Low-Overhead Early-Stopping Policies for Efficient Random Forests Inference on Microcontrollers

Low-overhead Early-stopping Policies for Efficient Random Forests Inference on Microcontrollers

Francesco Daghero[1], Alessio Burrello[2], Chen Xie[1], Luca Benini[23], Andrea Calimera[1], Enrico Macii[1], Massimo Poncino[1], and Daniele Jahier Pagliari[1]

[1] Politecnico di Torino, Turin, Italy, name.surname@polito.it
[2] University of Bologna, Bologna, Italy, name.surname@unibo.it
[3] ETH, Zurich, Switzerland, surname@iis.ee.ethz.ch

Abstract. Random Forests (RFs) are popular Machine Learning models for edge computing, due to their lightweight nature and high accuracy on several common tasks. Large RFs however, still have significant energy costs, a serious concern for battery-operated ultra-low-power devices. Following the adaptive (or dynamic) inference paradigm, we introduce a hardware-friendly early stopping policy for RF-based classifiers, halting the execution as soon as a sufficient prediction confidence is achieved. We benchmark our approach on three state-of-the-art datasets relative to different embedded classification tasks, and deploy our models on a single core RISC-V microcontroller. We achieve an energy reduction ranging from 18% to more than 91%, with an accuracy drop lower than 0.5%. Additionally, we compare our approach with other early-stopping policies, showing that we outperform them.

Keywords: Machine Learning, TinyML, Adaptive Inference, Dynamic Inference, Energy-efficiency, Random Forests, Microcontrollers

1 Introduction

Machine Learning (ML) inference is one of the core components of an increasing number of Internet of Things (IoT) applications, from time-series processing to computer vision [12, 31]. The cloud-based paradigm is the most popular deployment approach for this kind of application, relying on a powerful high-end server performing the inference with a computationally expensive and accurate model. IoT devices are instead only responsible for the data collection and transmission, offloading almost all the computations to the cloud and receiving back the final output of the inference.

This approach however presents several limitations, mostly stemming from the need to continuously send data to remote hardware [36, 33]. A stable and reliable internet connection is in fact permanently necessary, an assumption that may not always hold (e.g. for a wearable system used in a remote area). Even when present, wireless connectivity may be unstable or slow, increasing the inference latency in an unpredictable way, and posing a serious challenge for real-time

applications. Additionally, transmitting possibly sensitive data over an untrusted network poses a challenge to security, leading to privacy-related concerns. Last but not least, sending large amounts of data to the cloud is an energy-hungry operation [40], reducing the lifetime of battery-operated devices.

For all the above reasons, *edge computing* is becoming an increasingly popular approach for ML-based IoT applications [36, 33], consisting of an on-device deployment of the ML model, which completely eliminates (or limits to particularly complex tasks) the interaction with remote servers. Performing all computations locally eliminates latency and privacy concerns at the source, while also possibly obtaining higher energy efficiency.

However, directly deploying ML models at the edge is not easy due to their memory and computational requirements, which clash with the tight constraints of IoT nodes, mostly based on Microcontrollers (MCUs). Deep Learning (DL) approaches, in particular, while reaching state-of-the-art accuracy on many domains, maintain high complexity even after applying multiple optimizations [18, 10], and are often too expensive, in terms of energy consumption and memory occupation, for MCU-based edge devices.

There are however lightweight alternatives to DL, particularly suited for easy recognition tasks such as the ones involved in IoT applications. Among them, tree-based ensemble models, and in particular Random Forests (RFs) [5], are increasingly popular. Their success stems from their inexpensive inference, requiring often a small number of compare and branch operations, while also having a compact memory footprint. At the same time, Random Forests (RFs) [5] often reach an accuracy close to DL models and good resistance to overfitting for simple IoT tasks, such as human activity recognition, ECG analysis, and seizure detection [35, 14, 30, 15]. For instance, the DL-based classifiers proposed by the authors of [30] for an Electrocardiogram (ECG) anomaly detection requires around 200k arithmetic operations and the storage of a similar amount of parameters, while in Section 6, we show that an RF can achieve comparable accuracy with $\approx$ 2k parameters and less than 1k operations.

Although less expensive than DL, the inference time and energy consumption of RFs can nonetheless have a relevant impact on the battery lifetime of MCU-based systems. Hence, inference optimization techniques are fundamental even for these simple models.

In this work, which extends [11], we propose one such optimization originating from the observation that, for single-core MCUs, RF inference time and energy costs are *linearly dependent on the number of trees* (the forest "width"). In fact, the MCU will evaluate all the Decision Trees (DTs) that constitute the ensemble in a sequential fashion, one after the other. However, evaluating the whole forest may be necessary only for a subset of complex input samples, while being wasteful in terms of energy for easy inputs. Intuitively, if the initial bunch of trees executed during an inference predicts that the output belongs to a specific class with *very high confidence*, it becomes unlikely (or even impossible) that the remaining DTs will overturn that prediction. Thus, the execution of

the latter can be *skipped completely*, reducing the total time and energy, while not affecting the final accuracy negatively.

Leveraging this idea, we propose an *early stopping* mechanism for RF inference, which stops the evaluation of DTs as soon as a user-defined confidence level has been reached. While *adaptive (or dynamic) inference* approaches such as this are widely adopted for DL [28, 37, 19, 22, 20, 6], to the best of our knowledge, we are the first to consider them for RFs, with a focus on embedded/IoT deployment. In fact, the few existing techniques for tree-based models [39, 16] have been studied only theoretically, without evaluating a practical implementation on a low-power device, hence largely ignoring some important overheads derivating from their deployment. In contrast, our proposed method is designed specifically for embedded RFs, being based on low-overhead early stopping policies, easy to execute efficiently at runtime, with minimal latency/energy overheads.

We benchmark our approach on three different embedded tasks, i.e., human activity recognition, heart failure detection, and gesture recognition. Deploying our models on a popular single-core RISC-V MCU, we obtain an energy reduction ranging from 18% to 91% with less than 0.5% accuracy drop, with respect to a standard (i.e., static) RF inference.

2 Background

2.1 Decision Trees and Random Forests

When used in a supervised learning setting, Decision Trees (DTs) learn a set of decision rules extracted at training time from the data features, in order to perform either a classification or regression task. Several training algorithms for DTs have been proposed in the literature [23], differing in the criteria used for selecting the features and decision thresholds considered at each internal node. The details of the training phase are out of the scope of this work, and interested readers may refer to [23]. Since this work proposes an *inference* optimization, herein we detail only the operations of the inference phase.

Figure 1 shows a high-level overview of a "grown" (i.e., trained) DT used for a 2-class classification task, in which leaf nodes are depicted as rectangles and other nodes as circles. Leaf nodes contain the probabilities of the input belonging to a specific class, while each non-terminal node stores the index of the input data feature considered for branching in that node, and the threshold that determines the left or right branch.

The DT inference pseudo-code is shown in Algorithm 1, where Root(T) denotes the root node and Leaves(T) the set of leaves. Feature(n) and Threshold(n) are the input feature and comparison threshold considered in the n-th node, and Left(n) and Right(n) are the left and right children of the node. Finally, Prediction(n) is only defined for leaves and contains the corresponding output prediction (an array of probabilities for a classification, and a continuous scalar value for regression).

