TinyML Benchmark: Executing Fully Connected Neural Networks on Commodity Microcontrollers

Bharath Sudharsan*, Simone Salerno†, Duc-Duy Nguyen*, Muhammad Yahya*, Abdul Wahid*
Piyush Yadav*, John G. Breslin*, Muhammad Intizar Ali‡
*Data Science Institute, NUI Galway, Ireland
{bharath.sudharsan, ducduy.nguyen, muhammad.yahya, abdul.wahid, piyush.yadav, john.breslin}@insight-centre.org
†eloquentarduino@gmail.com
‡School of Electronic Engineering, Dublin City University, Ireland, ali.intizar@dcu.ie

2021 IEEE 7th World Forum on Internet of Things (WF-IoT) | 978-1-6654-4431-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/WF-IoT51360.2021.9595024

***Abstract*—Recent advancements in the field of ultra-low-power machine learning (TinyML) promises to unlock an entirely new class of edge applications. However, continued progress is restrained by the lack of benchmarking Machine Learning (ML) models on TinyML hardware, which is fundamental to this field reaching maturity. In this paper, we designed 3 types of fully connected Neural Networks (NNs), trained each NN using 10 datasets (produces 30 NNs), and present the benchmark by reporting the onboard model performance on 7 popular MCU-boards (similar boards are used to design TinyML hardware). We open-sourced and made the complete benchmark results freely available online [1] to enable the TinyML community researchers and developers to systematically compare, evaluate, and improve various aspects during the design phase of ML-powered IoT hardware.**

***Index Terms*—IoT Devices, Offline Inference, Edge Intelligence.**

I. Introduction

TinyML aims to bring ML inference on ultra-low-power IoT devices, typically under a milliWatt, thereby breaking the traditional power barrier that prevents widely distributed machine intelligence. By performing offline inference near data source, TinyML enables greater responsiveness while avoiding the energy cost associated with wireless communication, which is far higher than that of computing. Since TinyML has a significant role to play in future technology, a widely accepted benchmark is required to unlock the full potential of the field.

Neural Networks on MCUs. Mounting interest in TinyML has led to some maturity in the field, thus releasing software stacks such as Edge-ML [1], Open-NN [2], RCE-NN [3], TensorFlow Micro inference runtime [4]. Particularly the TFMicro attracts attention due to its ability to allow the portable and straightforward execution of NNs on commodity MCUs. For the TinyML benchmark, over the code generation-based methods such as uTensor [5], we use TFMicro as it provides portability across MCU vendors, at the cost of a fairly minimal memory overhead. Also, TFMicro uses an interpreter to execute an NN graph, which means the same model graph can be deployed across different hardware platforms such as GPUs, TPUs, and also MCUs. Although NNs (DNNs, CNNs, RNNs, LSTMs) are the dominant force in ML, non-NN based ML solutions also show a great importance in TinyML due to their low-power computation and memory requirements.

[1]Trained TFLite models, complete benchmark results, and more details on chosen MCU boards, datasets, NNs are available at https://github.com/bharathsudharsan/TinyML-Benchmark-NNs-on-MCUs

TABLE I
MCUs, datasets, NNs chosen for TinyML benchmark.

	Board Name	Processor, Flash (MB), SRAM, Clock (MHz)
MCUs	B1: Teensy 4.0	Cortex-M7, 2, 1MB, 600
	B2: STM32 Nucleo H7	Cortex-M7, 2, 1 MB, 480
	B3: Arduino Portenta	Cortex-M7+M4, 2, 1MB, 480
	B4: Feather M4 Express	Cortex-M4, 2, 192KB, 120
	B5: Generic ESP32	Xtensa LX6, 4, 520KB, 240
	B6: Arduino Nano 33	Cortex-M4, 1, 256KB, 64
	B7: Raspberry Pi Pico	Cortex-M0+, 16, 264KB, 133
	Name: Feature dimension, Class counts	
Datasets	D1: Iris Flowers: 4, 3	D6: Breast Cancer: 30, 2
	D2: Wine: 13, 3	D7: Texture: 40, 11
	D3: Vowel: 13, 11	D8: Drive Diagnosis, 48, 11
	D4: Silhouettes: 18, 4	D9: MNIST Digits: 64, 10
	D5: Anuran Calls: 22, 8	D10: Human Activity: 74, 6
	Name: Topology	
Fully Connected NNs	*FC 1x10*: 1 layer with 10 neurons	
	FC 10+50: 1^{st} layer with 10 neurons, $2^{nd}$with 50	
	FC 10x10: 10 layers, with 10 neurons in each layer	

