

---

# TREAM: A Tool for Evaluating Error Resilience of Tree-Based Models Using Approximate Memory

Mikail Yayla, Zahra Valipour Dehnoo, Mojtaba Masoudinejad, Jian-Jia Chen  
TU Dortmund, Department of Computer Science, Dortmund, Germany





Citation: [https://doi.org/10.1007/978-3-031-15074-6\\_4](https://doi.org/10.1007/978-3-031-15074-6_4)

---

## BIB<sub>T</sub><sub>E</sub><sub>X</sub>:

```
@inproceedings{10.1007/978-3-031-15074-6_4,  
author = {Yayla, Mikail and Valipour Dehnoo, Zahra and Masoudinejad, Mojtaba and Chen, Jian-Jia},  
title = {TREAM: A Tool for Evaluating Error Resilience of Tree-Based Models Using Approximate Memory},  
year = {2022},  
isbn = {978-3-031-15073-9},  
publisher = {Springer-Verlag},  
address = {Berlin, Heidelberg},  
url = {https://doi.org/10.1007/978-3-031-15074-6_4},  
doi = {10.1007/978-3-031-15074-6_4},  
booktitle = {Embedded Computer Systems: Architectures, Modeling, and Simulation: 22nd International Conf},  
pages = {6173},  
numpages = {13},  
keywords = {Random forest, Decision tree, Bit errors, Approximate memory, Error resilience},  
location = {Samos, Greece}  
}
```

# TREAM: A Tool for Evaluating Error Resilience of Tree-based Models using Approximate Memory

Mikail Yayla , Zahra Valipour Dehnoo , Mojtaba Masoudinejad , and  
Jian-Jia Chen 

Department of Computer Science, TU Dortmund University, Dortmund, Germany  
{mikail.yayla, zahra.valipour, mojtaba.masoudinejad,  
jian-jia.chen}@tu-dortmund.de

**Abstract.** Approximate memories reduce the resource demand of machine learning (ML) systems at the cost of bit errors. ML models have an intrinsic error resilience and are therefore suitable candidates to use with approximate memories. Although the error resilience of neural networks has been considered in many studies, tree-based applications have received less attention. In addition, there is no tool available to specifically evaluate the error resilience of tree-based models. In this work, we present TREAM, a general tool built upon the sklearn framework for injecting bit flips during the inference of tree-based models. TREAM is capable of injecting bit flips into the tree and input parameters, i.e. feature and split values, in addition to feature and children indices. It can also be used for both floating point and integer values.

Furthermore, we provide an abstract assessment of bit flip injection into the aforementioned parameters. Based on this, we construct a set of experiments using TREAM for different random forest structures and datasets over a set of bit error rates, where the relation between accuracy and bit error rate is considered for error resilience. The results demonstrate that child indices have the highest deviations in accuracy under bit errors. Moreover, the results give us insights into how varying numbers of trees, depth and different datasets affect the resilience.

**Keywords:** Random forest · Decision tree · Approximate memory · Error resilience · Bit errors

## 1 Introduction

Over the past decade, machine learning (ML) algorithms have become ubiquitous on various hardware platforms. For instance, by integrating ML into embedded devices, data can be processed directly at the edge instead of being sent over the network [4,7]. Among ML algorithms, decision trees (DTs) are one of the most widely used [3]. This popularity is due to their interpretability, simplicity and excellent results, specifically in the form of random forests (RFs) [16]. DTs/RFs can be successfully applied to complex problems, such as high-dimensional data [17] and can even outperform other ML models, such as neural networks for

structured data [4]. Therefore, they are suitable candidates for ML application in resource-constrained embedded systems [7].

To achieve competitive inference accuracy, high depths and a large number of trees are recommended, which requires storing large amount of data in the memory. Hence, the memory subsystem is the main resource used in tree-based models [2], which makes it a challenge to apply them on conventional memories.

Recent studies have explored the use of approximate memory realized by reducing the memory supply voltage and tuning latency parameters. Although an approximate memory can achieve lower power consumption and faster access, it can cause high bit error rates (BERs). This has been explored for a variety of state-of-the-art and emerging memory technologies, e.g. for volatile memories (SRAM [19], DRAM [11]), and approximate non-volatile memories (RRAM [9], MRAM or STT-RAM [10], FeFET [20]). Furthermore, various studies have investigated the error resilience of ML models by creating a bit flip (i.e. faults) injector tool. Most of these studies focus on deep neural networks and propose several tools. The framework in [12] enables the modeling of bit flips into various parts of the hardware. Similarly, [15,5,13] propose tools and techniques to quantify the error resilience and accuracy trade-offs of neural networks. Although TensorFI [6] injects bit flips into any generic ML program written using TensorFlow, faults are only injected into the TensorFlow operators and not into the memory used by the program. In addition, it is not designed specifically for RFs or other tree-based models.