The time complexity of Algorithm 1 is $O(D)$, where D denotes the tree depth, i.e., the maximum length of a path from the root to the leaves. For a

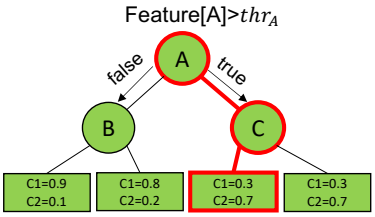

Fig. 1: High-level overview of a DT structure for a 2-class classification problem. The leaves are represented as rectangles, each storing the class probabilities of an input belonging to that path. Other nodes are represented as circles

Algorithm 1: Decision Tree inference.

```
n = Root(T)
while n ∉ Leaves(T) do
    if Feature(n) > Threshold(n) then
        n = Right(n)
    else
        n = Left(n)
    end
end
out = Prediction(n)
```

classification, an additional $O(M)$ scan over the output probabilities is then needed to determine the final class label, where M is the number of classes. The memory complexity, instead, grows with $O(2^D)$, i.e., it is proportional to the total number of nodes, which is at most 2^D in the case of a *balanced and unpruned* DT, with all root-leaf paths having the same length [23].

DTs are prone to over-fitting, suffer from high variance even with small perturbations in the training data, and introduce biases when used with unbalanced datasets. In order to overcome these limitations, Random Forests (RFs) have been proposed [5]. RFs are ensembles of DTs (called "weak learners"), trained with *bagging* (bootstrap aggregating) and, more recently, random features selection [29]. In practice, each DT is trained on a random subset of the training samples, drawn with replacement, and on a limited set of the input features, thus ensuring a low correlation among weak learners, which reduces overfitting.

At inference time, the individual DTs predictions are combined to obtain the final RF output, as shown in Figure 2. Specifically, in early implementations of RFs for classification, each weak learner outputs a class prediction, then aggregated with a majority voting. In contrast, modern RF libraries [29] store in the leaf nodes of the trees the entire set of class probabilities, thus allowing the final predictions to be computed as the average (or sum) of all the weak learners' class probabilities. The final label is then selected as the argmax of the array.

Algorithm 2 reports the pseudo-code of a RF inference pass, for an implementation that loops sequentially over the weak learners (such as the one for a

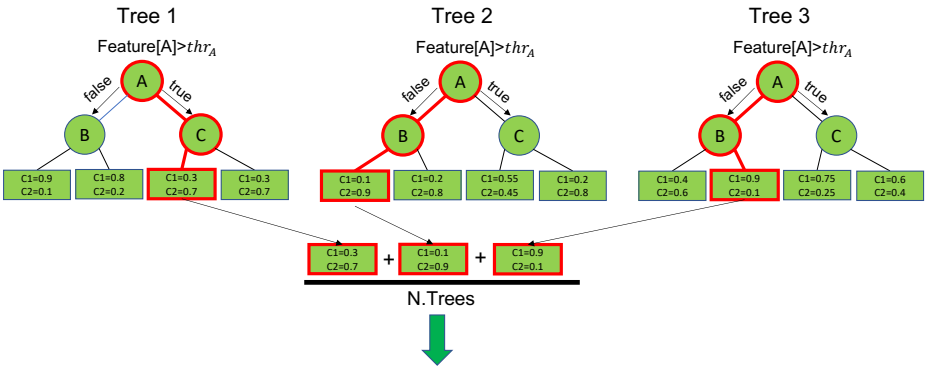

Fig. 2: High-level overview of a RF inference with width 3 and depth 3. The output of each DT is averaged to obtain the final predictions of the ensemble.

single-core processor). The function DecisionTreeInference corresponds to Algorithm 1. From a complexity point of view, a RF of *width* N, i.e., including N trees, has a time and memory complexity of $O(ND)$ and $O(N2^D)$ respectively, where D is the maximum depth over all weak learners. Lastly, the argmax that extract the predicted label has time complexity $O(M)$, as for a single DT.

Algorithm 2: Random Forest Classification.

$out = \mathbf{0}_M$ //array of 0s of size M
for $T \in$ Forest **do**
  $out = out +$ DecisionTreeInference(T)
end
$class = \arg\max(out)$

2.2 IoT End Nodes

The great majority of IoT end-nodes are based on low-power microcontrollers (MCUs), whose main compute unit is a general-purpose CPU, typically based on a RISC instruction set [17]. This is mainly due to their employment on extremely low-cost devices. In this context, the flexibility and high programmability of MCUs make them preferable to custom Application-Specific Integrated Circuits (ASICs), potentially orders of magnitude more efficient, but whose design and manufacturing costs are only affordable for high-end, high-volume devices.

Specifically, the RISC-V Instruction Set Architecture (ISA) is recently becoming more and more adopted both in the research world and in companies for the realization of IoT devices [13, 34]. Following this trend, we benchmark our results on a RISC-V processor from the PULP family [8]. Given the very low-power

requirements and tight cost constraints of our target applications, we select one of the smallest architectures in the family, the single-core *PULPissimo*. This device is based on a RI5CY core with a 4-stage, in-order, single-issue pipeline. The core implements the *RV32IMC* ISA, enhanced with domain-specific extensions for DSP, such as Single Instruction Multiple Data (SIMD) operations, hardware-loops, and loads/stores with index increment, and no caches, all design choices aimed at providing significant speedups and energy saving for ML applications.

2.3 Machine Learning at the edge

In order to bridge the gap between the computational requirements of Machine Learning models and the limited resources of IoT end nodes, several works have introduced optimizations with the goal of making the inference as energy efficient as possible, without affecting significantly its accuracy [9, 28, 19, 37, 27, 21, 1, 15]. These optimizations can be divided into two categories: *static* and *adaptive*.

The first category optimizes a model before deployment, usually at training or post-training, with the goal of reducing the inference latency, energy, or the memory required to store the classifier parameters. Pruning and quantization are among the most popular static optimizations for DL [26, 18], reducing models' complexity respectively through the removal of redundant parameters or by using low-precision arithmetic. Notably, pruning can also be applied to DTs and RFs during their training (or growth), with the goal of eliminating unimportant nodes from the trees, hence reducing the number of parameters of the model [25].

Static approaches are, by definition, unable to efficiently support multiple runtime operating modes, with different complexity versus accuracy trade-offs. Nonetheless, this would be useful to respond to changes in external conditions, such as the remaining battery life of the device or, more interestingly, to promptly adapt to variations the complexity of the task being executed [28, 9].

The naive solution to achieve such runtime flexibility is deploying multiple, independent models, each with a different accuracy and computational complexity, and selecting the most appropriate model at any given time. However, this approach is often unfeasible due to the limited memory of IoT end nodes, which makes it impossible to store a large number of models on a single device.

Adaptive (or *dynamic*) inference techniques try to overcome these limitations, proposing a set of optimizations, mostly orthogonal to static ones, that allow multiple operating points at runtime with limited memory overheads. These optimizations are based on the concept that *not all inputs are equally hard to process* for a ML model, and that easy inputs are often far more common than difficult ones. Adapting the computational effort spent for inference based on the difficulty of the processed input (i.e., reducing the effort for easy inputs and increasing it for difficult ones), could then enable significant energy savings, while keeping the classification accuracy unchanged. Accordingly, one of the focal points of any adaptive inference technique is the design of an automatic mechanism (or *policy*) for discerning between easy and hard inputs. Furthermore, this policy should introduce low computational overheads, which do not overshadow the energy savings obtained thanks to the adaptive effort tuning.

3 Related Works

In the literature, adaptive inference implementations have been proposed by multiple works, with a particular focus on DL. One of the earliest endeavors proposed the so-called Big-Little scheme [28], combining two deep neural networks with different complexity and accuracy. At runtime, the inexpensive yet less accurate network, named "little", performs the first inference on each input. The confidence of this model is then evaluated, stopping the execution in case it surpasses a user-defined threshold. Otherwise, the input is fed to the second model, an accurate yet more complex network named "big", and its output is taken as final prediction. The rationale of this technique is that, as long as the easy inputs, predicted with high confidence by the "little" model, are more common than hard ones, the average energy required for inference will decrease significantly. At the same time, the final accuracy is not affected, since complex inputs are still re-directed to the "big" model. The main flaw of this approach, however, lies in its considerable memory overhead, since it requires the deployment of two completely separate networks on the edge device.