Supervised ML Models on MCUs. The sklearn-porter [6], m2cgen [7], emlearn [8], micromlgen [9] are the popular libraries to generate optimized C code. These libraries can be used to create ML use case models with non-NN ML techniques like KNN, SVM, and Naive Bayes that can execute on TinyML hardware. Here, the trained ML model is first ported to produce its plain C version, then written/exported inside a `.h` file. When the users aim to port tree-based decision tree, random forest models, the SRAM optimized method [10] can be used.

II. TinyML Benchmark

Table I presents the MCU boards (B1 - B7), datasets (D1 - D10) and NNs (3 types) used for TinyML benchmark. The chosen MCUs are popular example hardware that is widely used to design IoT devices, and billions of similar specification devices exist globally. In TensorFlow, we defined 3 types of fully connected NNs (*FC 1X10*, *FC 10+50*, *FC 10X10*) and trained using D1 - D10 datasets. The resultant 30 models are converted into TFLite format and then converted into a C byte array. Later these models are compiled and flashed using Arduino IDE. Finally, each model are executed on B1 - B7, and the experimental results are reported in Fig. 1. For statistical validation, the results correspond to the average of 5 runs. In the remainder of this section, we analyze the results.

Inference Performance on MCUs. Fig. 1. a, presents the average time taken by MCUs to infer using D1 - D10 datasets. For all 3 NN types, Teensy 4.0 (B1) is the fastest as it performed

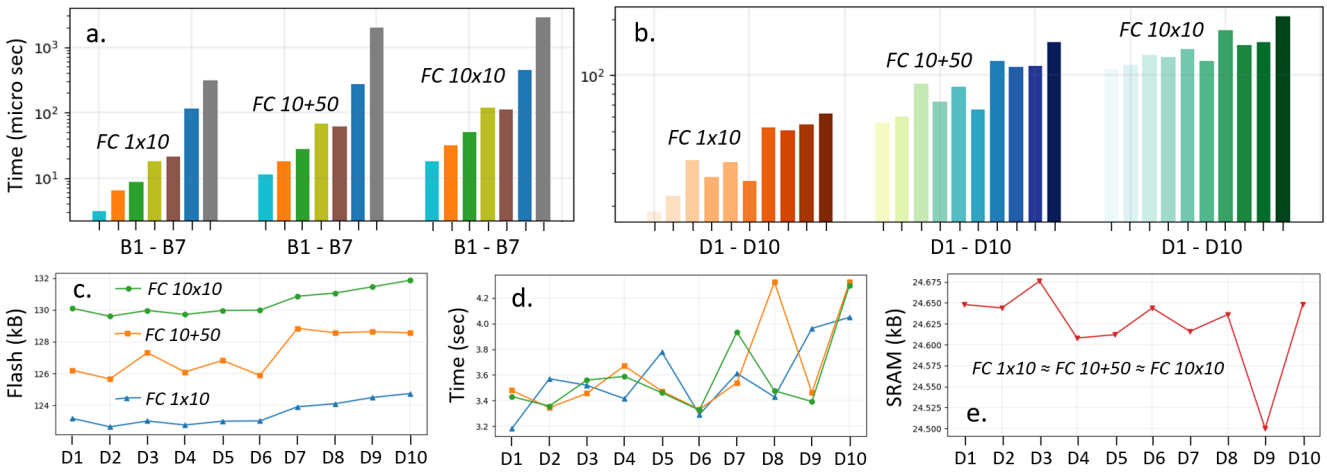

Fig. 1. TinyML benchmark results: a. Average time taken by boards B1 - B7 to infer using all datasets D1 - D10; b. Inference time on B2 for D1 - D10; c. Flash memory consumed by B2; d. Time consumed by Arduino IDE to compile each network trained using D1 - D10; e. SRAM memory consumed by B2.

inference in 3.14 μs, 11.13 μs, 18.12 μs respectively. For the same data samples, Raspberry Pi Pico (B7) is the slowest (≈99 - 175 x times slower than B1), as it took 313.77 μs, 1953.96 μs, 2801.82 μs. Although B7 has a faster clock than Arduino Nano 33 (B6), it is still slow as Cortex M4 is superior to Cortex M0+. Although B1 - B3 has the same Cortex M7 processor, B1 still is significantly faster as it has the highest clock speed of 600 MHz.