Hence, a tool for evaluating the accuracy of tree-based models under errors from approximate memory would allow us to conduct design space exploration in this domain. To the best of our knowledge, this work is the first study that investigates the error resilience of DTs/RFs with bit-flip injection into the DT/RF data. Specifically, we assess tree-based models with approximate memory, for which there are also no corresponding studies in the literature. This makes it possible to explore the future low-power systems employing tree-based models with approximate memory.

Ultimately, this tool allows users to inject the bit flips into DT/RF data for all the above-mentioned memory types. In this manner, after selecting the injection site or node data type, the injection into the integer or floating point data values can be performed in the form of multiple bit flips. All intended configurations are first recorded and then executed during the inference phase. Therefore, it enables the investigation of the impact of errors on tree-based models in approximate memory.

**Our contributions are as follows:**

- We present TREAM, which makes possible to evaluate the accuracy under errors from approximate memories or other sources of bit errors for tree-based ML-models. It is based on the popular machine learning library sklearn. We plan to publish the tool along with the paper.
- We provide an abstract assessment of the effects of bit flips into the different error sites, i.e. split and feature values, and feature and child indices.

- Using our tool, we conduct a series of experiments on different datasets for evaluating the inference accuracy degradation of the tree-based models due to errors. In the analyses, we identify the most susceptible parts of trees to errors, and determine the influence of the number of trees and depth on the accuracy.

The rest of this paper is organized as follows. In Section 2, we present the system model along with the posed research questions. TREAM and its implementation as the bit flip injector tool is shown in Section 3. In Section 4, we address the effect of bit flips in tree-based models. The experimental results and answers to the research questions are presented in Section 5. The tool TREAM is available at <https://github.com/myay/TREAM>.

## 2 System Model

Let us consider a supervised learning problem to build a prediction model  $M$  from a labeled training dataset  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \dots, N\}$ . The model’s input is a vector  $\mathbf{x}_i = (x_i^1, \dots, x_i^d)^\top \in \mathcal{X} \subseteq \mathbb{R}^d$ , and a prediction  $y_i \in \mathcal{Y}$  is a class index  $c \in \mathbb{N}_0$ . Note that the indices  $1 \dots d$  in the components of  $\mathbf{x}_i$  represent the dimension and not exponents. The model  $M : \mathbf{x} \rightarrow c$  is applied to predict the class for an input  $\mathbf{x}$ , where it may be an unseen input ( $\mathbf{x} \notin \mathcal{D}$ ).

A decision tree (DT) is a flow-like process used as a model for predicting unseen data, based on the learned information from the training data. Suppose that  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$  is a DT composed of  $|\mathbf{V}|$  vertices (typically called *nodes* in this context) and  $|\mathbf{E}|$  directed edges, in which  $|\mathbf{E}| = |\mathbf{V}| - 1$ . There is one *root* in  $\mathbf{V}$ , which has no incoming edge. We consider *binary DTs*, in which a node  $v \in \mathbf{V}$  is either a *leaf* without any outgoing edge or an *internal node* with exactly two outgoing edges. In a binary DT, a leaf provides a classification label and an internal node  $v \in \mathbf{V}$  is a split decision based on a comparison of a certain *feature* ( $f_v$ ) from the input data with the *split value* ( $s_v$ ), also called *threshold*, learned during the training to further classify the input using either the left or the right child.

The *depth* ( $K$ ) of a DT is the longest path from the root to one of the leaves. An example of a binary DT is presented in Fig. 1a. Each comparison divides the possible feature space into subsections with hyperplanes at the thresholds, such as in Fig. 1b, which operates with five classes using four thresholds.

A DT’s inference process from the root to a leaf can be implemented either in the *if-else* or in the *native* form [2]. Sklearn uses the *native* realization with a high level structure similar to Listing 1.1 written in Cython. The type `SIZE_t` in the node is an unsigned integer and `DOUBLE_t` is a floating point value with double precision. The data used on the node for `feature`, `split`, `leftchild`, and `rightchild` are shown in Listing 1.1. There are also other data, e.g. a binary value to mark a node as leaf (not shown here).

