Journal of Parallel and Distributed Computing 157 (2021) 220–232

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

# RF-RISA: A novel flexible random forest accelerator based on FPGA

Shuang Zhao, Shuhui Chen [*], Hui Yang, Fei Wang, Ziling Wei

*School of Computer, National University of Defense Technology, Chang Sha, 410073, China*

A R T I C L E  I N F O

*Article history:*
Received 13 January 2021
Received in revised form 11 May 2021
Accepted 3 July 2021
Available online 21 July 2021

*Keywords:*
Hardware accelerator
Random forest
Prediction acceleration
FPGA

A B S T R A C T

Recently, FPGA has been utilized to accelerate the Random Forest prediction process to meet the speed requirements of real-time tasks. However, the existing accelerators impose restrictions on the parameters of the accelerated model. The accelerators have to be reconfigured to adapt to a model whose parameters exceed the predefined restrictions. When these accelerators are applied in the scenarios where the model updates or switches frequently, non-trivial time overhead and maintenance costs may be introduced. To solve the above problem, a flexible accelerator RF-RISA, Random Forest Reduced Instruction Set Accelerator, is presented in this paper. Compared with the existing accelerators, RF-RISA eliminates all the restrictions by decoupling the model parameters from its hardware implementation. Specifically, RF-RISA encodes the information of the model into a group of instructions, then the instructions are stored in the memory rather than are hardcoded in the hardware. Meanwhile, a mapping scheme is proposed to map the instructions into the memory dynamically. Finally, a new hardware architecture is designed to support the pipelined computing. The theoretical analysis and experimental results show that the proposed RF-RISA can accelerate a wide range of RF models without reconfiguration. At the same time, it can achieve the same throughput as the state-of-the-art.

## 1. Introduction

Random Forest (RF) is a classical ensemble learning algorithm. It has the advantages of interpretability, simplicity, and high accuracy. Although deep learning has been got high expectations in recent years, RF is still widely chosen in various research areas. For example, object detection and tracking [30,4,24], network traffic classification [36,12], etc. It is trained with the "bagging" method to build multiple independent binary decision trees (DT), then the trees are merged to get better prediction performance. With the increase in data size, many real-time tasks require high prediction speed. However, the software implementation of the RF prediction process cannot meet the requirement even if the multi-threading technology is utilized. Hence, attention has been paid to the hardware-based RF prediction accelerators.

A few works have accelerated DT or RF on multiple hardware platforms. Brian et al. [39] compare the effectiveness of three different platforms for accelerating RF, including FPGA, GP-GPUs, and multi-core CPU. Among those platforms, FPGA provides the best performance. GP-GPU offers a more flexible solution than FPGA since the FPGA implementation cannot adapt to the evolving model. Although multi-core CPU provides the simplest solution, its performance is significantly slower than the GP-GPU and FPGA. Tong et al. [38] and Sebastian et al. [6] both show that the RF accelerator based on FPGA has a faster prediction speed compared with other platforms. Hiroki et al. [23] give reasons why GPU is not the best choice for the RF acceleration, such as a high cost for the all-to-all communications. Other works [16,29,31] further prove that the FPGA-based RF accelerator could achieve remarkable performance. Therefore, FPGA has become the preferred platform for accelerating RF.

According to the adopted architecture, the existing FPGA-based RF accelerators can be categorized into two types: comparator-centric (Comp-C) accelerators and memory-centric (Mem-C) accelerators [19]. The Comp-C accelerator implements the RF model as a threshold network that consists of a layer of threshold logic units and a layer of combinational logic units [1,8,14]. Such accelerators can achieve high throughput with low latency. The Mem-C accelerator accelerates the RF model by introducing the pipeline in layers of the tree [22,28,37]. It can achieve as much throughput as the Comp-C accelerator after full pipelining. However, both types of accelerators lack flexibility. For the Comp-C accelerator, it hardwires the calculations of the RF model into the hardware. As a result, one Comp-C accelerator can only adapt to one model. Any changes in the model would require FPGA reconfiguration. As for the Mem-C accelerator, its implementation has restrictions on the parameters

[*] Corresponding author.
*E-mail addresses:* zhaoshuang16@nudt.edu.cn (S. Zhao), shchen@nudt.edu.cn (S. Chen), huihui19870124@126.com (H. Yang), wangfei09a@nudt.edu.cn (F. Wang), weiziling@nudt.edu.cn (Z. Wei).

https://doi.org/10.1016/j.jpdc.2021.07.001
0743-7315/© 2021 Elsevier Inc. All rights reserved.

of the RF model, including the number of trees, the tree depth, and the number of nodes in each layer. When the above parameters of a model fall outside the restricted ranges, a Mem-C accelerator has to be reconfigured to adapt to the model. In practical applications, an RF model usually updates frequently due to the modification of classification targets and training datasets. Its new parameters may exceed the restrictions. In that case, both types of accelerators would suffer from significant time overhead and maintenance costs introduced by frequent reconfiguration. More detailed information on the above two types of accelerators will be covered in Section 2.2.

Based on the above two architectures, the existing accelerators aim at improving the throughput and reducing the latency. Their flexibility is ignored. A few studies have paid attention to the scalability of their solutions [16][26]. However, those solutions are implemented based on the Mem-C architecture. They are unable to get rid of the restrictions brought by the architecture itself. Therefore, the flexibility gain is limited. We identify this research gap and aim at providing a flexible solution while maintaining high throughput. Thus, a wide range of RF models can be accelerated without reconfiguration. In practice, such a flexible accelerator is necessary, especially for those application scenarios that demand to switch or update models constantly.

Based on this consideration, we propose RF-RISA, an RF Reduced Instruction Set Accelerator based on FPGA inspired by the Reduced Instruction Set Computer (RISC). RF-RISA adopts a different representation of the RF model. Specifically, a software driver encodes the structure and node information of the RF model into a group of instructions. Meanwhile, a novel hardware architecture is designed to provide storage resources and support pipelined computing. Then, the instructions are mapped into the memory dynamically and are executed in parallel. All parameters of the RF model are transparent to the hardware until the instructions are executed in RF-RISA. As long as the allocated memory in FPGA is adequate for storing the generated instructions, any RF model can be accelerated swiftly by RF-RISA. The contributions of this paper are summarized as follows:

(i) A novel flexible RF accelerator RF-RISA is presented. As far as we know, RF-RISA is the first to utilize a new architecture that decouples the parameters of the RF model from its hardware implementation for RF acceleration.
(ii) A new representation of the RF model is proposed in RF-RISA. A set of instructions is designed to encode the node and structure information of the model. Through avoiding hard-coding the parameters of the model into the hardware, it eliminates the restrictions on the accelerated model.
(iii) A node mapping scheme is designed in RF-RISA to map the node information into the memory of the FPGA. Instead of allocating fixed storage space to each layer of the tree as the existing accelerators do, our scheme allocates storage for each layer on demand.
(iv) A new hardware architecture based on FPGA is proposed in RF-RISA. Compared with the existing architectures, our solution organizes the storage and computing resources in a different way. It enables the accelerator to have great flexibility and high throughput at the same time. Finally, comprehensive analysis and experiments are conducted to evaluate the effectiveness of the proposed RF-RISA.

The structure of the rest of the paper is organized as follows. Section 2 provides the necessary background. Then the proposed RF-RISA is described in section 3. Section 4 gives a detailed implementation and evaluation of RF-RISA. Discussion and related work are presented in section 5 and section 6, respectively. Finally, section 7 concludes the paper.

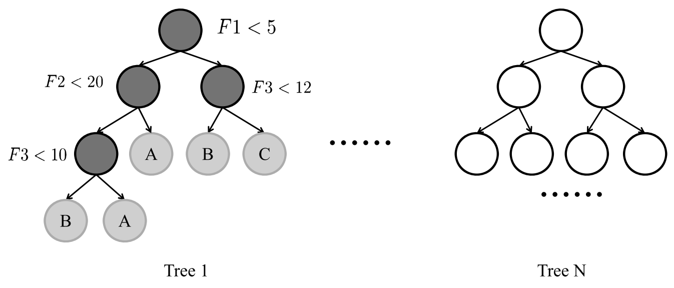


**Fig. 1.** An RF model with $N$ DTs. $Fj$ is the $j$th attribute. A, B, C are class labels.

## 2. Background

### 2.1. Random forest

The RF model is in the form of an ensemble of independent binary DTs [5]. Its training algorithm offers several parameters that closely relevant to the structure of the generated RF model. One parameter is the maximum tree depth, which limits the depth of the generated trees. Another parameter specifies the number of trees in the model. The other parameter controls the criterion of node splitting, which will affect the number of nodes in each layer. The optimal values of these parameters are usually determined by the grid search during training. When a model is retrained with a different dataset, the optimal values of these parameters are likely to change. As a result, the structure of the retrained model may also change. Further details on the training process could be found in [5].

Fig. 1 gives an example of a trained RF model. Each tree includes two types of nodes: internal nodes and leaf nodes. An internal node contains a comparison test, consisting of a comparison attribute (e.g. *F1, F2, F3* in Fig. 1) and a constant threshold. Besides, each internal node has two child nodes. A leaf node is assigned a class label (e.g. A, B, C in Fig. 1). It can be seen that Tree1 includes four internal nodes (dark gray) and five leaf nodes (light gray). For a binary tree, the maximum number of nodes in the *ith* layer is $2^i$ assuming its root node is in layer 0.