Based on this observation, multiple subsequent works have proposed alternative adaptive inference schemes for DL, that try to address the memory overhead problem. For instance, deriving the "little" network from the "big" model by using only a subset of the layers, channels or a lower bit-width quantization may reduce significantly the number of parameters that need to be stored [37, 27, 19]. On an orthogonal direction, other works enhanced the Big-Little paradigm by increasing the number of cascaded models to more than two, or improving the stopping mechanism to handle class-specific confidence [37, 9].

Applications of the adaptive paradigm to shallow ML models, and in particular to tree-based ones, are far less common compared to DL [39, 16, 32]. The authors of [32] propose an early stopping criterion for RFs and other tree ensembles, which allows reducing the number of trees invoked for inference on easy inputs, modeling it with a binomial or multinomial distribution (depending on the number of classes). The approach is benchmarked on 7 small public datasets and one private, showing that, for ensembles with a large amount of trees, they reduce the average number of weak learners required for inference by 63%. However, the proposed criterion requires the storage of a large lookup table with a dimension in the order of $O(N^2)$, which introduces a significant memory overhead (10s of kB) for large forests.

In another work, the authors of [16] propose an approach to determine the best order of execution for weak learners depending on the most likely class indicated by the DTs that have been already executed. This selection happens at runtime, and takes into account the different computational costs associated to weak learners due to their reliance on different features, finding the optimal trade-off between complexity and accuracy to select the next DT. The authors leverage a mixture of Gaussian distributions to design a probabilistic model of the classifier, exploiting it to trigger an adaptive early stopping based on the posterior probabilities. Furthermore, they also introduce a dimensionality reduction technique to prune the number of computations required to perform

the selection of the following DT. However, on an ultra-low-power MCU-based device, the introduced overhead would overshadow the energy savings obtained by performing the inference on a subset of the weak learners. Hence, as stated by the authors in the original paper, this approach becomes effective only if the target task involves *very complex feature extractions*, which is rarely the case for simple IoT applications.

The work closest to ours is named Quit When You Can (QWYC) [39]. In this case, the authors focus on binary classification tasks and propose a simple early stopping based on two probabilities thresholds (ϵ_- and ϵ_+) derived statically post-training. Additionally, the authors propose a static sorting of the weak learners, so that the DTs most likely to trigger an early stop are executed first. At inference time, as soon as one of the probabilities of the last executed DT is either lower than ϵ_- or higher than ϵ_+, the early stop mechanism is triggered, selecting the negative or positive class as the final prediction, respectively.

While QWYC requires a small overhead at runtime (only two comparisons), the extension to a multi-class problem is not straightforward. The authors propose a possible implementation of the multi-class version, but do not show any results for it, leaving its effectiveness yet to be tested. Moreover, their approach is still not tested on a real low-power IoT node.

In summary, all the works mentioned above are purely theoretical, and their effectiveness is evaluated only from a complexity reduction point of view, i.e., computing the average number of DTs executed for inference, with no deployment on a real embedded device. Additionally, many of these works introduce considerable overheads either in terms of memory or time/energy, both of which are very precious resources on IoT devices. In our work, we compare the proposed approach with QWYC [39], showing that we obtain similar or better performance, despite the higher simplicity and generality of our method.

4 Motivation and Goal

RFs generally use a large number of trees N (e.g, between 10 and 100) to improve the accuracy over single DTs. Indeed, using many weak learners instead of a single powerful one is demonstrated to reduce the overfitting and the bias of the model, leading to a better generalization on new unseen data and higher accuracy overall. On the other hand, easy inputs would be correctly classified also by means of fewer trees than the ones present in the complete forest. In this case, employing the full set of trees of the RF is sub-optimal, leading to a possible increment of energy consumption and higher latency, which could be critical for IoT devices. Nonetheless, deploying a smaller RF with N' trees, where $N' < N$ may result in errors when classifying more complex samples, and therefore in a reduction of the overall accuracy.

Our work is based on these observations: our aim is to design an adaptive early stopping policy for tree-based ensembles, minimizing the DTs executed to correctly classify easy inputs, while exploiting more DTs (up to the entire RFs) to classify the most complex ones. The key to achieve high energy saving

through this method lies in the light but accurate mechanism to distinguish easy from hard inputs. Therefore, the main goal of this work is the search for a way to allow an early stopping of the inference, before the execution of the whole RF, without affecting the final accuracy. At the same time, we also look for a *lightweight early-stopping policy*, to avoid overshadowing the savings obtained thanks to the lower number of weak learners executed.

5 Methodology

5.1 Aggregated Score Thresholds for Early Stopping

Among the various confidence metrics introduced in the literature for adaptive early-stopping in classification problems, the most common ones are based on the output probabilities (P^t) produced by the last model t executed. A first approach considers the highest probability (i.e the one associated with the most likely class) to compute the confidence of the model. A large maximum probability denotes a classifier confident in its prediction, while a small value is associated with an uncertain classification. We name this approach *Max Score* (or simply *Max*). This metric is fast to compute at runtime, requiring $O(M)$ pairwise comparisons. The second approach, named *Score Margin* (SM), extends the Max policy by considering the two largest probabilities of the model. For a target model t, we can compute its SM as:

$$SM = \max(P^t) - \max_{2nd}(P^t) \tag{1}$$

where $\max_{2nd}(P^t)$ denotes the second largest value in vector P^t. Even though the SM requires more operations compared to the simpler Max (around twice), it makes the computation of the confidence more robust. For instance, the max value for a 11-class prediction problem will be 0.5 in case of a distribution of $P^0 = 0.5$, $P^1 = 0.5$, and $P^{2-10} = 0$, which corresponds to a very uncertain prediction, but also in the case of $P^0 = 0.5$, $P^{1-10} = 0.05$, which is instead a quite reliable output. On the other hand, the SM would be 0 in the first case and 0.45 in the second, correctly capturing the different confidence of the model in the two cases. From this example, the reader can understand why the SM metric has become so popular in recent literature.

To determine when early-stopping should be performed, a threshold α is compared with the selected confidence metric (Max or SM): when the metric is higher than α, the inference is stopped and the output prediction is produced based on (some of) the outputs of the classifiers that have been already executed. Therefore, the value of α directly controls the energy vs accuracy trade-off, since it determines how many classifiers are executed on average.

The advantage of this early-stopping criterion lies in its inexpensive derivation (requiring a single comparison after the computation of the corresponding metric), while being accurate as long as the classifiers' output probabilities are *calibrated* (i.e., proportional to the likelihood of the class to be the correct one). Furthermore, the threshold α can be changed at run-time, e.g., based on the

system condition (level of battery charge, period of the day, etc.), to produce more accurate or more energy-efficient classifications.

Normally, the confidence metric (Max or SM) is computed using the output probabilities of the *last executed classifier* t, neglecting the outputs of the models executed before it, i.e., the "history" of the ensemble. This approach is ideal for cascades of increasingly accurate classifiers, since taking into account the $t-1$-th classifier output may actually worsen the prediction of the (much more accurate) t-th model [28, 19]. However, it is not appropriate for an ensemble of *equally predictive* weak learners, such as a RF.