A RF is an ensemble made of multiple DTs, predicting by majority votes. For brevity, the term *tree-based models* is used hereafter as an RF with  $T$  decision trees with  $T \geq 1$ .

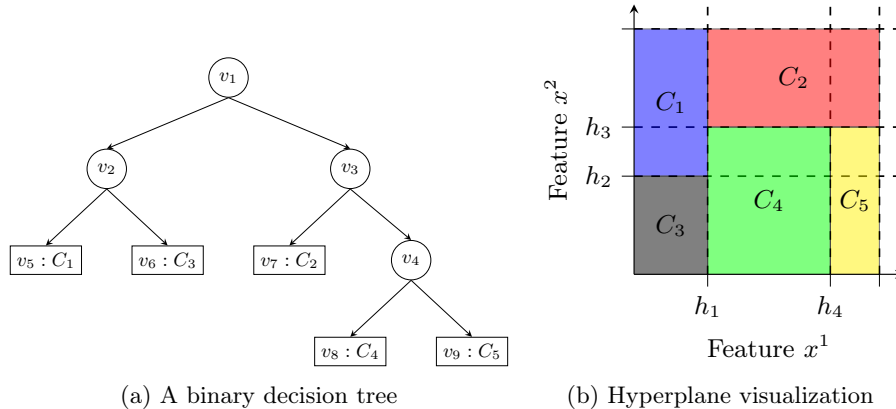


Fig. 1: Left: A DT with 4 internal nodes (in circle) and 5 leafs (in rectangular). Right: Partitions of space by the nodes (hyperplanes) of a DT into five classes.

```

cdef struct node:
    SIZE_t feature # targeted feature
    DOUBLE_t split # threshold
    SIZE_t leftchild
    SIZE_t rightchild
    ...
# all nodes and leafs of a tree are stored in this array
Node tree = [[350,143.5,1,2,...],[1,7.5,3,4],...]
# high-level description of predict function
def predict(X):
    i = 0
    while node is not leaf:
        if X[tree[i].feature] <= tree[i].split:
            i = node.leftchild
        else:
            i = node.rightchild
    return tree[i].prediction

```

Listing 1.1: Native realization of a DT in Cython

## 2.1 Memory and Error Model

Using the native realization of the DTs from Listing 1.1, the node data for making decisions includes the split value, feature value, feature index and two child indices. We suppose that they are stored in a one-dimensional array data structure in the memory with possible bit flips. In this study, we assume that the hardware (i.e. memory) may suffer from *transient faults* captured in a bit error model. Application of this model on the bit representation of the data is realized by injecting bit flips into the memory. Specifically, we consider that the bit errors are manifested as *multiple-bit flips*, and that they are *symmetric*, i.e., the probability for flipping  $0 \rightarrow 1$  is the same as  $1 \rightarrow 0$ . This assumption characterizes the probability of bit flips every time when a bit is read from the unreliable memory, also referred to as *approximate memory*.

We consider that errors occur during the inference phase of tree-based models with the probability of error, referred to as bit error rate (BER). This can occur in the bit representation of the mentioned node data read from the unreliable memory. Consequently, bit flips are injected into the binary representations of split and feature values, and the indices for features and children, according to the bit error model of the hardware. This model matches the assumptions in recent studies about approximate volatile memories (SRAM [18], DRAM [11]), and non-volatile memories (RRAM [9], MRAM or STT-RAM [10], and FeFET [20]). Errors stemming from other types of faults may also be applied using this model.

## 2.2 Definitions for Error Resilience

Let us use  $A_\beta$  to quantify the accuracy of models ( $M$ ) under bit flip injection with BER ( $\beta$ ). When this accuracy is measured using a set of experiments with  $\beta \in \mathcal{B}$ , we define the error resilience as:

$$R_M(\mathcal{B}) = \frac{100}{|\mathcal{B}| \times A_0} \sum_{\forall \beta \in \mathcal{B}} A_\beta \quad [\%] \quad (1)$$

In this formulation  $A_0$  represents the accuracy of the original model without any bit flip injection and is used as the reference. Higher drops in the accuracy will reduce the nominator and result in smaller resilience factors.

## 2.3 Research Questions

Considering a tree-based model with approximate memory storing the node’s data, the main goal is to analyze the impact of bit errors on the inference accuracy. To this end, the following research questions will be evaluated:

- **RQ1:** Which data in the tree-based model is more sensitive to the errors?
- **RQ2:** What is the influence of depth and the number of trees in the ensemble on the error resilience?
- **RQ3:** Are there differences concerning error resilience among different datasets?