To predict an instance, each tree classifies the instance from its root node independently. The forward branch of an internal node is determined according to the comparison result. The prediction of a tree is completed after reaching a leaf node. Then the label of the leaf node is the output of this tree for this instance. The final prediction of the RF model is obtained by majority voting.

### 2.2. FPGA-based DT accelerators

As mentioned previously, the accelerators that have been well researched can be divided into Comp-C accelerators and Mem-C accelerators. Note that the trees in the RF model are independent, the existing accelerators focus on how to accelerate one DT. Then the acceleration of an RF model can be achieved by accelerating all trees in parallel.

#### 2.2.1. Comparator-centric accelerator

The implementation of the Comp-C accelerator bases on the equivalence between DT and the threshold network. Such accelerators contain one threshold unit layer and one output layer. For DT, each internal node is implemented as a processing element(PE) in the threshold unit layer. Therefore, it is feasible to execute all nodes at the same time. In addition, the output layer consists of a group of boolean functions. Each boolean function is the ORing of all different paths that lead to the leaf nodes with the same label, and each path that leads to a leaf node is represented as an ANDing function. The Comp-C accelerator for Tree1 in Fig. 1 is illustrated in Fig. 2. To predict an instance, all PEs in the threshold

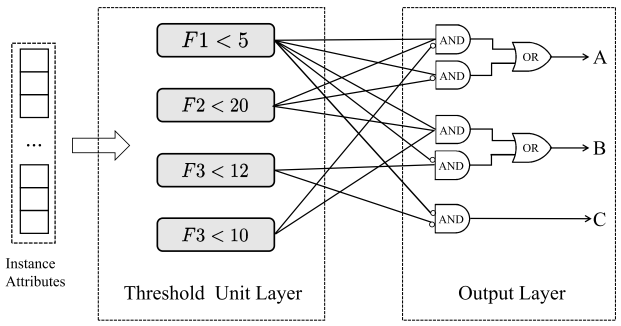


**Fig. 2.** Implementation of Tree1 in Fig. 1 using the Comp-C architecture.

unit layer perform the comparison tests in parallel. Afterward, the comparison results are sent to the output layer as input. Note that only one boolean function will be true, its label is the prediction of this tree accordingly.

The Comp-C accelerator could achieve high throughput and low latency since it is constructed in two layers. However, it requires a large number of computing resources. In addition, one accelerator can be only applied to one RF model since it needs all information of the model for its logic design. Any change in the accelerated model demands a reconfiguration of the accelerator. For application scenarios where the model updates frequently, the application of such an accelerator will introduce significant maintenance costs.

### *2.2.2. Memory-centric accelerator*

A fact of the RF model is that at most one node of each layer is visited during the prediction of an instance. On this account, the Mem-C accelerator realizes inter layer pipelining by processing different instances at different layers. Fig. 3 gives the implementation of the Mem-C accelerator for Tree1 in Fig. 1. As Fig. 3 shows, the internal nodes of one layer are mapped into one stage. Each stage shares one PE and one storage, hence different stages can process different instances at the same time. After pipelining, the accelerator can output one result per clock cycle.

The parameters that indicate the model structure should be defined before designing a Mem-C accelerator, including the number of trees, the maximum tree depth, and the number of nodes in each layer. The number of trees specifies how many trees are executed in parallel. The maximum tree depth determines how many stages should be designed. It is worth mentioning that a tree may not reach the maximum depth, while it should have stages with the maximum depth. By this means, all the prediction results of the trees can be synchronized after pipelining. Finally, the storage size for each stage is determined by the number of nodes in each layer. Although it can be allocated to hold $2^i$ nodes for the *ith* stage, a waste of storage resources would be caused when the tree is sparse.

An RF model can be accelerated by an implemented Mem-C accelerator when it satisfies the following restrictions: (1) the number of trees and the maximum tree depth of the model are not greater than the predefined values in the accelerator; (2) the storage of each stage is sufficient to contain the nodes of each layer. On the whole, the Mem-C accelerator provides more flexibility than the Comp-C accelerator since it can adapt to a batch of models. However, it still has restrictions on the structure of the applicable models. We denote the Mem-C accelerator as the state-of-the-art.

## *2.3. Motivation*

It can be seen that the high degree of coupling between the hardware architecture and the model parameters causes the poor flexibility of the existing accelerator. The Comp-C architecture is highly coupled with all parameters of the model. As regards the Mem-C architecture, it is decoupled from the nodes by storing the node information in its storage. However, it embeds the structural parameters of the model in its hardware.

**Table 1**
Notations used in this paper.

| Notation | Description |
|---|---|
| M | The number of BRAM allocated as memory |
| K | The size of one BRAM in bits |
| W | The length of the instruction |
| C | The number of class in the RF model |
| [ ] | Rounding down operation |

Hence, to improve the flexibility of the accelerator, the hardware implementation of the accelerator should be decoupled from the structural parameters of the model. To attain this objective, we plan to store the model structure in the memory as the node information. In contrast to being hardcoded into hardware, the structure information is obtained during the prediction. Meanwhile, the hardware (i.e., FPGA) plays the role of a platform that provides fast memory access and parallel computing. From this perspective, we find the expected role of FPGA is similar to that of the controller in RISC.

Consider a scenario that a piece of code is executed in RISC. First, the compiler compiles the code into machine-readable instructions. Then instructions are loaded into memory. Once the execution starts, the controller repeats two actions, i.e., fetching one instruction and performing the calculation, until all instructions are processed. For two different pieces of code, their instructions are likely to differ in number and the involved operations. However, the controller can execute both of them since it is responsible for fetching, decoding, and committing instructions.

The working principle of RISC inspires us to propose RF-RISA. To accelerate an RF model with RF-RISA, a set of instructions is first generated to encode the structure and node information of the model. Then the instructions are mapped into the memory of RF-RISA with a delicate design. Finally, the prediction is accelerated by executing instructions in the pipeline.

## 3. Method

### *3.1. RF-RISA framework*

The overall framework of RF-RISA is depicted in Fig. 4. As Fig. 4 shows, there are three phases in RF-RISA: (1) a software driver encodes the model into a group of instructions; (2) the instructions are loaded into the memory of the FPGA; (3) the FPGA executes the instructions. The memory supports simultaneous access for multiple instructions. The PEs are organized to support pipelining. Hence the prediction process can be accelerated.

To adapt to a new model, RF-RISA needs to perform its first two phases. The software driver compiles the new RF model, then the generated instructions are reloaded to the memory. Finally, FPGA executes the new instructions with the hardware unchanged.

The above three phases involve three key technologies proposed in RF-RISA: (1) a representation method for the RF model, i.e., an instruction set is used to represent the model; (2) a mapping scheme that maps the instructions into the memory; (3) a pipelined hardware architecture that accelerates the prediction process. Details about these technologies are given in the following sections. For clarity, Table 1 summarizes the notations used in this paper.

### *3.2. Instruction set*

An instruction set is designed to represent the RF model. There are three types of instructions: Node Operation Instruction

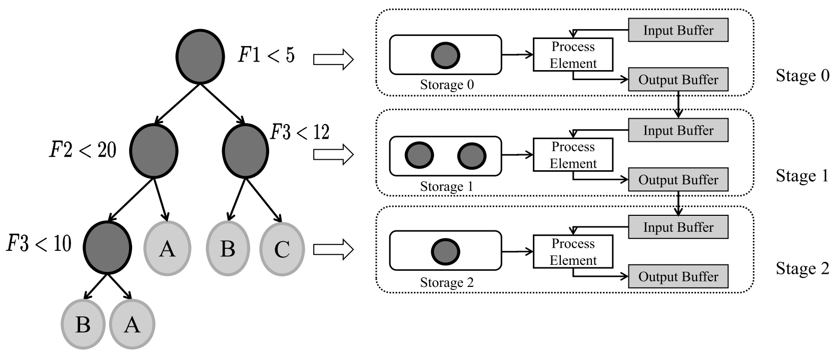
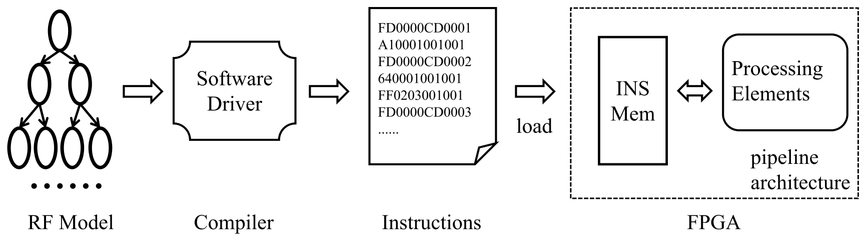


**Fig. 3.** Implementation of Tree1 in Fig. 1 using the Mem-C architecture.



```
FD0000CD0001
A10001001001
FD0000CD0002
640001001001
FF0203001001
FD0000CD0003
......
```

**Fig. 4.** The overall framework of RF-RISA.

| | | | | | | |
|---|---|---|---|---|---|---|
| Node Relative Address | Attribute ID | Threshold | Left Child Node Info | Right Child Node Info | Left Child Node Type | Right Child Node Type |

**Fig. 5.** The format of NOInst.

(NOInst), First Layer Instruction (FLInst), and Control Instruction (CTInst). NOInst and FLInst encode the node information and the structure information of an RF model, respectively. CTInst controls how the NOInst and FLInst are mapped into memory.