Starting from this observation, we extend the policies described above so that the early stopping is triggered using the aggregated predictions of *all the already executed classifiers* ($P^{[1,t]}$). Noteworthy, easy inputs will in fact have partial aggregated probabilities already skewed towards one class even after the execution of just a few DTs. Therefore, it is unlikely or even mathematically impossible that when the aggregated probabilities are sufficiently skewed toward one specific class, the remaining DTs will overturn the prediction, which makes their execution not necessary to improve the accuracy of the prediction.

We define the partial output of a RF after executing t trees as:

$$P^{[1:t]} = \sum_{i=1}^{t} P^i \tag{2}$$

where P^i denotes the vector of output probabilities of the i-th weak learner. We then define the *Aggregated Max Score* (S) early-stopping policy after the execution of the t-th classifier as the rule:

$$S^t = \max(P^{[1:t]}) > \alpha \tag{3}$$

while the *Aggregated Score Margin* SM policy is defined as:

$$SM^t = \max(P^{[1:t]}) - \max_{2nd}(P^{[1:t]}) > \alpha \tag{4}$$

In our experiments, we consider both of these policies, with a tunable threshold α, to determine when to perform early-stopping in a RF ensemble. To the best of our knowledge, we are the first to propose an early-stopping approach that considers the aggregated probabilities of the weak learners, while being based on a lightweight comparison with a threshold. Our results show that we outperform other state-of-the-art approaches that leverage only the last weak learner of the ensemble, achieving higher energy efficiency during the inference while also avoiding large accuracy drop.

Figure 3 shows a high-level overview of the adaptive inference mechanism proposed in this work, for the case of the SM policy and with a batch $B = 1$ (see Section 5.3 below). The RF represented has $N = 3$, $M = 2$, and $D = 3$. Orange nodes are those "selected" by the series of compare-and-branch operations for a hypothetical input. In a nutshell, after executing each DT, the partial predictions are accumulated and used to determine whether the confidence of the inference up to tree t is enough to trigger an early stop, based on α.

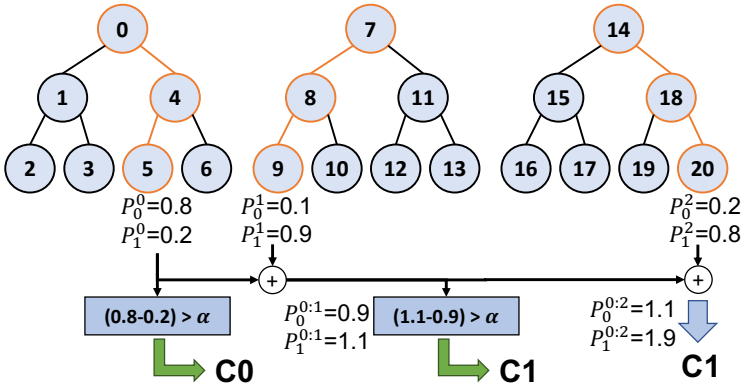

Fig. 3: High-level overview of the proposed adaptive inference method for RFs, for $B = 1$. At each step, the SM is computed on the partially aggregated scores.

5.2 Deployment on MCUs

Due to the lack of open-source RF libraries tailored for the target MCU (described in Section 6), we design and deploy an optimized implementation in C language of both the traditional RF and of our adaptive version, i.e, a RF augmented with the early-stop mechanism described above.

We take inspiration from the open-source implementation available in OpenCV [4], optimizing it for our target ultra-low-power platform. The main difference resides in the way RF nodes, leaves, and thresholds are stored: OpenCV lists are replaced with C arrays in our version, both to save precious memory space and to improve the memory locality of the data. Figure 4 shows the three main arrays that compose our RF representation, i.e., FOREST, ROOT, and LEAVES.

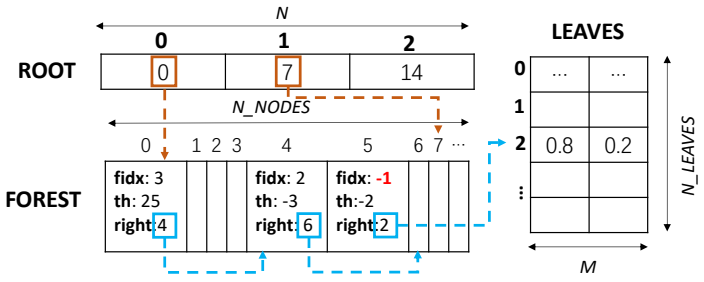

Fig. 4: C data structures of our RF implementation.

The array **FOREST** stores in each element a "struct" with the information relative to a node belonging to one of the RF trees. The struct has three member variables:

- *fidx* is the index of the feature on which the split has been performed. It is used to select the correct value from the input feature array to be compared with the threshold *th* at inference time. This value is set to -1 in leaf nodes.
- *th*: the value to be used as a comparison to determine the following node to visit; left child if the input feature is lower than *th*, right otherwise.
- *right*: the index in FOREST of the right child of the current node. Note that to reduce the memory occupation, the left child is always stored as the following element of the array. For leaf nodes, the right child index stores the index of the corresponding leaf probabilities in the LEAVES array.

The other two arrays, **LEAVES**, and **ROOT**, store respectively the output probabilities of each leaf and the indexes of the root node of each DT in FOREST.

Figure 4 reports some data structure values corresponding to the RF shown in Figure 3. In particular, it shows the elements of FOREST which correspond to the nodes in the decision path of the leftmost DT in Figure 3.

To further compress the memory required to store our RFs, we quantize to 16-bit integers all the fields of the FOREST and LEAVES arrays, simplifying also the deployment on MCUs not equipped with a Floating Point Unit. We verified that quantizing the inputs, comparison thresholds, and output probabilities to 16-bit integers yields close to 0 accuracy drop, compared to the original floating-point model. We also reduce to 16-bit the precision of the ROOT elements, which guarantees the possibility of deploying large RFs (up to 2^{16} nodes), while significantly reducing the memory overhead of this vector.

5.3 Tree Batching

One of the main advantages of the aggregated Score Margin and Max early-stop policies lies in their lightweight nature. Specifically, their time complexity at inference time is $O(M)$ to find either the highest or the two highest probabilities and $O(1)$ to compare with the threshold α. In the case of dynamic inference systems for deep learning [28, 37], this computational overhead is negligible w.r.t the execution of the individual neural networks. On the other hand, when working on lightweight classifiers such as RFs, the computation of either the aggregated Max or the SM can affect negatively the energy gains obtained by avoiding the execution of the full forest. In fact, as introduced in Section 2, the time complexity for a single DT inference is $O(D)$, plus $O(M)$ for the argmax over classes. Large yet shallow adaptive RFs may then have a significant overhead for the early-stopping decision (when D is comparable or lower than M), becoming significantly less efficient than a static forest with fewer trees.

In order to tackle this problem, we propose a simple but effective approach to reduce the impact of the early stopping policy, named *tree batching*. Rather than evaluating the aggregated confidence metric after every DT inference, we instead perform its computation after a *batch* of B trees. This additional hyper-parameter has a contrasting effect on the energy consumption of the system. In fact, larger batch sizes can reduce the overhead introduced by the computation of the confidence metric by a factor of B, thus saving additional energy. On

the other hand, evaluating the stopping criterion every B trees may cause the classifier to perform up to $B-1$ additional inferences that could be avoided with $B = 1$. Empirically we show that depending on the dataset, the results obtained setting $B > 1$ can outperform the ones of $B = 1$ in terms of accuracy vs energy.

Algorithm 3 reports the pseudo-code of the adaptive inference with batch size B, where $Batch(b)$ denotes the subset of weak learners belonging to the b-th batch. $Metric(out)$ represents instead the computation of the confidence metric at tree t, e.g., $SM^{1:t}$ in case of the aggregated Score Margin.