Fig. 1. b, presents the complete inference time on the second-fastest STM32 Nucleo H7 (B2) for each of the 30 models. When considering *FC 1x10*, for the 4 features D1, it took 5.16 μs to infer, and for the highest 74 features D10, it took 872.85 μs to infer. When considering *FC 10x10*, for D1, it took 20.15 μs, and 3369.54 μs for D10. Portenta (B3) and B2 are on quite a par since they share processors from the same ARM Cortex-M7 family, but B2 is faster across all the NN topologies.

Onboard Accuracy. We fed test sets to each of the 30 models when executing on B1 - B7 via COM Port to perform inference. We report that the same models, from board to board, show only 0.4 - 1.6 % variation in onboard accuracy. Also, the models during execution on MCUs, show the same level of accuracy, F1 score as its original TFlite models when evaluated on Google Colab.

Memory Consumption on MCUs. The run-time variables generated during NN execution are stored in the SRAM. The chosen boards have only 192 kB to a max of 1 MB SRAM, which restricts the deployment and execution of large models. SRAM in MCUs is always limited since adding more memory leads to higher power leakage and manufacturing costs. Before flashing, when compiling the NNs and IoT applications, the memory requirements for target boards are calculated by the compiler (such as Atmel Studio, Keil MDK) in use. In Fig. 1. c-e, we provide the time taken by Arduino IDE to compile each of the 30 models for B2, along with the complete Flash and SRAM requirements. The models trained using the datasets with more features, classes consumed higher compilation time, and higher fash memory.

Price-performance Ratio. Portenta (B3) that costs ≈100 $ (the price of NVIDIA Jetson Nano GPU) is the most expensive board but still does not outperform the ≈20 $ Teensy 4.0 (B1). Moreover, B1 can be the most fastest yet reasonably priced board as it can be overclocked up to 1 GHz. The ≈30 $ STM32 Nucleo H7 (B2) is the second-fastest, contains many IO pins and dev-related features like STLink debugger. ESP32 (B5) has the best price-performance, as it is only ≈3 $ and just ≈17 - 91 μs slower (see Fig. 1. a) than B1. The Raspberry Pi Pico is as cheap as ESP32 but ≈292 - 2692 μs slower.

III. Conclusion

TinyML is a rapidly evolving field that requires comparability amongst low-power hardware innovations, particularly when executing neural workloads. In this paper, to enable continued progress and stability in this field, we presented and analyzed the onboard performance of 30 NN models on 7 popular MCU boards. We open-sourced the complete benchmark results that can be utilized to speed up the design phase (going from idea to product) of ML-powered IoT hardware.

Acknowledgement

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/16/RC/3918 (Confirm) and also by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289_P2 (Insight), with both grants co-funded by the European Regional Development Fund.

References

[1] "Edgeml: https://microsoft.github.io/edgeml/," 2021.
[2] "Opennn: https://www.opennn.net/," 2021.
[3] B. Sudharsan, J. G. Breslin, and M. I. Ali, "RCE-NN: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices," in *International Conference on Internet of Things*, 2020.
[4] "Tfmicro: https://www.tensorflow.org/lite/," 2021.
[5] "utensor: https://github.com/utensor/utensor," 2021.
[6] "sklearn-porter: https://github.com/nok/sklearn-porter," 2020.
[7] "m2cgen: Code-generation for various ml models into native code." 2020.
[8] "emlearn: https://github.com/emlearn/," 2020.
[9] "Micromlgen: https://github.com/eloquentarduino/micromlgen," 2021.
[10] B. Sudharsan, P. Patel, J. G. Breslin, and M. I. Ali, "Ultra-fast machine learning classifier execution on iot devices without sram consumption," in *IEEE PerCom Workshops*, 2021.