### 3.2.1. Node operation instruction

The NOInst encodes the information of an internal node. Fig. 5 gives the instruction format. Assuming that the length of a NOInst is W bits.

**Definition 1.** The node relative address (NodeRA) of an internal node refers to the relative address where its NOInst is stored in the BRAM. If there are $n$ NOInsts in front of the node's NOInst stored in the BRAM, then the NodeRA of the node is $n$.

In NOInst, NodeRA realizes fast addressing for mapping the NOInst into the BRAM. “AttributeID” and “Threshold” are the comparison attribute and the constant in the internal node. For “Left Child Node Type”, 0 means that the left child of this node is a leaf node, and the corresponding label is stored in “Left Child Node Info”. Otherwise, the left child node is an internal node. In this case, the value of “Left Child Node Type” implies the target BRAM where the left child node's NOInst is located. In contrast to recording the specific BRAM, the distance from the current BRAM to the target BRAM is recorded in this field. Meanwhile, “Left Child Node Info” stores the NodeRA of the left child node. Finally, the address of the left child node's NOInst can be inferred through these two fields. “Right Child Node Type” and “Right Child Node Info” work similarly for the right child node. Note that leaf nodes have no NOInst because their information has been encoded in the NOInst of their parent.

A NOInst is executed as follows. According to “AttributeID”, the attribute of the input instance is obtained and is compared with the constant in “Threshold”. If the value of the attribute is not

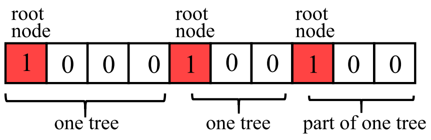


**Fig. 6.** An example of a FLInst.

greater than the constant, the execution continues based on “Left Child Node Type” and “Left Child Node Info”. Otherwise, the execution continues with “Right Child Node Type” and “Right Child Node Info”. Take the attribute value is smaller than the constant as an example. Suppose that “Left Child Node Info” is $a$. If “Left Child Node Type” is a non-zero integer $b$, then the next NOInst to be executed is stored in the subsequent BRAM whose distance from the current BRAM is $b$. The NodeRA is $a$. Otherwise, $a$ is the predicted label.

For simplicity, the term “node” is used to represent the node in the tree as well as its NOInst in the following sections. A specific meaning will be provided when it is necessary.

### 3.2.2. First layer instruction (FLInst)

A bit in a FLInst indicates whether a BRAM stores a root node of a tree. 1 means that the corresponding BRAM stores a root node. In other words, the nodes of one tree are stored from that BRAM as the beginning. An example of a FLInst with a length of 10 bits is depicted in Fig. 6.

According to this FLInst, it can be inferred that the involved 10 BRAMs store three trees. Furthermore, the first two trees occupy 4 BRAMs and 3 BRAMs, respectively. By connecting all the FLInsts, the number of trees and the BRAMs employed by each tree can be inferred. Therefore, the model structure could be encoded by FLInst. During implementation, the length of the FLInst is set to W bits to simplify the instruction decoding logic. If M BRAMs are

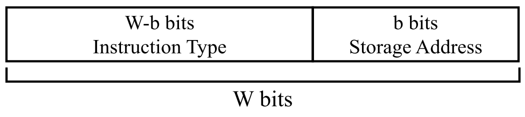


**Fig. 7.** CTInst format.

allocated to store the node information, [M/W]+1 FLInsts will be generated. Besides, the FLInst can be stored in registers due to its small data volume.

### *3.2.3. Control instruction (CTInst)*

In the second phase in RF-RISA, NOInsts and FLInsts of an RF model are mapped into the memory of FPGA. To indicate how the instructions are mapped into the memory, CTInst is added to the instruction set.

The instruction format of CTInst is shown in Fig. 7. The lower $b$ bits give the address where the subsequent NOInsts or FLInsts are stored. The remaining bits record the type of the subsequent instructions. While loading instructions, if a loaded instruction is a CTInst with the type “NOInst”, then the storage address gives the BRAM to be written. Thereafter, the subsequent NOInsts are written to that BRAM until the next CTInst is encountered. If the type is “FLInst”, a subsequent FLInst will be written into the register group given by the storage address. Similarly, the word length of CTInst is set to W bits.

## *3.3. Node mapping scheme*

This section presents the node mapping scheme used in RF-RISA. RF-RISA uses BRAM to store the node information. Suppose M BRAMs are available as the memory. The M BRAMs are organized as logically adjacent. The mapping scheme consists of the following four rules:

Rule 1. The NOInsts of the same layer of a tree are stored in BRAM in order from left to right.
Rule 2. The NOInsts of one layer can be stored in multiple adjacent BRAMs when one BRAM is inadequate to accommodate them.
Rule 3. The NOInsts of each layer of a tree are stored in continuous BRAM.
Rule 4. Each BRAM only stores the NOInsts of the same layer.

Among the above rules, Rule 1 and Rule 3 guarantee correct addressing of NOInsts. Rule 2 enables allocating the memory to each layer on demand. Hence there is no restriction on the number of nodes in each layer. Rule 4 ensures that the nodes of the different layers are stored in different BRAMs. Thus, the nodes in different layers can be accessed simultaneously. Lastly, the structure of the model is indicated by Rule 3 and the FLInsts.

According to the above rules, the NOInsts of an RF model are generated as Algorithm 1.

**Definition 2.** The node position of an internal node refers to its position in its layer. If one node is the $n$th internal node from the left to right in its layer, its node position is $n$. Leaf nodes do not participate in the counting.

In Algorithm 1, the mapping address of each node is calculated first, including the BRAM (i.e. *n*.ram_id) and the NodeRA (i.e. *n*.NodeRA) (lines 4-11). Then, the information of the child nodes of each internal node is calculated (lines 13-20). The attribute ID and threshold of the comparison of a node can be obtained directly from the model (line 21). Finally, as long as the final *Current_RAM_id* is not greater than M, the generated NOInsts of the RF model can be mapped into the memory of FPGA.

**Algorithm 1** NOInst generation.

```
Input: M, N(the maximum number of NOInst in one BRAM), T(RF model)
Output: NOInst for each internal node, The root node indicator V
 1: Current_BRAM_id = 0, V[M] = {0}
 2: for each tree t in T do
 3:     V[current_BRAM_id] = 1
 4:     for each layer i in t do
 5:         Needed_BRAM = [layer_node_num/N] +1 //layer_node_num is the number
    of nodes in layer i
 6:         for each internal node n do
 7:             n.ram_id = Current_BRAM_id + [n.position / N]
 8:             n.NodeRA = n.position % N
 9:         end for
10:         Current_BRAM_id += Needed_BRAM
11:     end for
12:     for each internal node n in t do
13:         if n.left_child is a leaf node then
14:             n.left_child_node_type = 0
15:             n.left_child_node_info = n.left_child.label
16:         else
17:             n.left_child_node_type = n.left_child.ram_id - n.ram_id
18:             n.left_child_node_info = n.left_child.NodeRA
19:         end if
20:         The processing of the right child node of n is similar as lines 13-18
21:         n.NOInst = [n.NodeRA, n.attribute_id, n.threshold, n.left_child_node_info,
    n.right_child_node_info, n.left_child_node_type, n.right_child_node_type]
22:     end for
23: end for
```

After the NOInsts of an RF model are generated, the FLInsts of the model can be generated according to the output root node indicator **V** in Algorithm 1. Finally, the CTInsts are generated according to the mapping scheme. The skeleton of the whole instructions of an RF model is illustrated in Fig. 8. Each layer of a tree would have at least one CTInst with the type of “NOInst” and a series of NOInsts. If a layer needs more than one BRAM, multiple groups of the CTInst with the type of “NOInst” and NOInsts would be generated. Finally, a CTInst with the type of “FLInst” and a FLInst appear in pairs with a total of [M/W]+1 times.

## *3.4. Hardware design*

RF-RISA adopts a hardware architecture that differs from the existing architecture. Fig. 9 provides the hardware design of RF-RISA.

As Fig. 9 shows, M BRAMs are organized as logically connected units to store the NOInsts. If one BRAM is inadequate for storing the NOInsts of one layer, the NOInsts can be placed in consecutive BRAMs (e.g. Layer $i$ in Fig. 9). Red rectangles near the BRAM denote the registers that store the data of the FLInsts. 1 means that the BRAM stores a root node. Since the NOInsts are mapped into the BRAMs dynamically, each BRAM owns a PE to support pipelining. Besides, buffers are set to store the intermediate results, such as the NOInst location and the predicted results.

