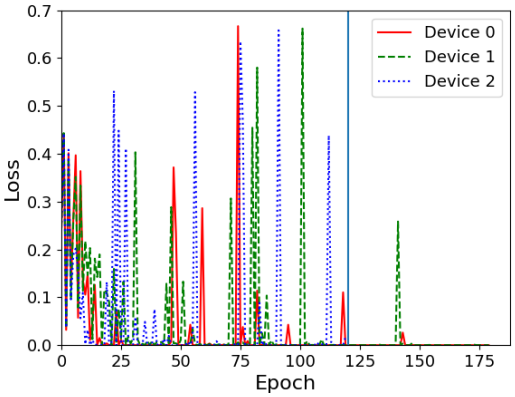
Model training performance: We flash the same KWS application code to the two types of boards but with different hidden layer size, corresponding to the maximum allowed by the RAM of each board. We split our dataset in three parts of 180 samples, each with an equal number of the three keywords. Then we conduct with three devices of each board on-device training. The first 120 samples train the neural network on the board, the last 60 samples are used for inference. Figure 2 describes the model loss in training (i.e., training error from the epoch number 1 to 120) and inference (i.e., inference error from the epoch number 121 to 180) measured at each epoch. We update the weights per sample (i.e., each epoch utilizes one sample.). The model loss is measured using the MSE. Visual inspection shows that although the main training effect happens in the first tens of training samples, there are occasionally samples which later still produce a high loss. We stop the training when the number of epochs reaches 120.

Model inference performance: In inference task using the 60 samples (i.e., epoch number from 121 to 180 in Figure 2), the loss is very low in both networks, which turns into a very high accuracy achieved when doing the inference for keyword classification. We observe that the inference accuracy of a NN using more neurons on Ardunio Portenta board is significantly higher than Nano board (refer to Device 1 setting for each board).

Computing time: We compare the computing time when we use the same machine learning model on both boards, i.e the feedforward neural network with a hidden layer of 25 neurons.



(a) Arduino Nano board: training on three boards a neural network with hidden layer of 25 neurons.



(b) Arduino Portenta board: training on three boards a neural network with hidden layer of 70 neurons.

Figure 2: Loss vs. epochs during training on both types of board the neural network with different hidden layer size.

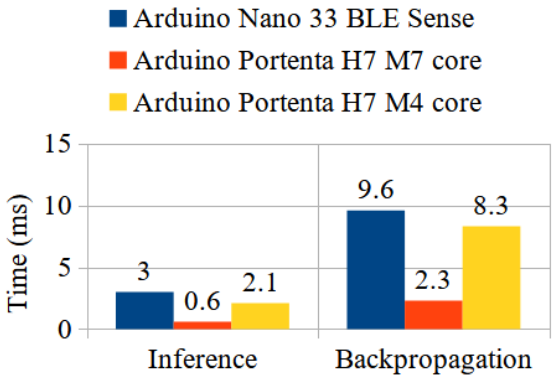


Figure 3: Inference and backpropagation times for both boards and M7 and M4 cores in Adruino Portenta, respectively.

Figure 3 shows the time we measured for doing the training with backpropagation and the inference on both boards. It can be seen that with the M7 core the Arduino Portenta H7 performs between 4-5 times faster.

B. Analysis of the KWS application on the dual cores of Arduino Portanta H7

The dual-core architecture of the Arduino Portenta H7 allows one core to manage the audio recording, while the other can manage the tasks related to the NN. The first approach was to manage the NN on the more powerful M7 core, and use the M4 core to capture the audio and prepare it so the M7 can consume it. Unfortunately, the audio capturing library was only designed for the M7 core [3], so the tasks had to be shifted. This means that the number of neurons in a hidden layer is restricted to 25 on the M4 core at most for our dual-core implementation version.

[3]GitHub issue:https://github.com/arduino/ArduinoCore-mbed/issues/416

When the audio is recorded by one core, the captured data, processed from the audio capturing library, has to be accessible for the other core in order to perform the ML tasks. Both cores can communicate via Remote Procedure Call (RPC) and using a shared memory region. We notice that the recorded signal occupies 32kB of memory, and transmitting this information via RPC would be slow, while storing it in a shared memory region and notifying the other core via an RPC is practically instantaneous. Therefore, it would be beneficial to store the recorded signal (e.g., pre-processed data) in the shared memory region first and utilize RPC to inform the other core of this event later. This implementation method will be useful particularly for latency-critical time series applications that utilise online learning method. Using both cores affects the power consumption of the device. When only the M7 core is used, the idle current consumption is 84mA. When both are used, the consumption increases by 10mA. When recording, both in dual-core and single-core mode, the consumption increases by 5mA. Differently, when the NN is trained (i.e. the forward and backward pass), using different cores has a significant impact on the consumption. When the NN is trained on the M7 core, the consumption increases by 20mA, while on the M4 it only increases by 6mA.

In figure 4 we can observe the power consumption while performing different tasks in the dual-core mode. In this mode, when both cores are idle, 94mA are used. When the M7 core starts recording the consumption rises to 100mA. Then, when the M4 starts training the network, the consumption increases to 111mA. When the device communicates with the server, for example when the testing samples are sent, the maximum consumption of 127mA is reached. This implies that since the data transfer cost is a main factor for current (or power) consumption for on-device training on Arduino Portenta H7 board, it would be recommended to compress the data prior to storing them to memory in the board to save energy consumption.

