Memory-efficient Random Forests in FPGA SmartNICs

Andrea Monterubbiano*
monterubbiano@di.uniroma1.it
Sapienza University
Rome, Italy

Raphael Azorin*
raphael.azorin@huawei.com
Huawei Technologies Co. Ltd
Paris, France
EURECOM
Biot, France

Gabriele Castellano
gabriele.castellano@huawei.com
Huawei Technologies Co. Ltd
Paris, France

Massimo Gallo
massimo.gallo@huawei.com
Huawei Technologies Co. Ltd
Paris, France

Salvatore Pontarelli
pontarelli@di.uniroma1.it
Sapienza University
Rome, Italy

Dario Rossi
dario.rossi@huawei.com
Huawei Technologies Co. Ltd
Paris, France

ACM Reference Format:
Andrea Monterubbiano, Raphael Azorin, Gabriele Castellano, Massimo Gallo, Salvatore Pontarelli, and Dario Rossi. 2023. Memory-efficient Random Forests in FPGA SmartNICs. In *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies (CoNEXT Companion '23), December 5–8, 2023, Paris, France.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3624354.3630089

1 INTRODUCTION

Random Forests (RF) have been a popular Machine Learning (ML) algorithm for more than two decades. This success can be attributed to its simplicity, effectiveness and explainability. However, implementing them in a high-speed programmable data plane is not trivial. To make predictions, i.e., inference, RFs must traverse each tree from the root to the leaf by comparing the features vector at each split node. This process is particularly challenging in network devices where memory is limited, and packet processing cannot be delayed, i.e., predictions occur at line rate. Nevertheless, this implementation is crucial for incorporating recent ML advances in the network, which could benefit use cases such as scheduling, measurements, and routing [1]. Prior studies such as Planter [4] have examined the implementation of RF in network switches, mapping trees to Match-Action Tables (MAT). Another line of work focused on RF implementations optimized for FPGA, mapping tree layers to pipeline stages as done in [2]. Such approaches use different tree representations that naturally come with their strengths and weaknesses depending on the trees' sparsity, depth, and input features. In this work we *(1)* propose a novel representation for FPGA-based Random Forests, *(2)* compare it against state-of-the-art implementations in terms of memory and computation requirements, and *(3)* evaluate our design on a flow classification task using CAIDA traffic traces.

*Authors contributed equally.

CoNEXT Companion '23, December 5–8, 2023, Paris, France

Approach	Computation	Advantageous trees' characteristics memory-wise
MAT	$O(T)$	sparse, shallow, few features
Dense	$O(T \times D)$	full, deep, many features
Hybrid	$O(T \times D)$	full in the first layers then sparse, deep, many features

Table 1: RF implementations time and space complexity with T trees with D as maximum depth.

2 APPROACHES

In this work, we focus on Random Forest models trained for binary classification tasks with binary features as input. First, we provide a brief introduction to the *MAT* RF implementation. Next, we present a *dense* RF implementation that maps tree nodes to specific memory locations. Finally, we detail our novel *hybrid* RF implementation that exploits the trees' sparsity to provide an alternative compact representation when other approaches are inadequate.

Match-action table representation. MATs have been used to implement Decision Trees (DT) in [3]. In our binary classification case with binary input features, a DT can be implemented using a single MAT featuring one rule for each leaf of the tree. In each row, the matching key corresponds to the concatenation of the feature bits, while the action corresponds to the predicted label. This approach scales well for Random Forests [4], where each additional tree in the forest only requires an extra MAT. The memory requirement (in terms of number of bits) of a RF is then:

$$Mem_{MAT} = \sum_{t \in RF} L_t (2F + 1), \tag{1}$$

where t is a tree of the RF, L_t the number of leaves in t, F the number of features, and the factor of two is due to the fact that encoding a ternary value i.e., "0", "1", or "don't care", for each feature requires 2 bits.

When restricted to binary leaves, a simple yet efficient optimization consists in encoding solely the leaves that correspond to one of the two classes, capturing the other with a default miss condition in the MAT (cf. Figure 1). In particular, if one of the classes is vastly under-represented in the tree's leaves, this approach can significantly shrink the model size, as only the leaves corresponding to the less frequent class are stored in the MAT.

Dense tree representation. An alternative approach to implement a RF in an FPGA, is to map each tree to its own memory block

Features					Label
0	1	X	0	X	1
1	0	0	X	1	1
X	X	X	X	X	0

Figure 1: MAT representation

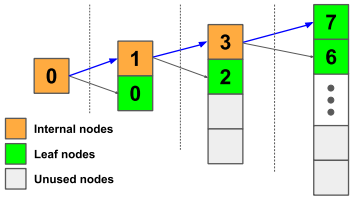

Figure 2: Dense tree representation

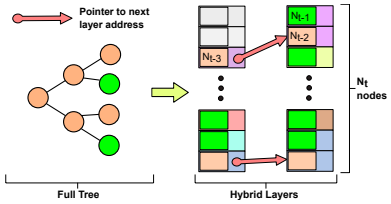

Figure 3: Hybrid tree representation

which are then accessed sequentially as in [2]. For each DT, each node inside a layer is numbered sequentially, starting from 0 to $2^k - 1$, where k is the layer depth starting from $k = 0$ at the root, as depicted in Figure 2. Then, each DT is executed as follows: given any node at position i in layer k, its children in the next layer $k + 1$ can be found at the memory locations $2i$ and $2i + 1$.