Our architecture differs from the Mem-C architecture in the way how the memory and PEs are organized. The Mem-C architecture groups BRAMs into stages, which correspond to layers in the tree. The sizes of stages are determined in advance. Then each stage shares one PE. Hence the Mem-C architecture can be seen as an image of the tree to some extent. By contrast, each BRAM is an independent unit and owns a dedicated PE in RF-RISA. In the hardware implementation of RF-RISA, M is the only parameter that needs to be fixed in advance, and it is irrelevant to the RF model. Its value could be decided according to the chosen FPGA device and application scenarios. In this way, the concept of tree is removed in RF-RISA, which makes RF-RISA more flexible.

The hardware prediction process is given in Algorithm 2. To predict an instance, FPGA starts its execution from the first NOInst stored in BRAM 0. If the comparison result of the instruction points to an internal node, the location of the NOInst of the node is ob-

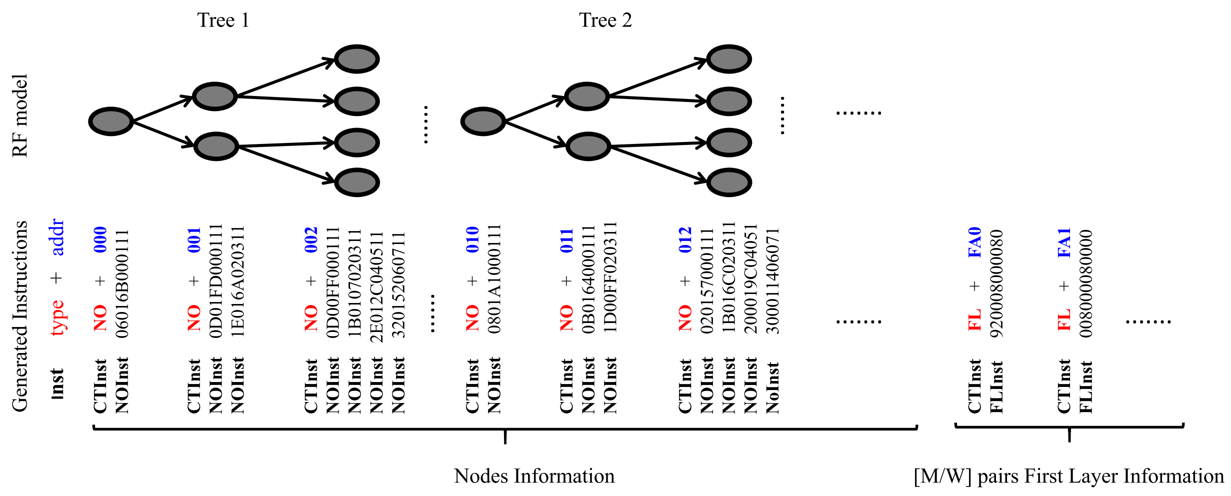


**Fig. 8.** The skeleton of the generated instructions of an RF model.

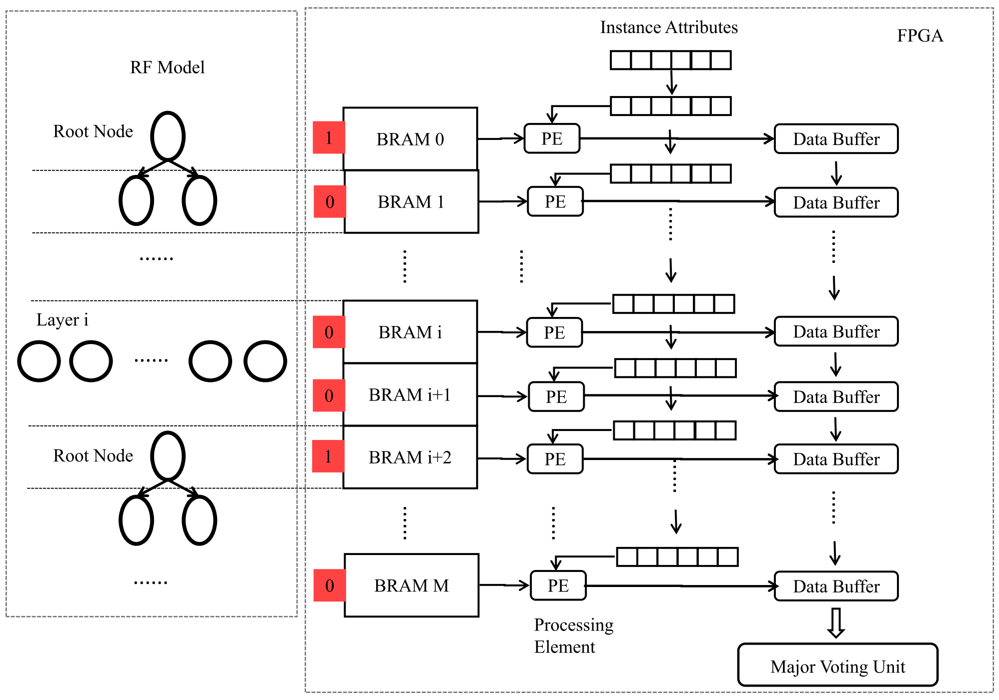


**Fig. 9.** The hardware architecture of RF-RISA. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

tained (lines 19-20). Then the subsequent BRAMs are accessed. No operation is performed unless the BRAM where the above NOInst is stored is reached (line 3-4). Otherwise, the current instruction output a label (line 13). Afterward, the subsequent BRAMs are accessed with no operation until a BRAM with a root node is reached (line 14-17). The prediction is finished after M BRAMs are accessed. Finally, the prediction result is given by the majority voting (line 23). After fully pipelining, one prediction per clock cycle can be achieved, and the prediction latency is M+1 clock cycles at least.

## 4. Evaluations and experiments

### 4.1. Implementation

A prototype is implemented to validate the performance of the proposed RF-RISA. The software driver is developed using C++ and the generated instructions are stored in the form of a txt file. The length of the instruction is set as 72 bits. The employed FPGA device is Arria 10-10AX115N2F4012SG which provides more than 2000 BRAMs with a size of 20 Kb. Quartus17.0 is used to develop the hardware.

Fig. 10 gives the circuit design of the PE in the hardware. The signal “i_rden” has three states to indicate what operations to perform in this PE. One state means that the next NOInst to be executed is not in this BRAM. Hence no calculation is performed. Another state indicates that fetching the NOInst from this BRAM according to the NodeRA and performing the comparison test. The last state implies the current tree already outputs a label, hence no calculation is performed. Besides, a PE will execute the first NOInst in the BRAM when the BRAM stores a root node.

### 4.2. Memory resource utilization

The memory resource utilization of RF-RISA and the existing Mem-C accelerator is analyzed and compared in this section. $\mathbf{T} = < t_1, t_2, \ldots, t_k >$ denotes an RF model with $k$ trees. The depth of the

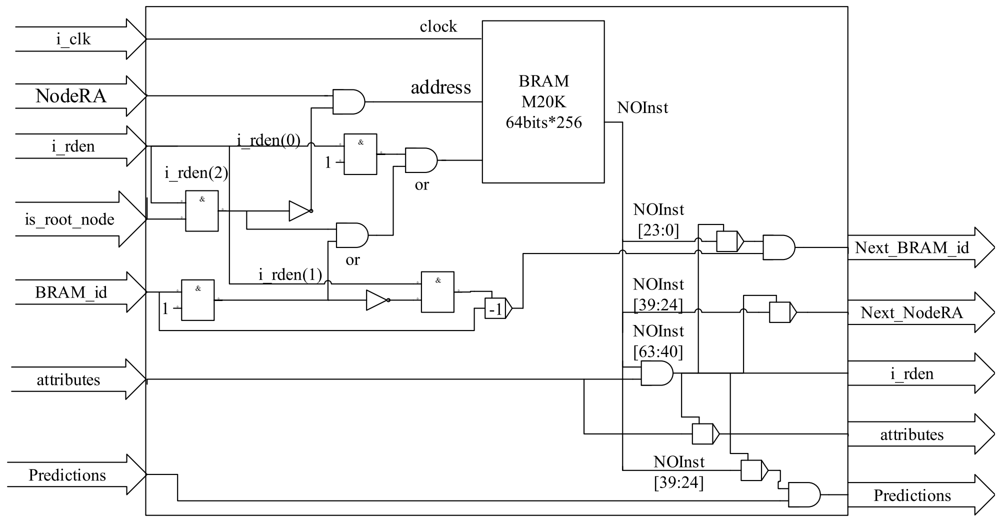


**Fig. 10.** The circuit design of the processing element.

**Algorithm 2** Prediction process.

```
Input: X:the attribute vector of an instance, C:the class number, M:the available number of BRAM
Output: Label
1: BRAM_id = 0, NodeRA= 0, result[C] = 0, i=0 // result records the times each label is predicted
2: while i <M do
3:     while i <BRAM_id do
4:         i += 1 // the NOInst to be executed is not in ith BRAM
5:     end while
6:     n: NOInst with address of BRAM_id and NodeRA
7:     if X[n.attribute_ID] <= n.threshold then
8:         direction = left
9:     else
10:        direction = right
11:    end if
12:    if n.direction_child_node_type == 0 then
13:        result[n.direction_child_node_info] += 1 // the child node is a leaf node
14:        while BRAM_id.root_node == 0 and BRAM_id <M do
15:            i += 1, BRAM_id += 1 // there is no root node stored in BRAM with address BRAM_id
16:        end while
17:        NodeRA = 0
18:    else
19:        BRAM_id += n.direction_child_node_type
20:        NodeRA= n.direction_child_node_info
21:    end if
22: end while
23: Output the Label with the maximum value in result
```

trees is represented by $\mathbf{D} = < d_1, d_2, ..., d_k >$. The maximum depth is $d_{max}(Max(\mathbf{D}))$. For tree $t_i$, the number of the internal nodes in each layer is denoted as $\mathbf{N} = < n_1^{t_i}, n_2^{t_i}, ..., n_{d_i}^{t_i} >$. For simplicity, the analysis is first launched on the premises of that both accelerators use BRAM to store the node information and each BRAM could store $m$ nodes.