Algorithm 3: Adaptive Random Forest Classification.

```
for b ∈ [0, N/B] do
    for T ∈ Batch(b) do
        out = out + DecisionTreeInference(T)
    end
    if Metric(out) > α then
        break
    end
end
class = arg max(out)
```

5.4 Tree Ordering

The introduction of an early stopping mechanism that depends on the output probabilities of each DT makes the inference results become dependent on the order of the weak learners. As opposed to the classic approach, which sums or averages the contributions of all DTs, the adaptive inference will, for most of the inputs, leverage only the probabilities of a subset of them. As a consequence, invoking first the most informative and confident DTs increases the probability that the early-stopping mechanism will be triggered sooner, and therefore the energy savings. Intuitively, one could then think of finding an optimal ordering of the DTs on a subset of the training data (e.g. the validation set), by means of a search algorithm such as greedy, random, exhaustive search, or others. As mentioned in Section 3, multiple previous works including QWYC [39] have proposed mechanisms to determine such a "hardcoded" ordering of the classifiers.

However, in our experiments, we demonstrate that such an optimized ordering does not actually provide statistical advantages over executing the DTs in a random order. In fact, we compare multiple permutations of the DTs composing the ensembles, showing that those orderings that reduce the average number of weak learners per inference on the validation dataset, do not obtain comparable results on the test set. In other words, there is no correlation between the "goodness" of a given ordering on the two data subsets.

Therefore, we conclude that an optimized hard-coded ordering of weak learners do not provide advantages, at least in our considered scenario, i.e., for a RF

classifier and considering the simple early-stopping policies described above. In contrast, input-dependent DTs reordering could be effective, but is extremely difficult to implement at low overhead [16], hence we leave it to future work.

6 Results

6.1 Benchmarks, deployment setup, and comparisons

We evaluate the proposed technique on three different datasets for popular tinyML tasks: ECG5000 [7], Ninapro DB1 [2], and UniMiB-SHAR [24].

ECG5000 [7] features annotated electrocardiogram (ECG) data, provided already preprocessed in windows of 0.8 seconds, each containing a single heartbeat. We perform the same task as the authors of [30], which consists in detecting whether congestive heart failure happens. For this dataset, we take as a baseline for comparison a static RF with $N = 40$ and $D = 3$. Our adaptive model uses an identical RF structure, but dynamically reduces the number of trees executed at runtime as described in Section 5.

The second set of experiments is performed on the popular **Ninapro** DB1 [3], featuring Electromyography (EMG) signals of 27 healthy subjects performing different hand movements. We follow the experimental setup proposed in [3], performing the classification of 14 hand movements using a 10-channel EMG signal. In order to do so, we employ the same preprocessing used by the authors of [3]. Our starting RF for this task has $N = 24$ and $D = 12$.

Finally, **UniMiB-SHAR** [24] is a Human Activity Recognition (HAR) dataset featuring a tri-axial accelerometer signal collected from a sensor mounted on a smartphone. The recorded motions belong either to one out of 9 daily-life activities (e.g. walking, sitting, etc.) or one out of 8 kinds of falls. The signals are collected at 50 Hz, and already provided in fixed-size windows of 151 samples, centered around peaks. We keep the same preprocessing as proposed in [24], benchmarking our results on the AF-17 task, which is the one considering all the target classes in the dataset. We derive the adaptive RFs from a baseline with $N = 32$ and $D = 9$.

The three datasets refer to tasks with a significant difference in the level of complexity, ranging from a binary classification (ECG5000) to a 17-classes one (UniMiB-SHAR). Accordingly, the time and energy associated with the accumulation of output probabilities during inference vary significantly, which influences our policies' overheads, as explained in Section 5.3. As shown in the following section, however, our approach remains effective even in conditions far from ideal ($M \approx D$). Additionally, after benchmarking the RFs both with raw data and simple embedded-friendly features extracted in the time domain, we always achieve higher accuracy with the former. Therefore, we report for all the three datasets results obtained using raw data as input.

Due to the class imbalance of all three datasets, we always report the scoring metric proposed in [24], i.e., the top-1 macro-average accuracy. All results are reported on each dataset's test set.

We deploy all RFs on PULPissimo[8], a 32-bit single-core RISC-V MCU belonging to the PULP family of architectures. Specifically, we refer to a 22nm realization of PULPissimo running at 205 MHz and equipped with 520 KB of L2 memory [13]. We estimate the inference clock cycles using a virtual platform [38], deriving the energy values from [13]. Concerning the software stack, we train the Random Forests using the open-source package scikit-learn [29] in Python 3.8. The inference phase uses the MCU-oriented C language implementation described in Section 5.2 both for the baseline and adaptive classifiers.

We compare the proposed approach with a static RF, the standard Max/SM policies evaluated on the last DT (as proposed in [28, 37]), and the QWYC method [39]. Concerning the latter, we limit the comparison to the binary ECG5000 task, since as mentioned in Section 3, QWYC is only benchmarked on binary problems. Independently on the early stopping criterion, the baseline models have been derived from the RFs with the N and D reported above for each dataset.

6.2 Hardware-Independent Results

Since all previous works on adaptive inference for RFs have only been evaluated in theoretical terms, without any real deployment at the edge, we perform a first hardware-independent comparison.

To this end, we consider the *average number of trees executed per inference* as a metric to quantify the complexity of the various techniques. This is a reasonable proxy for the time and energy consumption of inference, especially for a single-core platform (such as an MCU) that executes weak learners sequentially. Of course, this evaluation is unable to factor in the additional overhead introduced by the evaluation of the early stopping policy, thus possibly favoring accurate yet complex mechanisms to stop the inference. Thus, these results are meaningful under the assumption that evaluating a single weak learner has a significantly higher complexity than evaluating the early stopping criterion.

Figures 5-9 report the results of this experiment. Specifically, they report Pareto fronts obtained by the various considered techniques in terms of accuracy versus the average number of DTs per inference (N.Trees). In case of adaptive methods, different points of the curve, when present, are obtained by varying the early stopping threshold (α in Eq. 3 and 4). Furthermore, all graphs also report, as a comparison baseline, the results obtained with a static RF. In this case, different points refer to ensembles with progressively fewer weak learners (i.e., decreasing N), which have been retrained from scratch each time.

State-of-the-art Comparison. Figure 5 compares one of our proposed policies (the Aggregated SM) with the standard SM applied to the last executed model (as in [9, 28, 37]), and with a static RF. Additionally, for the binary ECG5000, we also report the results obtained with QWYC, both with and without the static ordering of the DTs. We do not apply tree batching yet.

For all three datasets, the Aggregated Score Margin with $B = 1$ lies on the Pareto front, often outperforming both other adaptive approaches and static

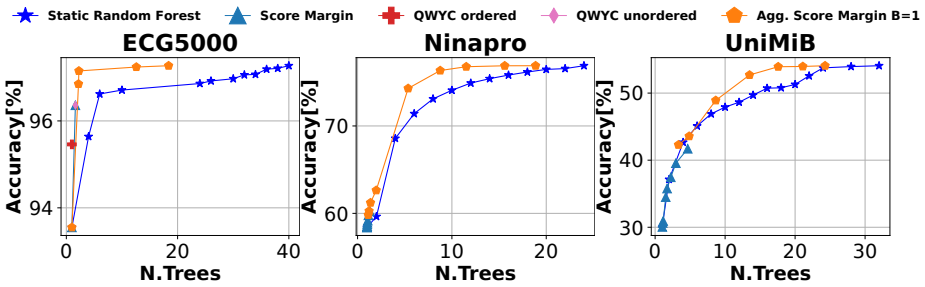

Fig. 5: Accuracy versus average number of trees. Each point represents either a different static RF for the baseline or the same RF with different early-stopping thresholds for adaptive ones.