In our binary classification case with binary input features, each node is *only* required to store *(i)* one flag bit to distinguish between the two types of nodes (split or leaf); and *(ii)* the index of the feature to compare against (if it is a split node) or the predicted class label (if it is a leaf node). The memory required in bits for RF is thus:

$$Mem_{Dense} = \sum_{t \in RF} (2^{D_t} - 1)(\lceil log_2(F) \rceil + 1), \tag{2}$$

with t being a tree of the RF, D_t the depth of tree t and F the number of binary features. We remark that, while this implementation can be more efficient in terms of memory occupancy than MAT in some cases, it has the drawback of requiring more memory accesses.

Hybrid tree representation. The dense tree representation may waste a significant amount of memory in the case of a sparse tree that is far from being a full binary tree. Thus, we propose a hybrid tree approach that encodes a tree using a dense representation for the first layers and an *indexed encoding* for the deeper layers.

We identify the maximum number of nodes N across all the layers of all trees in the forest. For the first $M = \lfloor log_2(N) \rfloor + 1$ layers, we use the dense tree implementation; while for the remaining deeper and sparser layers, we only allocate N nodes. This allocation calls for a more sophisticated addressing. For each of these sparse layers' nodes, we add a pointer to its left child in the next layer. Its right child is found at this position plus 1. This implementation is sketched in Figure 3. The memory requirement in bits for a Random Forest RF can now be expressed as:

$$Mem_{Hybrid} = \sum_{t \in RF} \underbrace{(2^M - 1)(\lceil log_2(F) \rceil + 1)}_{\text{dense M-tree representation}} + \underbrace{N(D_t - M)(\lceil log_2(F) \rceil + 1 + \lceil log_2(N) \rceil)}_{\text{indexed encoding}} \tag{3}$$

with t being a tree of the RF, F the number of binary features and D_t the depth of tree t.

3 EVALUATION

In Table 1, we qualitatively compare the three approaches in terms of computation complexity and report their most favorable scenario from a memory perspective. Naturally, the memory footprint of each approach depends on the trees characteristics. From the previous equations, we conclude that our *hybrid* approach is more memory-efficient than the *dense* approach when:

$$\sum_{t \in RF} \text{indexed enc.} < \sum_{t \in RF} (2^{D_t} - 2^M)(\lceil log_2(F) \rceil + 1). \tag{4}$$

Similarly, *hybrid* is more memory-efficient than *MAT* when:

$$\sum_{t \in RF} \text{indexed enc.} < \sum_{t \in RF} L_t(2F + 1) - \sum_{t \in RF} \text{dense M-tree}. \tag{5}$$

Note that these implementations require leaf nodes to store the class label (1 bit) rather than the class probability (1 float). The final RF output corresponds to a majority vote across trees rather than an average of class probabilities. Therefore the probability threshold needs to be hard-coded in each leaf. We evaluate the performance impact of this change with a binary flow classification task: separating elephants from mice flows. We use a public 1-hour CAIDA TCP traffic trace from 2016-01-21 at 1 PM. Each flow is labeled as elephant (top 1% sizes) or mice (bottom 99% sizes). The RF takes as input 96 binary features which correspond to the flow 4-tuple in binary format, omitting protocol. The RF is trained and pruned offline on the first 5 minutes of the trace and tested on the last 5 minutes, with a probability threshold of 0.04 to classify approx. 1% of the flows as elephants. The model F1-score decreases by only 1.07% when hard-coding the threshold in the leaves vs. the untouched vanilla Random Forest. Using a *hybrid* representation, we report 4.4× and 48.8× memory savings vs. the *MAT* and *dense* approaches respectively.

4 FUTURE WORK

The main limitation of this work is that it only supports binary features. While this restriction can naturally accommodate categorical variables (thanks to, e.g., one-hot encoding), it would require some pre-processing to deal with numerical floating point quantities (e.g., with quantization). In this case, we plan to thoroughly study the trade-offs at play in terms of memory and performance. Additionally, we plan to explore the implementation of Gradient Boosting Decision Trees (e.g., XGBoost, CatBoost). Unlike Random Forests, these ensemble models are additive, meaning that each tree contributes to the final outcome through summation. Their implementation in FPGA is challenging because the quantities added are typically highly sensitive floats.

REFERENCES

[1] Đukić et al. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *16th USENIX Symposium NSDI'19*. 565–580.
[2] Elnawawy et al. 2020. FPGA-based network traffic classification using machine learning. *IEEE Access* 8 (2020), 175637–175650.
[3] Lee et al. 2020. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications* (2020), 1–12.
[4] Zheng et al. 2021. Planter: seeding trees within switches. In *Proceedings of the SIGCOMM'21 Poster and Demo Sessions*. 12–14.