A Memory Representation of Random Forests Optimized for Resource-Limited Embedded Devices

Justin Beaurivage, Messaoud Ahmed Ouameur, *Senior Member, IEEE*, and Frédéric Domingue

***Abstract*—Random forests are a versatile and effective machine learning technique widely applied across various tasks. With the increasing demand for deploying machine learning models on resource-constrained embedded devices, such as microcontrollers, challenges arise from the growing complexity of modern datasets. These challenges often result in models that are too large in memory and storage requirements to be feasibly implemented on small devices.**

In this work, we propose a lossless memory representation of random forests that significantly limits the amount of random-access memory (RAM) required for prediction tasks, while also reducing the amount of non-volatile memory needed to store the model. The approach achieves efficiency by embedding the data of leaf nodes within the decision nodes, thereby streamlining the tree structure. Additionnally, it supports in-place prediction without requiring a decompression step.

To evaluate our method, we implemented four random forests derived from real-world datasets onto four microcontroller platforms. Our results demonstrate that prediction tasks can be performed using at most 144 bytes of RAM for classification tasks, and at most 48 bytes for regression tasks, while memory accesses account for a maximum of 27.0% of the total CPU cycles. On the fastest platform, prediction times ranged between 59 and 75 μs, highlighting the suitability of this method for a variety of real-time applications.

***Index Terms*—Random forests, resource-limited systems, edge computing, embedded devices**

I. Introduction

Random forests (RF) are a powerful tool used in a variety of machine learning tasks. They are widely popular for their robustness, interpretability, and ease of implementation. Due to their simplicity, random forest models may be used to perform real-time predictions on resource-constrained devices deployed on the edge [1]. Yet, as modern prediction tasks become increasingly complex, model sizes also expand, often growing too large for deployment on resource-limited embedded devices. However, there are still significant advantages to performing machine learning tasks on edge devices: more useable data and actionable insights, reduced data transfer, reduced latency, and reduced power consumption for connected end devices [2]. There is therefore a demand to develop methods allowing resource-scarce devices, such as low-end microcontrollers, to perform predictions on larger and more complex machine learning models.

Some works have already shown that random forests can be compressed to occupy significantly less storage space than their uncompressed counterparts. Nonetheless, these compression methods often have limitations: some are lossy [3], meaning that the forest may yield different prediction outcomes after compression. Others can only compress random forests in which all trees have identical structures (homogeneous forests). Heterogeneous forest structures can however help further reduce the model size by yielding prediction accuracies comparable to larger, heterogeneous structures [4]. It has been shown that random forests for regression tasks also typically have poorer compression ratios than those meant for classification tasks [5]. Most methods found in the literature also typically aren't optmized for deployment on resource-constrained devices: the compressed forests may need to be decompressed prior to being used for prediction tasks. Some work has also gone into reducing the amount of RAM and memory accesses needed to train forests and perform predictions [6]. However we find that these methods still require much more RAM than what is available on certain embedded devices, which can be as low as 2 KiB [2].

In this paper, we propose a lossless, memory-efficient representation of random forests, optimized for deployment on resource-scarce embedded devices. With the understanding that these devices are often extremely limited in the amount of RAM and NVM available, we focus on minimizing RAM necessary to perform predictions, while simultaneously reducing the non-volatile storage space needed to store the model. A secondary objective is reducing the prediction latency.

The method works by encoding the prediction outcomes directly in the decision nodes, thus eliminating the need for distinct leaf nodes; the forest is then stored in a flat data structure. This method allows for in-place prediction, thereby removing the need for a decompression step, while avoiding unaligned memory accesses, and reducing the RAM and NVM requirements. It is also characterized by a deterministic prediction time, making it suitable for some real-time applications. This method also supports random forests with heterogeneous tree structures [4].

II. Proposed memory representation

We consider a random forest as schematized in Fig. 1. It is composed of $\{t_1, t_2, \cdots, t_N\}$ trees, where each tree t_n contains $\{u_{n,1}, u_{n,2}, \cdots, u_{n,M_n}\}$ nodes, and where M_n is the

Justin Beaurivage and Frédéric Domingue are with the Laboratory for Technology, Innovation, and Performance in Sports (L-TIPS), and Messaoud Ahmed Ouameur is with the Laboratory of Signal and System Integration (LSSI). All authors are affiliated with Université du Québec à Trois-Rivières (UQTR), Canada (e-mail: justin.beaurivage@uqtr.ca; messaoud.ahmed.ouameur@uqtr.ca; frederic.domingue@uqtr.ca).