For RF-RISA, Eq. (1) and Eq. (2) give the required number of BRAM to store each layer of a tree and the whole RF model. Similarly, Eq. (3) and Eq. (4) give the corresponding calculations for the existing Mem-C accelerator.

$$\left\lceil \frac{n_j^{t_k}}{m} + 1 \right\rceil \tag{1}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{d_i}(\left\lceil \frac{n_j^{t_k}}{m} \right\rceil + 1) \tag{2}$$

$$\left\lceil \frac{MAX(n_j)}{m} + 1 \right\rceil \tag{3}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{d_{max}}(\left\lceil \frac{MAX(n_j)}{m} \right\rceil + 1) \tag{4}$$

Obviously, Eq. (1) is always not greater than Eq. (3), hence Eq. (2) would not be greater than Eq. (4). The equality holds if and only if the depth of all trees is $d_{max}$, and the same number of BRAM is needed for the same layer of different trees. Therefore, the required minimum memory for storing the node information of RF-RISA is less than the existing Mem-C accelerator under the above premises.

However, one BRAM could store a different number of nodes in different accelerators, which causes the above premise may not hold. With different representations of the node, one node can be encoded in bits with different lengths. For example, the bit width of "Left Child Node Type" in RF-RISA is related to the number of the allocated BRAM. The bit width of "Left Child Node Info" is related to the number of nodes stored in one BRAM. In terms of the existing Mem-C accelerator, it can use 1 bit to represent the type of the child node. Since one stage may contain multiple BRAMs, it may need a wider bit width to encode the node address than RF-RISA. In this case, it is difficult to evaluate which accelerator uses fewer bits to encode one node unless a specific model is given. However, it is reasonable to infer that there is no architecture with evident advantage in memory utilization in all cases.

## 4.3. Flexibility analysis

The flexibility refers to the range of models that an accelerator can adapt to without reconfiguration in this paper. More models can be accelerated, more flexible the accelerator is. This section analyzes the flexibility of RF-RISA and the existing Mem-C accelerator theoretically.

Suppose that M BRAMs are allocated as the node storage. Based on the analysis in Section 4.2, the RF model that can be accelerated by RF-RISA and the existing Mem-C accelerators should satisfy Eq. (5) and Eq. (6), respectively. Obviously, all models that satisfy Eq. (6) would satisfy Eq. (5). On the contrary, the model that satisfy Eq. (5) may not satisfy Eq. (6), since Eq. (6) sets restrictions on k,**D**,**N**. Therefore, when the same storage is consumed, RF-RISA can adapt to a wider range of models than the existing Mem-C accelerators.

$$\sum_{i=1}^{k}\sum_{j=1}^{d_i}(\left\lceil \frac{n_j^{t_k}}{m} \right\rceil + 1) < M \tag{5}$$

$$\sum_{i=1}^{k}\sum_{j=1}^{d_{max}}\left(\left\lceil\frac{MAX(n_j)}{m}\right\rceil+1\right)<M,\ \text{k},\mathbf{D},\mathbf{N}\ \text{satisfy the limits} \tag{6}$$

### 4.4. Comparison with the state-of-the-art

The introduced Mem-C accelerator in Section 2.2.2 is implemented as the state-of-the-art. For its implementation, each node is represented with 48 bits. Meanwhile, the node storage of each stage is allocated with one BRAM in the following experiments.

#### 4.4.1. Flexibility

Section 4.3 has proved that RF-RISA is more flexible than the state-of-the-art theoretically. In other words, compared with the state-of-the-art, RF-RISA could generate fewer reconfigurations when switching models. In this section, we further verify this statement.

At first, two accelerators are set with initial configurations of almost the same size of storage. Then 16 RF models with different structures are built randomly. Finally, the reconfigurations are simulated by adapting the two accelerators to these models. For the sake of simplicity, all trees in the model are assumed to have the same depth, and the storage required by each layer is one BRAM. Then for a model with $n$ trees and $d$ layers, RF-RISA and the state-of-the-art both require $n*d$ BRAMs as the storage. When generating the models, the number of BRAM required is set to be no more than a certain value. It can be considered as the maximum memory that the hardware can provide. 600 is selected in our experiment. The initial M of RF-RISA is set as 50, 300, 600. Accordingly, the number of trees and the tree depth of the state-of-the-art are configured as (7-7), (17-17), (24-24), respectively.

The simulation results are shown in Fig. 11. Each time the accelerator is reconfigured to adapt to the model, its new configuration is annotated in the figure. We can draw a few conclusions from the results. First of all, the reconfiguration frequency of RF-RISA is far lower than that of the state-of-the-art. Therefore, less cost would be introduced by RF-RISA, such as the repeated development and time overhead. Secondly, as the initially allocated storage increases, two accelerators show different characteristics. For RF-RISA, its reconfiguration times decrease. Since the larger the storage, the wider the range of models can be accelerated by RF-RISA. In terms of the state-of-the-art, there is a weak relationship between the reconfiguration times and the initial configuration. This is because its reconfiguration is affected by both the tree depth and the number of trees. Thus, it is beneficial to allocate storage resource as much as possible for RF-RISA, which ensures to reduce the overhead of switching models. However, it is difficult to choose a good initial configuration for the state-of-the-art. Finally, although the 16 models are generated randomly, it reveals a typical scene of how the two accelerators would perform.

In the above experiment, the number of nodes in each layer is not considered. In fact, RF-RISA also has more tolerance to this factor than the state-of-the-art. Suppose the initial configurations of RF-RISA and the state-of-the-art are 600 BRAMs and (30-20), respectively. Besides, each BRAM can store up to 256 nodes. Table 2 lists several models and whether they can be accelerated. Among them, Model 4 gives a new case that the state-of-the-art cannot be applied effectively. The number of nodes in layer 14 of one tree exceeds 256, the state-of-the-art must be reconfigured to accelerate Model 4 even if the total required BRAM decreases. In conclusion, the proposed RF-RISA has greater flexibility than the state-of-the-art.

#### 4.4.2. Logical resource consumption

In this section, we compare the logical resource consumption of the core module of the two accelerators. Fig. 12 illustrates their usage in combinational ALUTs and dedicated logic registers. For each storage size of RF-RISA, the state-of-the-art is configured with the parameters which consume the same size of storage.

Apparently, RF-RISA consumes many more logical resources than the state-of-the-art. The consumption of ALUTs by RF-RISA is more than 6 times that of the state-of-the-art. In terms of dedicated logic registers, RF-RISA occupies them more than 15 times as much as the state-of-the-art. This result is reasonable since the logic in RF-RISA is complex. For example, each PE in RF-RISA undertakes three different operations for different states, while the PE in the state-of-the-art only performs the comparison. Meanwhile, the node address can be obtained straightly by one reading operating in the state-of-the-art. However, the NOInst address is obtained through a calculation in RF-RISA.

Another observation is that the logical resources consumed by RF-RISA increase linearly as the number of allocated BRAM increases. The reason is that each BRAM in RF-RISA possesses the same PE. For each BRAM unit, RF-RISA consumes about 644 ALUTs and 660 dedicated logic registers. By contrast, the state-of-the-art shows something different. When the storage is increased with the same size, the resource consumption of increasing the number of trees is slightly less than that of increasing the tree depth. However, the difference between them is so minute that it is negligible.

In conclusion, RF-RISA gains its flexibility at the cost of logical resources. From the growth trend of the consumed logic resources, it can be inferred that the logical resources may be the bottleneck of RF-RISA.