# 3 TREAM: An extension to sklearn

TREAM is an extension for the well-known machine learning library sklearn [14]. Sklearn provides implementations of common supervised and unsupervised ML algorithms with a Python-based interface. It is one of the most efficient ML-tools available due to its use of Numpy and Cython in the back-end [14].

## 3.1 High-level Overview of TREAM

According to the structure of DT in Listing 1.1, the bit flip injection into the following data sites are considered: 1) Split value, 2) feature value, 3) feature

index, and 4) child indices. These sites can be selected individually or in any combination to perform the error resilience and accuracy evaluation.

Algorithm 1 shows the high-level overview of how TREAM works. It first initializes the experiment parameters and then activates bit flip injection into the aforementioned sites depending on the flags. From Line 11 on, the program iterates over the specified BERs in the experiment parameters. The bit flip injection takes place during the model inference and are only injected into the accessed data for efficiency. The accuracy estimation is repeated for the specified number in *REPs* (i.e. repetitions, specified in the experiment parameters). When the execution of the loop is finished, a list of accuracies is returned.

---

**Algorithm 1:** TREAM high-level overview

---

**Input:** Trained Model  $M$ , Input  $\mathbf{X}$ , Set of BERs ( $\mathcal{B}$ )  
**Output:** Accuracy  $\mathbf{A}$  over  $\mathcal{B}$

- 1 Initialize experiment parameters
- 2 Initialize list of accuracies  $\mathbf{A}$
- 3 **if** *BitFlipSplitValue* **then**
- 4   activate SplitValueInjection
- 5 **if** *BitFlipFeatureValue* **then**
- 6   activate FeatureValueInjection
- 7 **if** *BitFlipFeatureIdx* **then**
- 8   activate FeatureIndexInjection
- 9 **if** *BitFlipChildIdx* **then**
- 10   activate ChildIndexInjection
- 11 **for**  $\beta$  in  $\mathcal{B}$  **do**
- 12   **for** *rep* in *REPs* **do**
- 13     Apply model  $M(\mathbf{X})$  under  $\beta$
- 14     Append  $A_\beta$  to  $\mathbf{A}$
- 15 **return**  $\mathbf{A}$

---

The process of bit flip injections into tree-based models can be configured through an external configuration file (ECF) to enable fast design space exploration for error resilient systems. It provides the required parameters and configurations to train a model (or load a pre-trained model) and inject bit flips in the specified sites during inference. This workflow is shown in Fig. 2.

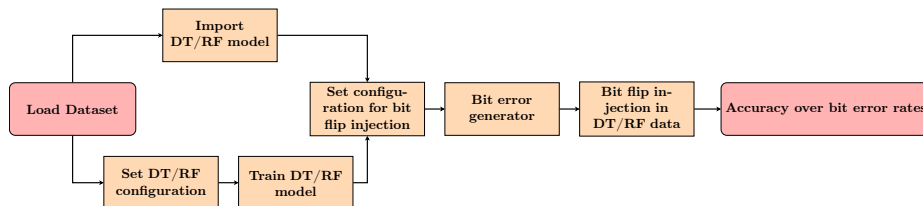


Fig. 2: TREAM operational steps.

## 3.2 Implementation

The back-end of the tree computations in sklearn is implemented in the programming language Cython [1]. Cython code uses Python-based syntax, with optional syntax inspired by the programming language C (such as specifying data types). When the implementation is complete and the code should be executed, it is compiled to C/C++ code first, and then processed further. This not only produces efficient machine code, it also allows direct bit-level access to the DT's data, enabling the iteration through the binary representation of values and their manipulation. Hence, this is used to implement the bit flip injection during the inference. In the following, we explain how we extended the sklearn codebase to support bit flip injection into the DT data.

**New Files** are files that were not part of sklearn and are added for the implementation of TREAM.

`sklearn/tree/_bfi.pdx`: This file includes type and function definitions for bit flip injections, for floating point and integer representations with any number of bits. For example, the function for the floating point format is `cdef DTYPE_t bfi_float(DTYPE_t x, DTYPE_t ber)`. Input ( $x$ ) and bit error rate (BER, in decimal) need to be specified.