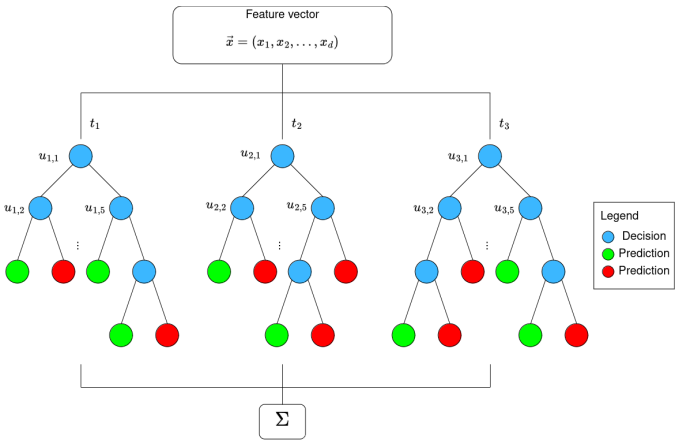


Fig. 1: A typical random forest.

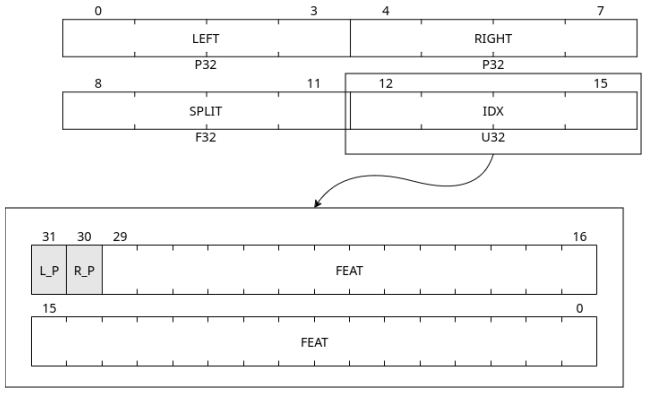


Fig. 2: Memory layout of a decision node, and bit layout of the IDX bitfield

number of nodes within the tree t_n. We also consider the feature vector $\vec{x} = (x_1, x_2, \cdots, x_d)$.

Building the memory representation of this forest is accomplished in two steps: First, we define a memory representation of a single node, in which the potential prediction outcomes are encoded. Then, we define a method to sequentially store all the nodes in the forest.

A. Step one: Decision nodes

Each node contains 4 fields: LEFT, RIGHT, SPLIT, and IDX. The field layout is depicted in Fig. 2. The IDX bitfield determines the type of the LEFT and RIGHT fields. If IDX.L_P $= 1$, then LEFT contains a prediction (ie, a leaf). For classification problems, it contains the index of the predicted class. For regression problems, it contains the prediction in 32-bit floating-point format. Otherwise, if IDX.L_P $= 0$, LEFT contains a relative pointer to the next decision node in the tree. Similarly, if IDX.R_P $= 1$, RIGHT contains a prediction, and a pointer to the next decision node otherwise.

Together, SPLIT and IDX.FEAT determine whether the left or right branch is considered. SPLIT contains a 32-bit floating point number which represents the threshold to obtain a decision. IDX.FEAT contains the index i of the decision feature x_i. Thus if SPLIT $\leq x_i$, the left branch is taken. Otherwise if SPLIT $> x_i$, the right branch is taken.

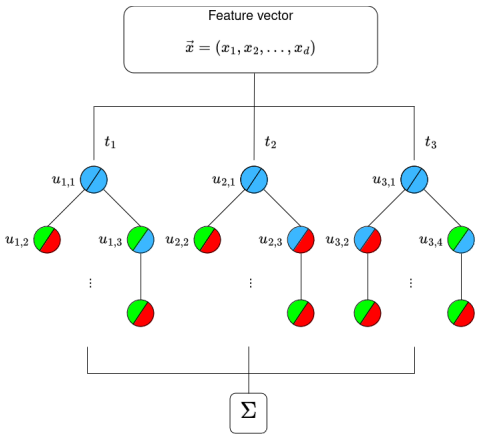


Fig. 3: Random forest using our node layout.

By encoding the predictions in the decision nodes, we eliminate the need for dedicated leaf nodes. The forest depicted in Fig. 1 becomes much simpler, as illustrated in Fig. 3.