Figure 5 depicts the current evolution in a record-and-train loop. The device is idle for 6 seconds, and after that the

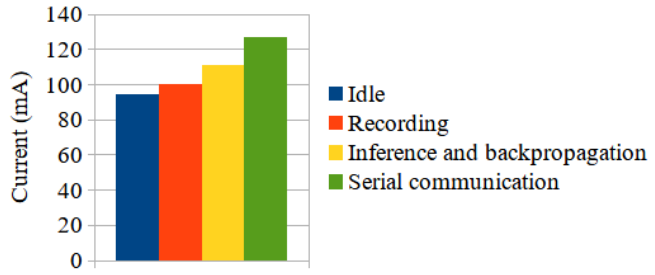


Figure 4: Arduino Portenta H7 current consumption per activity.

Table I: Disribution of KWS application code on the two cores.

M7 Core	M4 Core
Communicate with the server	Preprocess the audio (MFCC)
Record audio	Inference
Manage shared buffer	Backpropagation
Proxy the M4 serial communication	

recording of a one-second sample starts. Then, the NN will be trained with the captured audio. The current recordings are taken approximately every 1,3s. As the inference and backpropagation times are very small, about 10ms, the current spike we can observe is mainly due to the recording process.

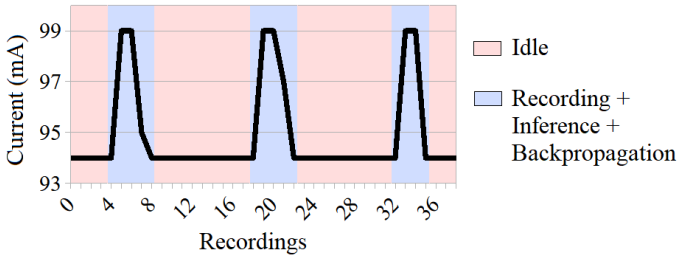


Figure 5: Arduino Portenta H7 current evolution when periodically recording and training.

In the dual-core setup the tasks of the KWS applications are distributed as pictured in Table I. The M7 core performs various types of tasks, like interacting with the server and preparing the audio samples for the M4 core. The M4 core is focused on the ML aspect only; preprocessing the audio signal, running the inference and the backpropagation.

V. CONCLUDING REMARKS AND OUTLOOK

The memory of the Arduino Nano 33 BLE Sense was sufficient for the on-device training of a simple KWS task. Upgrading the hardware to an Arduino Portenta H7 with the increased SRAM and computational power can open new possibilities for training more complex neural networks such as CNNs for KWS tasks which was proposed in [14]. The Arduino Portenta H7 dual-processor architecture could be further exploited for dividing the machine learning application components on the two cores, according to the application's requirements for speed and memory. On-device training on MCUs becomes more significant as the compute capability of MCUs is improving and the privacy appears as one of the main concerns in the artificial intelligence community. Our work will help readers from the artificial intelligence community understand the feasibility of on-device training at the intersection between the problem complexity and the compute resource budgets.

ACKNOWLEDGMENT

This work was partially supported by the Spanish Government under contracts PID2019-106774RB-C21, PCI2019-111850-2 (DiPET CHIST-ERA CHIST-ERA-SDCDN-002), PCI2019-111851-2 (LeadingEdge CHIST-ERA), and UK Engineering and Physical Sciences Research Council (EP/T022345/1).

REFERENCES

[1] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking tinyml systems: Challenges and direction," 2021.

[2] H. Ren, D. Anicic, and T. A. Runkler, "Tinyol: Tinyml with online-learning on microcontrollers," *CoRR*, vol. abs/2103.08295, 2021. [Online]. Available: https://arxiv.org/abs/2103.08295

[3] C. Arcadius Tokognon, B. Gao, G. Y. Tian, and Y. Yan, "Structural health monitoring framework based on internet of things: A survey," *IEEE Internet of Things Journal*, vol. 4, no. 3, pp. 619–635, 2017.

[4] K. Kopparapu and E. Lin, "Tinyfedtl: Federated transfer learning on tiny devices," 2021.

[5] N. Llisterri Giménez, M. Monfort Grau, R. Pueyo Centelles, and F. Freitag, "On-device training of machine learning models on microcontrollers with federated learning," *Electronics*, vol. 11, no. 4, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/4/573

[6] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 11 285–11 297.

[7] S. Disabato and M. Roveri, "Incremental on-device tiny machine learning," in *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, ser. AIChallengeIoT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 7–13. [Online]. Available: https://doi.org/10.1145/3417313.3429378

[8] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 6105–6114. [Online]. Available: http://proceedings.mlr.press/v97/tan19a.html

[10] "Neuton.ai," https://neuton.ai, accessed: 09-March-2021.

[11] B. Zoph and Q. Le, "Neural architecture search with reinforcement learning," in *ICLR '17: International Conference on Learning Representations*, 2017.

[12] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[13] J. Lee, L. Mukhanov, A. S. Molahosseini, U. Minhas, Y. Hua, J. M. del Rincon, K. Dichev, C.-H. Hong, and H. Vandierendonck, "Resource-efficient deep learning: A survey on model-, arithmetic-, and implementation-level techniques," 2021.

[14] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech 2015*, 2015, pp. 1478–1482.