RFs. On the other hand, the classic SM computed only on the last tree either obtains close results to the baseline or is underperforming. The only notable exception is represented by the ECG5000 dataset, where with few DTs the classic SM is able to achieve results comparable to our method. Nonetheless, that technique is unable to further grow in terms of prediction quality when changing the early stopping threshold.

Both QWYC versions, lie close to the global Pareto front. However, we found that even when testing several values of the hyperparameters that determine ϵ (the parameter used to decide for early stopping in QWYC), the average number of trees executed remains almost unchanged. Most importantly, the maximum accuracy that we were able to obtain with QWYC on ECG500 is significantly lower than with our approach, or with the largest static RF. Additionally, we found that the DT sorting proposed in QWYC actually underperforms on our dataset, leading to lower accuracy than the "unordered" version.

Considering the whole set of trade-off points of our approach, we obtain a reduction in terms of average trees executed per inference of up to 93% on ECG5000, with respect to a static RF achieving the same accuracy (2.26 vs 34 DTs on average, at 97% accuracy. On Ninapro, we achieve up to 47% reduction (10.47 vs 20 average DTs at 76.5% accuracy), and on UniMiB up to 43% (12.5 vs 22 DTs at 52% accuracy).

Batch Size Exploration and Criteria Comparison. Figures 6, 7, 8, 9 report a detailed comparison of the two proposed metrics (Aggregated SM and Aggregated Max.) for different tree batching conditions (i.e., B values).

Intuitively, since these results still do not consider the overheads of the early stopping criterion, increasing B should worsen the results. In fact, $B = 1$ theoretically offers a finer granularity of control on the early stopping, allowing to interrupt an inference just after executing the *first* DT that makes the aggregated SM or Max. overcome the threshold α. This is indeed what happens on average, as shown by the fact that curves relative to larger B values come closer to the static RF ones. However, it is not a hard rule, since the random sampling

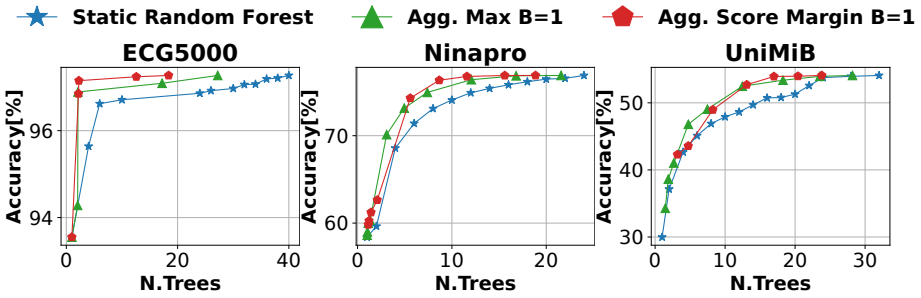

Fig. 6: Hardware-independent comparison of the two proposed metrics for $B = 1$.

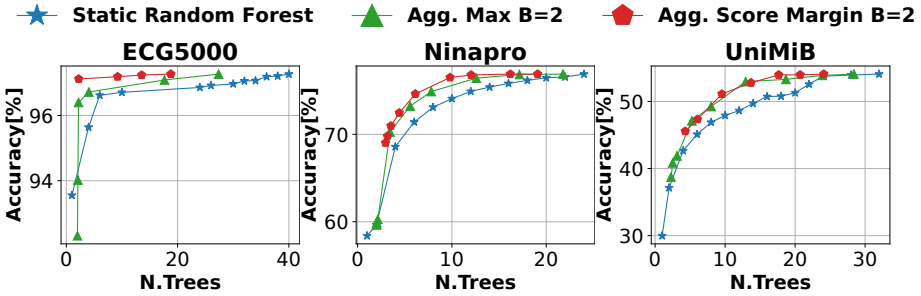

Fig. 7: Hardware-independent comparison of the two proposed metrics for $B = 2$.

and feature selection used to train the DTs can lead to a non-monotonically increase in prediction quality when adding weak learners. For instance, for the UniMiB dataset, the Aggregated Max with $B = 8$ obtains the largest reduction in the average number of DTs without accuracy drop with respect to the complete static RF (20.82 trees on average with +0.2% accuracy). On the contrary, Ninapro shows the expected results, with the Aggregated SM with $B = 1$ yielding the least average DTs for the same accuracy as the static RF (18.73).

Table 1 reports the detailed results of this comparison. Specifically, we show the average number of trees executed by the different variants of the adaptive inference policy, for two different accuracy conditions, i.e., to reach iso-accuracy with the original RF (Drop 0.0%) or allowing a negligible degradation (Drop 0.5%) The *Red.RF* column reports smallest static RF obtaining the same accuracy. Since the standard Score Margin and QWYC only achieved accuracy values with drops larger than 0.5% with respect to the original RF, they are not reported in the table.

On the ECG dataset we are able to reduce the average number of trees by 57% (17.18 vs 40) with no accuracy loss. Concerning the Ninapro dataset, the proposed approach can reduce the number of weak learners by 22% (18.73 vs 24), while on UniMiB by 35% (20.82 vs 32).

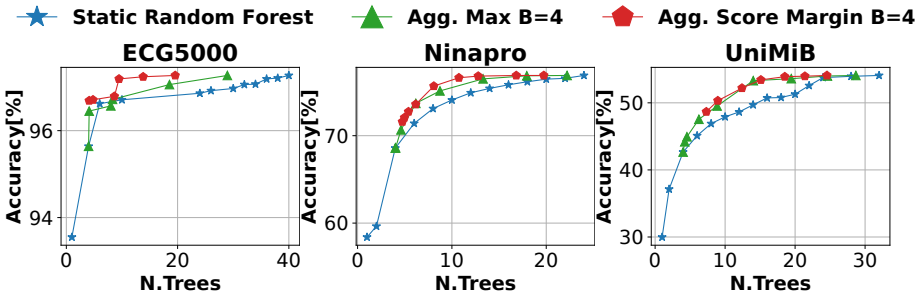

Fig. 8: Hardware-independent comparison of the two proposed metrics for $B = 4$.

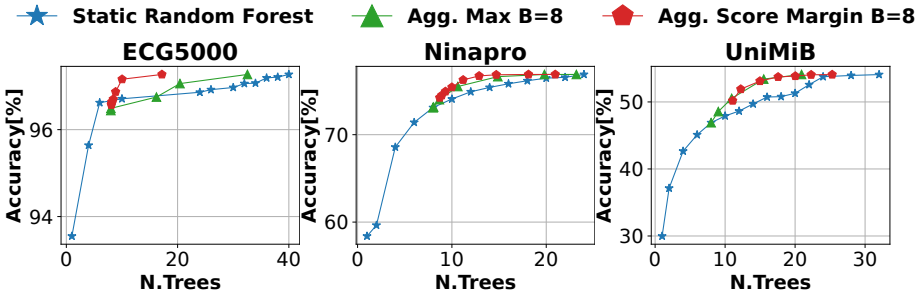

Fig. 9: Hardware-independent comparison of the two proposed metrics for $B = 8$.

When accepting an accuracy drop of 0.5%, we achieve a reduction in the average DTs executed of 91% with respect to the closest RF (Red. RF column) on the ECG dataset (2.1 vs 24 DTs). On Ninapro, we avoid the execution of 51% weak learners (9.73 vs 20) while for UniMiB of 29% (17.02 vs 24).