Note that in this example, all the node's fields are 4 bytes wide; however this layout may easily be adjusted to allow for a smaller storage utilization, at the cost of smaller maximum tree and feature vector sizes.

B. Step two: Store the trees in memory

Next, we propose a method to arrange the forest's nodes into a flat, array-like structure. This is achieved in 3 steps:

1) Insert the first node of each tree at the beginning of the array, in tree order:

$$\{u_{1,1}, u_{2,1}, \cdots, u_{N,1}\}$$

2) Append the remaining nodes in tree order, then in node order:

$$\{u_{1,2}, u_{1,3}, \cdots, u_{1,M_1}, u_{2,2}, \cdots, u_{2,M_2}, \cdots, u_{N,2}, \cdots, u_{N,M_N}\}$$

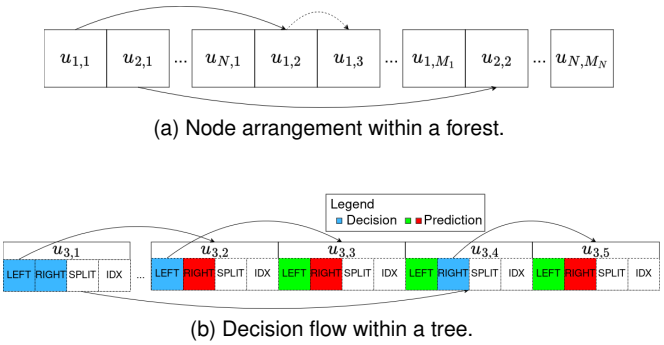
3) Finally, for each node that contains a pointer to another node, adjust the pointer such that it points to the correct node.

Fig. 4 shows the node arrangement as described in this section. Note that the method makes no assumptions with respect to the individual trees' structures – one such assumption might have been be that all trees share identical structures. It therefore becomes trivial to use this method to represent random forests with heterogeneous structures.

An open-source reference implementation is available at https://github.com/L-tips/embedded-random-forest/.

III. Experimental results and discussion

In this section, we implement our forest representation on four microcontroller platforms. Table I outlines the platforms used for this experiment. The RP2350 chip is a dual-core, dual-architecture microcontroller; we performed our benchmarks on both architectures using only a single core. To mitigate the significant performance penalty of storing the forest model on external flash, and given the RP2350's relatively abundant



(a) Node arrangement within a forest.

(b) Decision flow within a tree.

Fig. 4: Memory representation of a random forest

TABLE I: Hardware platforms used in benchmarks

Chip name	Microchip ATSAMD21 [8]	Microchip ATSAMD51 [9]
Architecture	ARM® Cortex-M0+	ARM® Cortex-M4 with hardware float
CPU frequency	48 MHz	120 MHz
Onboard RAM	32 KiB	192 KiB
Onboard flash memory	256 KiB	512 KiB

Chip name	Raspberry Pi RP2350 [10] (ARM, single-core)	Raspberry Pi RP2350 (RISC-V, single-core)
Architecture	ARM® Cortex-M33	RV32IMAC
CPU frequency	150 MHz	150 MHz
Onboard RAM	520 KiB[a]	520 KiB[a]
Onboard flash memory	None[b]	None[b]

[a] Supports up to 8 MiB of optional external PSRAM
[b] Up to 16 MiB of external flash

RAM, the models were moved to RAM for the prediction speed benchmarks, In contrast, the other chips use onboard flash memory, which incurs much smaller access delays.

We use four real-world datasets to solve two classification and two regression problems. Three of the datasets used are publicly available on Kaggle [7]. The fourth dataset, *Skydive*, is a collection of sensor readings gathered by a wearable tracker during 22 freefall skydiving jumps.

The forests were trained using the `randomForest` R package [11], each containing 100 trees with a maximum of 50 nodes per tree; the remaining parameters were left as default. Table II shows some general information concerning the datasets used to train each model, as well as the proportion of nodes removed by our method when compared to a standard RF implementation, such as the one depicted in Figure 1.