#### 4.4.3. Throughput

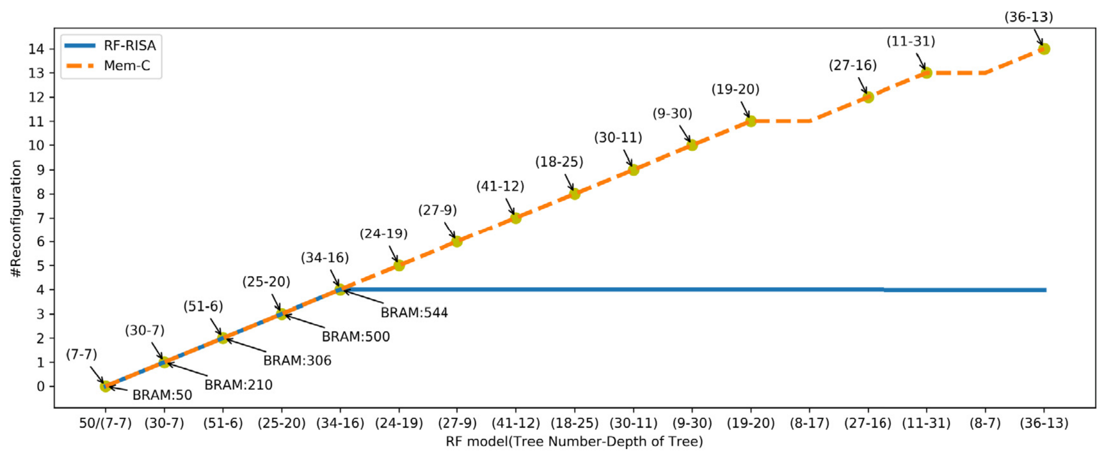
Based on the network traffic classification scenario, an RF model [34] is accelerated by RF-RISA and the state-of-the-art. The RF model includes 30 trees with a maximum depth of 20. Besides, the number of internal nodes in each layer is less than 256. Therefore, 600 BRAMs are required as the node storage for both accelerators. Moreover, two CPU platforms with different configurations are added in the comparison. On the CPU platform, the prediction process is implemented based on the scikit-learn library [32] and is parallelized over the trees. The transmission frequency of the network data is 156 MHz.

The comparison results are shown in Table 3. It can be seen that RF-RISA provides a slightly higher synthesized maximum frequency than the state-of-the-art. In terms of logic resource consumption, their performance is consistent with the analysis in Section 4.4.2. Besides, the BRAM bits used by RF-RISA are larger than that of the state-of-the-art. One reason is that more bits are used to encode the nodes in RF-RISA than the state-of-the-art. Another reason is that some intermediate data generated during the prediction are stored by BRAM in RF-RISA. Finally, since both accelerators utilize fully pipelined architectures, one output per clock cycle can be achieved by both of them. When the clock setting of the two accelerators is consistent with the data transmission frequency, the practical throughput of both of them reaches 156M. In contrast, the throughput of the two CPU platforms is much smaller than the FPGA-based accelerators.

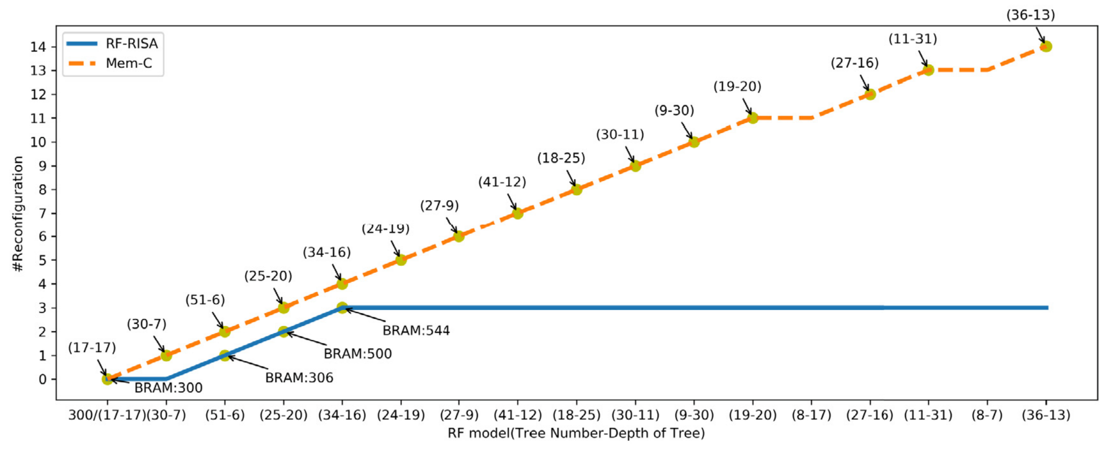
## 5. Discussion

### 5.1. Latency

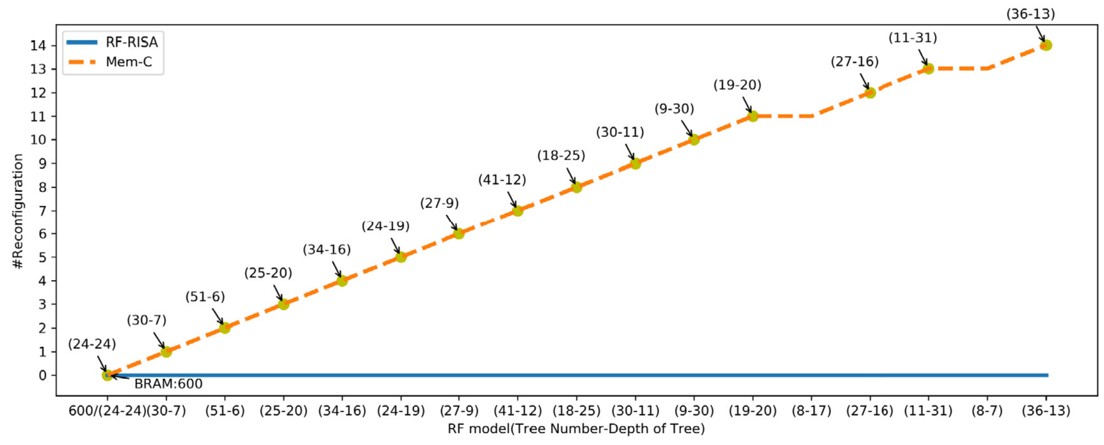
An evident issue of RF-RISA is that it has a longer latency compared with the existing Mem-C accelerators. The latency of the existing Mem-C accelerators is determined by the maximum depth of the model, while the latency of RF-RISA is decided by the number of the allocated BRAM. For example, the latency of the RF-RISA prototype and the implemented Mem-C accelerator in this paper are about 800 clock cycles and 20 clock cycles, respectively. That



(a) initial BRAM=50



(b) initial BRAM=300



(c) initial BRAM=600

**Fig. 11.** The reconfiguration simulation with different initial configuration.

**Table 2**
The flexibility comparison of the two FPGA-based accelerators.

| Model | # of Tree | Tree depth | # of nodes in layer 14 of one tree | # of needed BRAM (RF-RISA/Mem-C) | RF-RISA | Mem-C |
|---|---|---|---|---|---|---|
| 1 | 20 | 17 | 120 | 340/340 | √ | √ |
| 2 | 35 | 17 | 110 | 595/595 | √ | × |
| 3 | 20 | 25 | 110 | 500/500 | √ | × |
| 4 | 20 | 20 | 450 | 401/420 | √ | × |

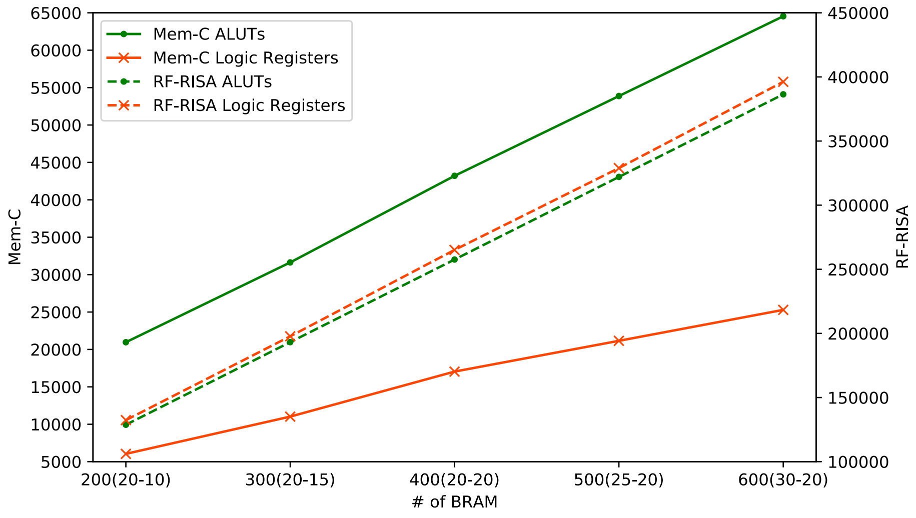


**Fig. 12.** The consumption of ALUTs and dedicated logic registers.

**Table 3**
The comparison of different platforms and accelerators.

| Platform | Synthesized maximum frequency | Logic utilize | BRAM bits | Practical throughput |
|---|---|---|---|---|
| FPGA/the state-of-the-art | 558.97 MHz | 18% | 12% | 156M |
| FPGA/RF-RISA | 577.7 MHz | 59% | 17% | 156M |
| CPU (3.4 GHz, 8G RAM, 4 cores, 8 logical processors | / | / | / | 0.358M |
| CPU (3.4 GHz, 32G RAM, 6 cores, 12 logical processors) | / | / | / | 1.21M |

is, the existing accelerator and RF-RISA give a prediction every 0.128us and 5.128us in the network traffic classification scenario.

Although the latency of RF-RISA has increased a lot, it has little impact on some practical applications, especially for the scenarios that focus on the throughput. On the other hand, due to the powerful computing power provided by FPGA, RF-RISA could still meet the speed requirements of many real-time tasks.

### 5.2. Optimization

The above implementation of RF-RISA can be further optimized. Here we suggest several optimizations that can be considered in future work.

(1) The current implementation of the RF-RISA utilizes one port of the BRAM. For some FPGA devices, the BRAM can be configured as dual-port. Then two instances can access one BRAM at the same time if two ports are utilized. In this way, the throughput can be doubled.