6.3 Tree-Ordering Analysis

As anticipated in Section 5.4, our results demonstrate that an optimized hard-coded ordering of DTs to favour early exit does not provide practical advantages. A first indication of this is shown in Figure 5, where QWYC with optimized tree ordering performs significantly worse than the randomly ordered one in terms of accuracy, for a negligible reduction in the number of invoked trees.

A further confirmation is provided in Figure 10. To generate it, we shuffled the DTs of the original RF 20 times at random. For each ordering, we then compared the early-stopping results on the validation and on the test set of each dataset. Specifically, we selected an α threshold so that the accuracy drop is 0% with respect to the static RF (as done in Table 1) and we then extracted the average number of DTs executed with that threshold on the two data subsets. We considered the Aggregated SM policy and a batch $B = 1$ for this experiment.

Two interesting results appear from the figure. First, tree ordering could ideally play a significant role in the early stopping effectiveness. In fact, the average

Table 1: Average number of trees for different accuracy drops with respect to a full RF.

Data	Full RF	Red. RF	Aggr. Max B=1	Aggr. Max B=2	Aggr. Max B=4	Aggr. Max B=8	Aggr. SM B=1	Aggr. SM B=2	Aggr. SM B=4	Aggr. SM B=8
Drop: 0%										
ECG	40	40	27.22	27.38	28.95	32.55	18.39	18.78	19.58	**17.18**
Ninapro	24	24	21.52	21.72	22.14	23.07	**18.73**	18.95	19.62	20.94
UniMiB	32	32	28.41	28.6	28.94	**20.82**	24.21	24.46	24.97	25.85
Drop: 0.5%										
ECG	40	24	**2.1**	17.63	18.54	20.43	2.19	2.23	8.64	8.86
Ninapro	24	20	12.02	12.56	13.26	14.83	11.55	**9.73**	10.69	12.83
UniMiB	32	24	23.76	24.08	19.34	20.82	**17.02**	17.64	18.41	17.43

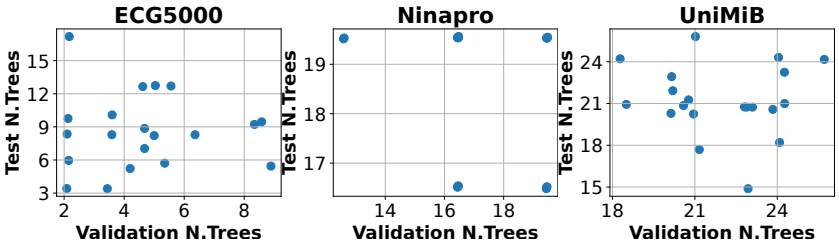

Fig. 10: Average number of DTs executed with the aggregated SM policy to reach the same accuracy as the original static RF on the Validation and Test sets respectively. Each point represent a different ordering of weak learners.

number of DTs executed on the full test set varies by up to ±15 depending on the weak learners' permutation. However, obtaining the optimal ordering on a *different data subset* (in our case, the validation set) does not work, as evident by the lack of correlation in the scatter plots.

The "optimal ordering" must therefore be computed dynamically based on the processed input. How to do so while keeping a low overhead will be subject of our future work.

6.4 Deployment Results

In this section, we report the results obtained with the proposed adaptive inference method when deployed on the target edge device. Figure 11 shows the Pareto fronts in terms of accuracy versus average energy consumption per inference on PULPissimo. For each dataset, we report the results of static RFs with different numbers of weak learners, as well as both our proposed early-stopping policies (Aggregated Max Score and Aggregated SM), with two batch sizes ($B = 1$ and $B = 2$). Differently from Section 6.2, here energy results include also the overheads for evaluating the early-stopping policies.

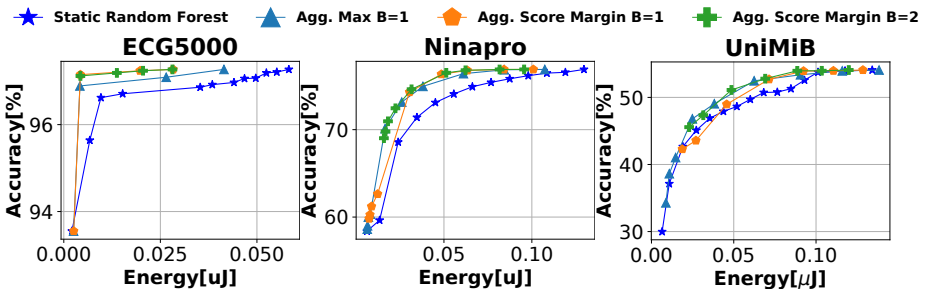

Fig. 11: Accuracy versus average energy per inference.

Indeed, as expected, while the curves are similar to the ones reported in Figure 5, the early stopping overhead becomes visible. This brings the adaptive approach closer to the baseline. Nonetheless, our proposed method still significantly outperforms static RFs. In detail, at iso-accuracy with a static RF, we obtain energy savings up to 26% on the UniMiB dataset, up to 91% on ECG5000 and up to 45% on Ninapro.

Table 2 reports the detailed energy results on each dataset, under the same conditions described in Table 1. While the top-performing approaches are similar to the hardware-independent case, some notable exceptions occur. For instance, on Ninapro, the aggregated Score Margin with $B = 2$, although requiring slightly more trees on average, requires less energy than the one with $B = 1$. This becomes even more evident for $B = 4$, requiring 0.67 additional trees on average than $B = 2$, while "costing" only 0.02 nJ more. Regarding the UniMiB dataset, the aggregated Score Margin with $B = 1$ requires the least amount of trees, however, it has a higher cost in terms of energy than all the other batched versions. Globally, these results show once again that properly accounting for the early stopping policy overheads is fundamental in order to assess the *real* effectiveness of an adaptive inference method.

Table 2: Average energy consumption, in nJ, for different accuracy drops with respect to a full RF.

Data	Full RF	Red. RF	Aggr. Max				Aggr. SM			
			B=1	B=2	B=4	B=8	B=1	B=2	B=4	B=8
Drop: 0%										
ECG	58.27	58.27	41.46	40.87	44.17	47.68	28.31	28.14	30.15	**25.77**
Ninapro	129.64	129.64	106.85	104.05	104.5	108.24	100.64	**95.5**	95.52	99.42
UniMiB	134.15	134.15	138.32	133.24	130.2	**96.33**	128.5	119.91	115.16	117.76
Drop: 0.5%										
ECG	58.27	35.32	**4.17**	26.68	28.95	31.21	4.3	4.29	13.66	13.59
Ninapro	129.64	108.54	61.32	61.76	64.23	71.27	63.56	**50.56**	53.52	62.29
UniMiB	134.15	101.21	115.92	112.52	89.18	96.33	92.41	88.61	87.69	**82.28**

7 Conclusions

In this work, we have presented an adaptive inference approach for RFs on MCUs, based on executing only a subset of the weak learners in order to save energy. To control this early-stopping mechanism, we have proposed two different lightweight policies which use the class probabilities produced in output by DTs to estimate the partial prediction confidence. In order to validate our approach, we have performed extensive experiments on three state-of-the-art datasets concerning popular embedded tasks. Moreover, we have deployed the proposed method on a single-core RISC-V MCU, showing that even when taking into account the overhead associated with the evaluation of the early stopping policy, we are able to save significant energy with respect to a static model, up to more than 90% for the same accuracy.

References

[1] Davide Anguita et al. "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine". In: *International workshop on ambient assisted living*. Springer. 2012, pp. 216–223.