TABLE II: Comparison of forests trained over different datasets

Dataset	**Problem type**	**# features, # classes**	**Nodes pruned**
Skydive	Classification	5, 9	53.2%
Iris	Classification	4, 3	50.5%
Bike Sharing	Regression	8, –	50.5%
Airfoil Self Noise	Regression	5, –	50.5%

TABLE III: CPU cycles spent on memory accesses

	Skydive	**Iris**	**Bike Sharing**	**Airfoil Self Noise**
SAMD51	13.4%	12.1%	13.9%	10.6%
RP-ARM	27.0%	16.5%	19.9%	24.1%

A. *Benchmark results*

Each forest was then benchmarked in terms of processing speed and RAM usage on all four platforms (Figure 5), using the Rust compiler's `"s"` optimization level and `"fat"` link-time optimizations [12]. CPU cycles spent on memory accesses were also measured on the SAMD51 and RP-ARM platforms – the only two platforms with hardware support for precise instruction-level profiling.

RAM usage during prediction is notably low: across all platforms, our method requires at most 144 bytes for the 9-class *Skydive* classification problem, and 48 bytes for regression problems. On the SAMD51 platform, less than 13.9% of CPU time is spent on performing memory I/O, the being remainder used to perform the actual predictions. On RP-ARM, a comparatively larger share of CPU time is spent on memory, due to its more efficient computations, shifting the bottleneck toward memory access. Regression forests also use less RAM than classification forests, as they compute a simple running average of all the tree outcomes instead of tallying votes across classes.

While RAM usage is not determined by the forest's size, it increases with the number of classes. It is used both for storing the votes for each outcome (in classification problems), and for saving CPU registers when entering the prediction subroutine. The latter can be reduced through inlining optimizations, which were disabled in this study for consistent benchmarking. Likewise, differences in RAM usage across platforms is explained by the number of available CPU registers, and the presence of a hardware floating point unit (FPU), both of which reduce temporary RAM usage.

RP-ARM achieved the fastest prediction speeds among all platforms. While it shares the same clock speed, RAM, and execute-in-place cache as RP-RISCV, its more capable architecture—including an FPU and digital signal processing (DSP) instructions—makes it far better suited for high-speed prediction. In contrast, RP-RISCV's simpler, area-optimized design and lack of such features significantly reduce its prediction throughput. On RP-ARM, prediction times ranged from 59 µs to 75 µs.

B. *Comparison with other methods*

We have also implemented a standard random forest algorithm on the SAMD51 plaform as a baseline on which to compare our findings (Table IV). We found that a standard RF uses more RAM than our method for classification problems, while it uses 8 fewer bytes for regression problems. Our method indeed requires the use of 2 additional CPU registers to save the left and right node types (ie, decision or prediction) in temporary variables. Therefore the state of these registers must be saved in RAM when entering the prediction subroutine. This increase in memory usage could easily be mitigated by using compiler inlining optimizations.

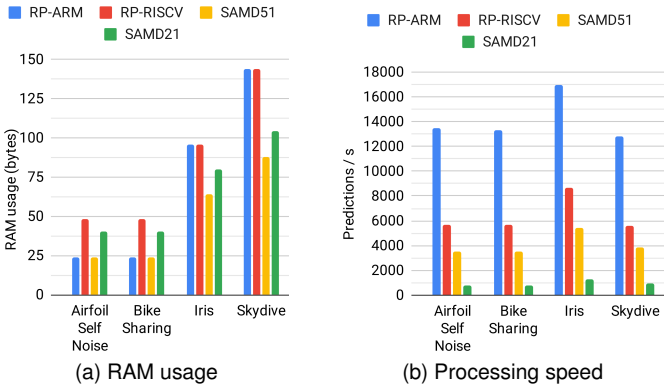


Fig. 5: Benchmark results for prediction tasks

TABLE IV: Comparisons with other RF methods

	Skydive	Iris	Bike Sharing	Airfoil Self Noise
RAM usage, in bytes (SAMD51)				
Ours	**88**	**64**	**24**	**24**
Standard RF	120	72	16	16
	(+36%)	(+13%)	(-33%)	(-33%)
Prediction speed, in predictions/s (SAMD51)				
Ours	**3892**	**5411**	**3517**	**3511**
Standard RF	2376	4403	2469	2451
	(-39%)	(-19%)	(-30%)	(-30%)
Total forest size, in KiB				
Ours	**57.42**	**8.63**	**76.56**	**76.56**
Standard RF	193.34	30.70	193.34	193.34
	(+236%)	(+256%)	(+153%)	(+153%)
Boosted RF [13]	57.52	8.72	72.13	72.13
	(+0.17%)	(+1.0%)	(-5.8%)	(-5.8%)

Our method also offers significant prediction speed gains over the standard RF implementation. The reduced total number of nodes in the forest also contributes to cutting the non-volatile memory required to store the forest by more than 60% when compared to a standard RF algorithm. Finally, we have also compared the NVM storage requirements with the Boosted Random Forest [13], which uses a similar memory representation as our proposed method. It uses a custom hardware architecture, implemented on a field-programmable gate array (FPGA). We find that the NVM storage requirements are quite similar between the two methods. Yet, while the memory representation proposed in [13] has the potential to be used in regression forests, the Boosted RF implementation itself can only be used for classification problems.