(2) In our prototype, one BRAM stores the nodes in one layer to realize pipelining. As a result, the bits utilization of BRAMs that store the first few layers of the trees would be extremely low. One solution to this problem is to map the shallow layers of the trees into smaller and faster storage provided by FPGA. Besides, Alireza et al. [22] suggest another solution that two layers of the tree can be merged and stored as one layer. This solution improves the bits utilization at the cost of decreasing throughput.

(3) In our prototype, the prediction of one instance completes after all BRAMs are traversed, even if the RF model only occupies a part of BRAMs. In future work, the number of consumed BRAM can be parameterized to indicate when should the prediction process finish. It can reduce the prediction latency when the model does not run out of the BRAMs.

### 5.3. Implementation of other flexible calculations

Although RF-RISA aims at the acceleration of Random Forest, we believe that the idea behind RF-RISA can be applied to other calculations beyond Random Forest.

For example, convolution is a critical calculation in the artificial neural network [13]. With the size of the convolution kernel as $n * m$, $n * m$ multiplications and additions would be performed in one convolution. When the hardware accelerator is required to support size-variable kernels, the idea behind the RF-RISA can be considered. That is, decoupling the size of the kernel from the hardware implementation. A straightforward approach is to design the multiplication and addition as an instruction, then the convolution can be transformed into $n * m$ instructions. Next, storage resources and PE should be organized in a way that supports executing instructions in pipeline. Finally, only the number of instructions changes when the kernel size changes. Other hardware modules would not be affected.

## 6. Related work

### 6.1. The Comp-C DT accelerator

Qu et al. [28] accelerate a DT-based lookup engine with the Comp-C architecture. The DT is transformed into a rule table and is implemented by combinational logic in FPGA. With dual-port DRAM, two search requests can be accepted per clock cycle. Li et al. [18] realize a DT accelerator for gas identification based on the threshold network. Flora et al. [1,2] design an automatic process to produce the VHDL code for the Comp-C DT accelerator. Similarly, Mario et al. [3] deploy a tool whereby hardware models can be automatically generated from software RF models. Compared with the Comp-c accelerator, the proposed RF-RISA in our work does not hard code the comparison operations into hardware.

**Table 4**
The summary of representative accelerators.

| | # of trees | Depth of trees | # of nodes in layers | Flexibility | Throughput (per clock cycle) |
|---|---|---|---|---|---|
| Jiang et al. [16] | 1 | 11 | variable | medium | 2/ppc |
| Fareena et al. [31] | 1 | 13 | $2^i$ | low | 8/ppc |
| Alireza et al. [22] | 1 | preseted by user | preseted by user | low | 1/ppc |
| Mohammed et al. [11] | 20 | 10 | $2^i$ | low | 1/ppc |
| Owaida et al. [26] | variable | variable | $2^i$ | high | 1/ppc |
| RF-RISA | variable | variable | variable | high | 1/ppc |

### 6.2. The Mem-C DT accelerator

Jiang et al. [16] present a Mem-C DT accelerator based on FPGA for packet classification. Two pipelines are used to traverse the tree and rules in nodes. Besides, a model adjustment method is proposed to balance the distribution of nodes between layers. In their work, the nodes in the same layer of the tree can be mapped onto different pipeline stages. Raaj et al. [29] utilize a similar DT accelerator for multi-field packet classification. Hiroki et al. [23] present an improved RF model that has a shorter path, hence its accelerator could achieve lower latency. They further develop a tool that could generate the host code and kernel code for the high-level synthesis tool for the FPGA [17]. In their implementation, the fixed-point number is used to represent the floating-point threshold, which may affect the accuracy of the model. Zhao et al. [33] present a discretization method to deal with the floating-point threshold in RF. Based on their method, the floating-point operations in RF can be converted into integer operations. Fareena et al. [31] realize a Mem-C DT accelerator and repeat its pipeline 8 times to improve the throughput. Fong et al. [12] implement a DT accelerator by integrating the above methods. Alireza et al. [22] implement the Mem-C DT accelerator based on NetFPGA. Then several optimizations are provided to reduce the memory consumption and the computation latency. Abdelrahman et al. [10,9] implement a customized Mem-C DT accelerator to verify correct Bluetooth operation in real-time. Mohammed et al. [11] implement a Mem-C RF accelerator for network traffic classification. Tong et al. [37] develop an automatic tool. The tool takes a DT as input and automatically generates the Verilog code for the corresponding Mem-C hardware architecture. What is closest to our work, Owaida et al. [26] present a FPGA-based tree ensemble classifier accelerator with a software driver to map the trees into the memory. It combines the CPU and FPGA to scale to tree ensembles that do not fit in on-chip memory. However, its hardware implementation still employs the existing Mem-C architecture. They further improve their work to partition, balance, and implement large applications on clustered FPGAs [25,27]. The comparison of FPGA-based DT accelerators with different architectures can be found in [19][35].

The summary of several representative works based on the Mem-C architecture is listed in Table 4. RF-RISA differs from the above Mem-C DT accelerators in the following aspects. Firstly, RF-RISA utilizes a whole new hardware architecture. In our architecture, storages are organized in a way that could eliminate the concept of the tree in the hardware. Hence, the structure of the model is transparent to the hardware architecture. Secondly, RF-RISA represents the RF model in a different way. It encodes not only the node information but also the structure information. The change of the model would result in the change of its representation results, without affecting the other modules in RF-RISA. Therefore, RF-RISA can provide greater flexibility than the existing accelerators.

### 6.3. Other application-specific accelerators

In addition to Random Forest, FPGA is commonly employed to accelerate other applications. Liu et al. [20] propose PuDianNao, a polyvalent accelerator that supports seven machine learning algorithms. They extract critical computational primitives from the algorithms and design them as atomic instructions. Then calculation units are implemented to execute the instructions. Similarly, Wang et al. [40] propose an accelerator that is able to accommodate four machine learning algorithms and seek instruction-level parallelism by running the tasks out-of-order.

Due to the great application potential of FPGA, some cloud providers are even offering the acceleration as a service. For example, Xilinx launches the framework xDNN [41]. It provides a compilation tool to accelerate customized CNN models. Zebra [21] is developed by Mipsology for compiling neural network models into Xilinx FPGA-based accelerators. Besides, Intel provides OpenVINO toolkit [15] to assist the acceleration of machine learning algorithms based on FPGA. Brainwave [7], a project proposed by Microsoft, also provides an accelerated serving of DNNs. It designs a parameterized soft vector processor to control the deployment and acceleration.

## 7. Conclusion

At present, the existing RF accelerators based on FPGA suffer from insufficient flexibility since their architectures have many restrictions on the applicable model. This paper presents RF-RISA, a flexible RF accelerator that can be applied to a wide range of RF models from multiple applications. Based on an instruction set, the RF model can be transformed into a set of instructions. With a pipelined hardware architecture that is independent of model parameters, the instructions can be loaded directly into the hardware memory for acceleration. Therefore, the switching of a model can be achieved without a reconfiguration of the FPGA. The proposed RF-RISA maximizes the flexibility of the RF accelerator and could achieve a comparable throughput as the state-of-the-art.

## CRediT authorship contribution statement

**Shuang Zhao:** Conceptualization, Methodology, Validation, Writing – original draft. **Shuhui Chen:** Conceptualization, Formal analysis, Supervision. **Hui Yang:** Writing – review & editing. **Fei Wang:** Conceptualization, Resources. **Ziling Wei:** Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was supported by the National Key Research and Development Program of China (2018YFB0204301) and The Science and Technology Innovation Program of Hunan Province (2020RC2047).

## References

[1] Flora Amato, Mario Barbareschi, Valentina Casola, Antonino Mazzeo, Sara Romano, Towards automatic generation of hardware classifiers, in: International Conference on Algorithms and Architectures for Parallel Processing, 2013, pp. 125–132.

[2] Flora Amato, Mario Barbareschi, Valentina Casola, Antonino Mazzeo, An FPGA-based smart classifier for decision support system, in: Intelligent Distributed Computing, vol. 511, 2014, pp. 289–299.

[3] Mario Barbareschi, Salvatore Del Prete, Francesco Gargiulo, Antonino Mazzeo, Carlo Sansone, Decision tree-based multiple classifier system: an FPGA perspective, in: Multiple Classifier Systems-International Workshop, 2015.

[4] Tobias Becker, Qiang Liu, Wayne Luk, Georg Nebehay, Roman Pflugfelder, Hardware accelerated object tracking, in: Proceeding of the Conference on Field Programmable Logic and Applications, 2011.

[5] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. stone, Classification and regression trees, Eur. J. Oper. Res. 19 (1) (1983).

[6] Sebastian Buschjäger, Katharina Morik, Decision tree and random forest implementations for fast filtering of sensor data, IEEE Trans. Circuits Syst. 65 (1) (2018) 209–222, https://doi.org/10.1109/TCSI.2017.2710627.

[7] Eric Chung, Jeremy Fowers, Kalin Ovrcharov, et al., Serving DNNs in real time at datacenter scale with project brainwave, IEEE MICRO 38 (2) (2018) 8–20, https://doi.org/10.1109/MM.2018.022071131.