[2] Manfredo Atzori et al. "Building the Ninapro database: A resource for the biorobotics community". In: *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE. 2012, pp. 1258–1265.

[3] Manfredo Atzori et al. "Electromyography data for non-invasive naturally-controlled robotic hand prostheses". In: *Scientific data* 1.1 (2014), pp. 1–13.

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[6] Alessio Burrello et al. "Embedding Temporal Convolutional Networks for Energy-Efficient PPG-Based Heart Rate Monitoring". In: *ACM Trans. Comput. Healthcare* 3.2 (Mar. 2022). ISSN: 2691-1957. DOI: 10.1145/3487910. URL: https://doi.org/10.1145/3487910.

[7] Yanping Chen et al. "A general framework for never-ending learning from time series streams". In: *Data mining and knowledge discovery* 29.6 (2015), pp. 1622–1664.

[8] Francesco Conti et al. "PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision". In: *Journal of Signal Processing Systems* 84.3 (2016), pp. 339–354. ISSN: 1939-8115. DOI: 10.1007/s11265-015-1070-9. URL: https://doi.org/10.1007/s11265-015-1070-9.

[9] Francesco Daghero et al. "Energy-Efficient Adaptive Machine Learning on IoT End-Nodes With Class-Dependent Confidence". In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. 2020, pp. 1–4. DOI: 10.1109/ICECS49266.2020.9294863.

[10] Francesco Daghero et al. "Energy-efficient deep learning inference on edge devices". In: *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*. Ed. by Shiho Kim and Ganesh Chandra Deka. Vol. 122. Advances in Computers. Elsevier, 2021, pp. 247–301.

[11] Francesco Daghero et al. "Adaptive Random Forests for Energy-Efficient Inference on Microcontrollers". In: *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. 2021, pp. 1–6.

[12] Francesco Daghero et al. "Ultra-compact binary neural networks for human activity recognition on RISC-V processors". In: *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 2021, pp. 3–11.

[13] Alfio Di Mauro et al. "Always-on 674μ W@ 4GOP/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.11 (2020), pp. 3905–3918.

[14] Cristian Donos, Matthias Dümpelmann, and Andreas Schulze-Bonhage. "Early Seizure Detection Algorithm Based on Intracranial EEG and Random Forest Classification". In: *International Journal of Neural Systems* 25.05 (2015), p. 1550023. DOI: 10.1142/S0129065715500239.

[15] Lin Fan, Zhongmin Wang, and Hai Wang. "Human Activity Recognition Model Based on Decision Tree". In: *Proceedings of the 2013 International Conference on Advanced Cloud and Big Data*. CBD '13. USA: IEEE Computer Society, 2013, pp. 64–68. ISBN: 9781479932610. DOI: 10.1109/CBD.2013.19. URL: https://doi.org/10.1109/CBD.2013.19.

[16] Tianshi Gao and Daphne Koller. "Active Classification based on Value of Classifier". In: *Advances in Neural Information Processing Systems 24*. Ed. by J Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 1062–1070. URL: http://papers.nips.cc/paper/4340-active-classification-based-on-value-of-classifier.pdf.

[17] Angelo Garofalo et al. "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2164 (Feb. 2020), p. 20190155. DOI: 10.1098/rsta.2019.0155. URL: https://doi.org/10.1098/rsta.2019.0155.

[18] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[19] Daniele Jahier Pagliari et al. "Dynamic Bit-width Reconfiguration for Energy-Efficient Deep Learning Hardware". In: *Proceedings of the International Symposium on Low Power Electronics and Design*. ISLPED '18. New York, NY, USA: ACM, 2018, 47:1–47:6. ISBN: 978-1-4503-5704-3. DOI: 10.1145/3218603.3218611. URL: http://doi.acm.org/10.1145/3218603.3218611.

[20] Daniele Jahier Pagliari et al. "Sequence-To-Sequence Neural Networks Inference on Embedded Processors Using Dynamic Beam Search". In: *Electronics* 9.2 (2020). ISSN: 2079-9292.

[21] Daniele Jahier Pagliari et al. "CRIME: Input-Dependent Collaborative Inference for Recurrent Neural Networks". In: *IEEE Transactions on Computers* (2020), p. 1. ISSN: 1557-9956 VO -. DOI: 10.1109/TC.2020.3021199.

[22] Daniele Jahier Pagliari et al. "Input-Dependent Edge-Cloud Mapping of Recurrent Neural Networks Inference". In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218595.

[23] Oded Z Maimon and Lior Rokach. *Data mining with decision trees: theory and applications*. Vol. 81. World scientific, 2014.

[24] Daniela Micucci et al. "Unimib shar: A dataset for human activity recognition using acceleration data from smartphones". In: *Applied Sciences* 7.10 (2017), p. 1101.

[25] John Mingers. "An empirical comparison of pruning methods for decision tree induction". In: *Machine learning* 4.2 (1989), pp. 227–243.

[26] Pavlo Molchanov et al. "Pruning Convolutional Neural Networks for Resource Efficient Inference". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJGCiw5gl.

[27] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. "Conditional Deep Learning for Energy-Efficient and Enhanced Pattern Recognition". In: *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, pp. 475–480. ISBN: 9783981537062.

[28] Eunhyeok Park et al. "Big/little deep neural network for ultra low power inference". In: *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2015, pp. 124–132. ISBN: 978-1-4673-8321-9. DOI: 10.1109/CODESISSS.2015.7331375. URL: http://ieeexplore.ieee.org/document/7331375/.

[29] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[30] Joao Pereira and Margarida Silveira. "Learning representations from healthcare time series data for unsupervised anomaly detection". In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2019, pp. 1–7.

[31] F Samie et al. "From Cloud Down to Things: An Overview of Machine Learning in Internet of Things". In: *IEEE Internet of Things Journal* 6.3 (2019), pp. 4921–4934. ISSN: 2327-4662 VO - 6. DOI: 10.1109/JIOT.2019.2893866.

[32] Alexander G. Schwing et al. "Adaptive random forest — How many "experts" to ask before making a decision?" In: *CVPR 2011*. 2011, pp. 1377–1384. DOI: 10.1109/CVPR.2011.5995684.

[33] W Shi et al. "Edge Computing: Vision and Challenges". In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. ISSN: 2327-4662 VO - 3. DOI: 10.1109/JIOT.2016.2579198.

[34] SiFive. *SiFive Core IP*. 2021. URL: https://www.sifive.com/risc-v-core-ip.

[35] STMicroelectronics. *iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*. Website. www.st.com/resource/en/datasheet/lsm6dsox.pdf. 2019.

[36] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329. ISSN: 00189219. DOI: 10.1109/JPROC.2017.2761740. arXiv: 1703.09039.

[37] Hokchhay Tann et al. "Runtime configurable deep neural networks for energy-accuracy trade-off". In: *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis - CODES '16*. 2016, pp. 1–10. ISBN: 9781450344838. DOI: 10.1145/2968456.2968458. arXiv: arXiv:1508.06655v1. URL: http://dl.acm.org/citation.cfm?doid=2968456.2968458.

[38] The PULP Platform. *GVSOC: PULP Virtual Platform*. 2020. URL: https://github.com/pulp-platform/gvsoc.

[39] Serena Wang et al. "Quit When You Can: Efficient Evaluation of Ensembles by Optimized Ordering". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 17.4 (2021), pp. 1–20.

[40] Z Zhou et al. "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing". In: *Proceedings of the IEEE* 107.8 (2019), pp. 1738–1762. ISSN: 1558-2256 VO - 107. DOI: 10.1109/JPROC.2019.2918951.