IV. Conclusions

In this work we suggest a method for representing random forests in memory. This representation is particularly well suited for resource-limited devices, such as microcontrollers. In particular, the represented forests use a relatively small amount of storage, with compression ratios comparable to the Boosted Random Forest [13], and an improvement of more than 60% over a standard random forest algorithm. Moreover, carrying out prediction tasks using the proposed method requires an almost-negligible amount of random-access memory, rarely exceeding 100 bytes in typical use cases. On the RP2350 microcontroller, sub-100 µs prediction times are highly achievable, even with relatively large forests – of the order of 100 trees with 50 decisions each – making this method useable for a variety of real-time classification and regression problems. Prediction times also show a significant improvement over a standard RF. We have shown that heterogeneous forests can trivially be represented, making this method quite flexible. The forest representation is also lossless: the compressed forest will generate identical outcomes as its uncompressed counterpart.

While the focus of this work was largely aimed at limiting the RAM required to perform predictions to a minimum while also reducing the amount of NVM space needed, further work could be carried out to improve the prediction speed of the proposed algorithm. In particular, the prediction tasks could be parallelized on multi-core processors, thus providing a simple and effective way of significantly reducing prediction times.

Acknowledgments

The authors would like to thank Théo Duville for his work on the *Skydive* dataset, as well as all those who participated in the data collection.

References

[1] F. Küppers, J. Albers, and A. Haselhoff, "Random Forest on an Embedded Device for Real-time Machine State Classification," in *2019 27th European Signal Processing Conference (EUSIPCO)*, Sep. 2019, pp. 1–5, iSSN: 2076-1465.

[2] S. Branco, A. G. Ferreira, and J. Cabral, "Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey," *Electronics*, vol. 8, no. 11, p. 1289, Nov. 2019, number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

[3] A. Painsky and S, Rosset, "Compressing Random Forests," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec. 2016, pp. 1131–1136, iSSN: 2374-8486.

[4] M. Gashler, C. Giraud-Carrier, and T. Martinez, "Decision Tree Ensemble: Small Heterogeneous Is Better Than Large Homogeneous," in *2008 Seventh International Conference on Machine Learning and Applications*, Dec. 2008, pp. 900–905. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4796917

[5] A. Painsky and S. Rosset, "Lossless Compression of Random Forests," *Journal of Computer Science and Technology*, vol. 34, no. 2, pp. 494–506, Mar. 2019.

[6] C. Slimani, C.-F. Wu, S. Rubini, Y.-H. Chang, and J. Boukhobza, "Accelerating Random Forest on Memory-Constrained Devices Through Data Storage Optimization," *IEEE Transactions on Computers*, vol. 72, no. 6, pp. 1595–1609, Jun. 2023, conference Name: IEEE Transactions on Computers.

[7] "Find Open Datasets and Machine Learning Projects." [Online]. Available: https://www.kaggle.com/datasets

[8] Microchip, "SAM D21 Family Data Sheet," 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-D21DA1-Family-Data-Sheet-DS40001882G.pdf

[9] Microchip, "SAM D5x/E5x Family Data Sheet," 2020. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/SAM_D5xE5x_Family_Data_Sheet_DS60001507F.pdf

[10] Raspberry Pi, "RP2350 Datasheet," 2024. [Online]. Available: https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf

[11] L. Breiman, A. Cutler, A. Liaw, and M. Wiener, "Breiman and Cutlers Random Forests for Classification and Regression," Jan. 2022. [Online]. Available: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[12] The Rust Project, *The rustc book*, May 2025. [Online]. Available: https://doc.rust-lang.org/1.87.0/rustc/

[13] Y. Mishina, R. Murata, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi, "Boosted Random Forest," *IEICE Transactions on Information and Systems*, vol. E98.D, no. 9, pp. 1630–1636, 2015.