[8] Ismael-Antonio Dávila-Rodríguez, Marco-Aurelio Nuño-Maganda, Uahir Hernández-Mier, Said Polanco-Martagón, Decision-tree based pixel classification for real-time citrus segmentation on FPGA, in: International Conference on ReConFigureable Computing and FPGAs, 2019, pp. 1–8.

[9] Abdelrahman Elkanishy, Derrick T. Rivera, Abdel-Hameed A. Badawy, Paul M. Furth, Z.M. Saifullah, Christopher P. Michael, An FPGA decision tree classifier to supervise a communication SoC, in: IEEE High Performance Extreme Computing Conference, 2019, pp. 1–6.

[10] Abdelrahman Elkanishy, Derrick T. Rivera, Paul M. Furth, Abdel-Hameed A. Badawy, Youssef Aly, Christopher P. Michael, FPGA-accelerated decision tree classifier for real-time supervision of bluetooth SoC, in: International Conference on ReConFigurable Computing and FPGAs, 2019, pp. 1–5.

[11] Mohammed Elnawawy, Assim Sagahyroon, Ramer Shanableh, FPGA-based network traffic classification using machine learning, IEEE Access 8 (2020) 175637–175650, https://doi.org/10.1109/ACCESS.2020.3026831.

[12] Jeffrey Fong, Yaxuan Qi, Jun Li, Weirong Jiang, ParaSplit: a scalable architecture on FPGA for terabit packet classification, in: Annual Symposium on High-Performance Interconnects, 2012, pp. 1–8.

[13] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[14] Taiga Ikeda, Kento Sakurada, Atsuyoshi Nakamura, Masato Motomura, Shinya Takamaeda-Yamazaki, Hardware/algorithm co-optimization for fully-parallelized compact decision tree ensembles on FPGAs, in: International Symposium on Applied Reconfigurable Computing, 2020, pp. 345–357.

[15] Intel, Intel vision products deep learning inference acceleration white paper [online], https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/intel-vision-accelerator-design-with-FPGA-wp.pdf.

[16] Weirong Jiang, Viktor K. Prasanna, Large-scale wire-speed packet classification on FPGAs, in: International Symposium on Field Programmable Gate Arrays, 2009, pp. 219–228.

[17] Akira Jinguji, Shimpei Sato, Hiroki Nakahara, An FPGA realization of a random forest with k-means clustering using a high-level synthesis design, IECE Trans. Inf. Syst. 101 (2) (2018) 354–362, https://doi.org/10.1587/transinf.2017RCP0006.

[18] Qingzheng Li, Amine Bermak, A low-power hardware-friendly binary decision tree classifier for gas identification, J. Low. Power Electron. Appl. 1 (1) (2011) 45–58, https://doi.org/10.3390/jlpea1010045.

[19] Xiang Lin, R.D. Shawn Blanton, Donald E. Thomas, Random forest architectures on FPGA for multiple applications, in: Proceeding of the on Great Lakes Symposium on VLSI, 2017, pp. 415–418.

[20] Daofu Liu, Tianshi Chen, Shaoli Liu, et al., PuDianNao: a polyvalent machine learning accelerator, in: Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems, 2015, pp. 369–381.

[21] Mipsology, Accelerate your inference, anywhere, in no time and with no effort using mipsology's zebra [online], http://www.mipsology.com/zebra.html.

[22] Alireza Monemi, Roozbeh Zarei, Muhammad Nadzir Marsono, Mohamed Khalil-Hani, Parameterizable decision tree classifier on NetFPGA, in: Advances in Intelligent Systems and Computing, vol. 182, 2013, pp. 119–128.

[23] Hiroki Nakahara, Akira Jinguji, Simpei Sato, Tsutomu Sasao, A random forest using a multi-valued decision diagram on an FPGA, in: International Symposium on Multiple-Valued Logic, 2017, pp. 266–271.

[24] Jason Oberg, Ken Eguro, Ray Bittner, Alessandro Forin, Random decision tree body part recognition using FPGAs, in: International Conference on Field Programmable Logic and Application, 2012.

[25] Muhsen Owaida, Gustavo Alonso, Application partitioning on FPGA clusters: inference over decision tree ensembles, in: International Conference on Field-Programmable Logic and Applications, 2018, pp. 295–300.

[26] Muhsen Owaida, Hantian Zhang, Ce Zhang, Gustavo Alonso, Scalable inference of decision tree ensembles: flexible design for CPU-FPGA platforms, in: International Conference on Field Programmable Logic and Applications, 2017, pp. 1–8.

[27] Muhsen Owaida, Amit Kulkarni, Gustavo Alonso, Distributed inference over decision tree ensembles on cluster of FPGAs, ACM Trans. Reconfigurable Technol. Syst. 12 (4) (2019), https://doi.org/10.1145/3340263.

[28] Yun R. Qu, Viktor K. Prasanna, Scalable and dynamically updateable lookup engine for decision-trees on FPGA, in: High Performance Extreme Computing Conference, 2014, pp. 1–6.

[29] R. Sathesh Raaj, J. Kumarnath, FPGA based packet classification using multi-pipeline architecture, Int. J. Comput. Sci. Mobile Comput. 4 (4) (2015) 541–548, https://doi.org/10.11648/j.wcmc.20150303.11.

[30] Kuaga Rafa, Gorgoń Marek, FPGA implementation of decision trees and tree ensembles for character recognition in VIVADO HLS, Image Process. Commun. 19 (2–3) (2014) 71–82, https://doi.org/10.1515/ipc-2015-0012.

[31] Fareena Saqib, Aindrik Dutta, Jim Plusquellic, Philip Qrtiz, Marios S. Pattichis, Pipelined decision tree classification accelerator implementation in FPGA, IEEE Trans. Comput. 64 (1) (2015) 280–285, https://doi.org/10.1109/TC.2013.204.

[32] Scikit-learn: machine learning in Python, http://scikt-learn.org/stable/.

[33] Zhao Shuang, Sun Yipin, Chen Shuhui, A discretization method for floating-point number in FPGA-based decision tree accelerator, in: IEEE International Conference on Computer and Communications, 2018, pp. 2698–2703.

[34] Zhao Shuang, Chen Shuhui, Sun Yipin, Cai Zhiping, Su Jinshu, Identifying known and unknown mobile application traffic using a multilevel classifier, Secur. Commun. Netw. 2 (2019), https://doi.org/10.1155/2019/9595081.

[35] J.R. Struharik, Implementing decision trees in hardware, in: International Symposium on Intelligent Systems and Informatics, 2011, pp. 41–46.

[36] Deepak K. Thakkar, B.S. Agarkar, Scalable packet classification on FPGA, Int. J. Eng. Dev. Res. 5 (3) (2017) 521–528, https://doi.org/10.1109/TVLSI.2011.2162112.

[37] Da Tong, Lu Sun, Kiran Matam, Viktor Prasanna, High throughput and programmable online traffic classifier on FPGA, in: The International Symposium on Field Programmable Gate Arrays, 2013, pp. 255–264.

[38] Da Tong, Yun Rock Qu, Viktor K. Prasanna, Accelerating decision tree based traffic classification on FPGA and multicore platforms, IEEE Trans. Parallel Distrib. Syst. 28 (11) (2017) 3046–3059, https://doi.org/10.1109/TPDS.2017.2714661.

[39] Brian Van Essen, Chris Macaraeg, Maya Gokhale, Ryan Prenger, Accelerating a random forest classifier: multi-core, GP-GPU, or FPGA?, in: International Symposium on Field-Programmable Custom Computing Machines, 2012, pp. 232–239.

[40] Chao Wang, Lei Gong, Xi Li, Xuehai Zhou, A ubiquitous machine learning accelerator with automatic parallelization on FPGA, IEEE Trans. Parallel Distrib. Syst. 31 (10) (2020) 2346–2359, https://doi.org/10.1109/TPDS.2020.2990924.

[41] Xilinx, AI inference acceleration [Online], https://www.xilinx.com/applications/megatrends/machine-learning.html.

**Shuang Zhao** received the M.Sc. degree in computer science from the Chongqing University, Chongqing, China, in 2016, and the B.Sc. degree in computer science from the National University of Defense Technology, Changsha, China, in 2018. She is currently pursuing a Ph.D. in cyberspace security at the National University of Defense Technology. Her research interests include network traffic analysis and user profile.

**Shuhui Chen** received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 2007. He is currently an professor with the National University of Defense Technology. His research interests include network traffic analysis and network security.

**Hui Yang** is currently an assistant professor of computer science at the National University of Defense Technology, China. She received the Ph.D. degree in computer science from National University of Defense Technology. Her main research interests include network architecture, network processor and router.

**Fei Wang** received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 2013. She is currently an associate professor with the National University of Defense Technology. Her research interests include network traffic analysis and network security.

**Ziling Wei** received the B.Sc. and M.Sc. degrees in computer science from the National University of Defense Technology, Changsha, China, in 2012 and 2014, respectively, and the Ph.D. degree in electrical engineering from the University of Alberta, Edmonton, AB, Canada, in 2019. He is currently an Assistant Professor with the National University of Defense Technology. His research interests include wireless communications, network security, and edge computing.