`sklearn/tree/_bfi.pyx`: This file provides the implementation of the bit flip injection. We note that floating-point numbers are represented by the IEEE 754 binary format in modern computing systems. Therefore, bit flip injection can be directly applied onto the binary representation of floating points or integers. To achieve this in Cython, the corresponding bits, denoted as  $\mathbf{a}$ , of a floating point number or an integer are copied using the `memcpy` function (to work around the strict-aliasing rule). Then, a loop iterates over each bit that can flip, and if the corresponding random value is smaller than the BER, the bit is marked as '1', i.e., to be flipped; otherwise, marked as '0', i.e., kept as it is. Suppose that  $\mathbf{m}$  is the resulting bit vector of the above operation. Then,  $\mathbf{m}$  is considered as a mask and the bit flips are performed using the  $\mathbf{a} := \mathbf{m} \text{ xor } \mathbf{a}$ , in which `xor` is a bit-wise xor operation. In the end, the bit vector  $\mathbf{a}$  containing the original input (with flips, if bits have been flipped) are copied to the original value, again using `memcpy`.

**Modified Files** are files that have been part of the original sklearn but are modified for the TREAM implementation.

`sklearn/tree/_tree.pxd` and `sklearn/tree/_tree.pyx`: with values to turn on and off bit flip injection into different types of DT data during inference. BERs can also be specified for each type of data. In the function `apply_dense`, the bit flip injection function calls are performed (the case sparse application is not supported, but can easily be added).

`sklearn/tree/setup.py`: To compile sklearn with the changes, this installation file has been extended with the new files related to `_bfi.pyx`.



## 4 The Effect of Bit Flips in Tree-based Models

In this section we give an overview of the effects of bit flips in the different bit flip sites in tree-based models and how bit flips are related to the model structure. By this we intend to give context for the research questions, before conducting the bit flip injection experiments.

### 4.1 Bit Flips in Split and Feature Value

Let us denote the nodes' comparison using  $x \leq s$ , where  $x$  is an input feature and  $s$  the split value. When bit flips occur in the bit representation of  $s$  (independent of the data type), the representation with bit flips ( $s^*$ ) encodes either a larger or a smaller value. Depending on the values of  $x$  and  $s$ , i.e. whether  $x > s$  or  $x \leq s$ , the magnitude of  $s^*$  determines whether the node chooses the wrong child. For the wrong child to be chosen, the magnitude of  $s^*$  has to be small or large enough to *invert* the result of the comparison  $x \leq s$ . The same principle holds for bit flips in the bit representation of the feature value  $x$ .

We now explain how a wrong child (by bit flips in  $s$  and/or  $x$ ) can affect the prediction. Consider a DT with only two features, splitting the space with hyperplanes  $h_i$  as shown in Fig. 1. For  $h_1$  and  $h_4$ , the split to the left child corresponds to splitting the feature space such that the space on the left remains. For  $s_2$  and  $s_3$ , a split to the left child corresponds to splitting the feature space such that the space above remains. For right children, the space is partitioned the other way around. As an example, consider a two-dimensional input  $\mathbf{x} = (x^1, x^2)$  belonging to the class  $C_2$ . The model has to predict this class from the five classes denoted as  $C_1 - C_5$ . A wrong child selection in  $v_1$  will fall either in  $C_1$  or  $C_3$ , which are the wrong classes. The same principle applies to all the other nodes. In each case, one wrong split leads to a wrong classification.

### 4.2 Bit Flips in Child Indices

Child indices are unsigned integers, one pointing to the index for the left and one for the right child. Considering that only the child indices have bit flips in their bit representation, following cases can occur:

1. The index under bit flips gets out of array bounds. In this case, a memory address which is not inside the memory region reserved for the nodes will be accessed which may store any other data. This memory address could also be protected by the operating system, and an access attempt could cause a segmentation fault, leading to a termination of the program.
2. The index under bit flips stays inside the array bounds reserved for the node. In this case, the wrong index will be used, though it is a valid node in the tree. If the execution jumps to a parent node, the execution is repeated. If this happens repeatedly, the execution time may become longer.

### 4.3 Bit Flips in Feature Index

Bit flips in the bit representation of the feature index (also an unsigned integer) can have the following outcomes:

1. The feature index under bit flips gets out of bounds. This case is the same as case 1 for the child index.
2. The feature index under bit flips stays inside the array bounds reserved for the feature values. In this case, a wrong feature will be chosen which may have a value different to the correct one. This is similar to the case in Section 4.1.

## 5 Evaluation

To demonstrate the TREAM tool, we introduce bit flips into tree-based ML models. We focus on RFs with depths and number of trees in  $\{5, 10\}$ . As datasets, we use MNIST, sensorless-drive, wine-quality, and adult, from the UCI machine learning repository [8].

For training RFs, the standard sklearn libraries, without any error tolerance or correction methods are used. After training an RF, the accuracy under errors is evaluated using TREAM with BERs between  $10^{-5}\%$  and 50%. To acquire precise estimates of the accuracy, the inference under each BER is repeated five times for each RF. All experiments are run on a machine with Intel Core i7-8700K 3.70 GHz, 32 GB RAM.

We inject bit flips into the bit representations of the floating point format of the split and feature values. TREAM also supports bit flip injection into integer representations. The feature index is an unsigned integer, into which bit flips are also injected. A check finds out-of-bounds errors in the index of the feature value (explained in Section 4.1) and sets it to zero which retrieves the first feature. If it is within the bounds, the feature at the wrong index is retrieved by the node. The child indices are unsigned integers as well. If bit flips cause the child indices to get out of bounds, the execution of the tree is terminated and the class with index zero is returned as the prediction. If the indices are within the bounds, a wrong node is retrieved as the next child node.

Using TREAM, the effect of bit flip injection on all data sites on the RF accuracy of two example datasets are presented in Fig. 3. The value  $K$  refers to the maximum depth of the trees, and  $T$  refers to the number of trees in the RFs.

The extracted accuracies after bit flip injections can be used to quantify the error resilience of each data site for each dataset using Eq. (1). From collected data this value is presented for all datasets in Table 1. Using accuracy drops and extracted resilience factors we refer to the research questions posed in Section 2.3.

**RQ1:** According to Fig. 3 (fourth column), we observe that the child indices are most sensitive to errors. This leads to wrong decisions with one bit flip, which causes a high accuracy drop despite small BERs. Bit flips in the split value, feature value, and feature index (columns one to three in Fig. 3) are less sensitive to errors compared to the child index. In these three cases, bit flips

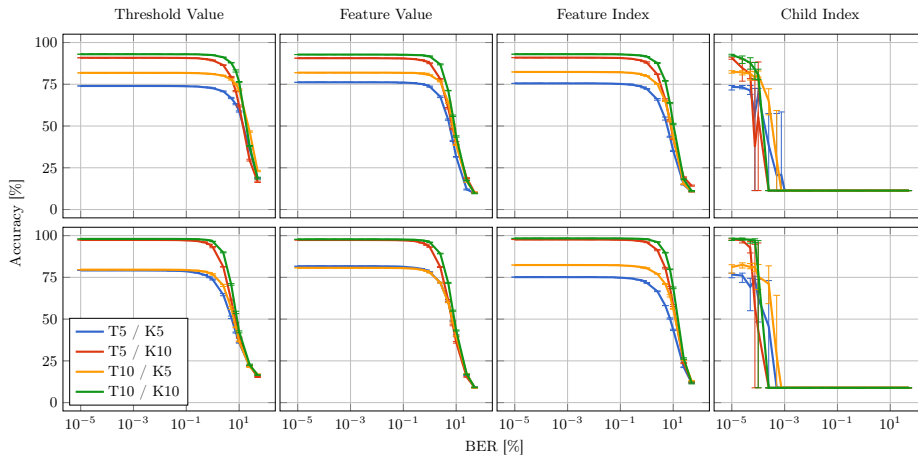


Fig. 3: Accuracy over BERs for different RFs, varying bit flip sites, datasets, and RF architectures. Row 1: MNIST, Row 2: sensorless-drive.

Table 1: Error resilience (in percent) of different experiments with bit flip injection on Feature Value (FV), Threshold Value (TV), Feature Index (FI) and Child Index (CI)

Model	MNIST				Sensorless-drive				Wine quality				Adult			
	FV	TV	FI	CI	FV	TV	FI	CI	FV	TV	FI	CI	FV	TV	FI	CI
T5 / K5	87.9	93.3	88.3	31.9	88.5	87.9	89.7	28.1	93.4	95.0	96.9	86.1	98.1	97.8	98.6	91.5
T5 / K10	87.9	91.6	88.6	24.3	86.8	87.5	90.3	22.0	90.6	90.8	94.1	80.9	97.5	96.0	97.8	89.3
T10 / K5	89.2	94.6	89.3	32.4	88.7	88.9	91.3	30.0	95.9	96.9	97.4	85.9	98.5	97.8	98.4	90.6
T10 / K10	89.0	93.4	89.9	27.2	88.4	88.8	91.7	24.9	92.0	92.2	94.3	78.2	98.1	97.1	98.1	89.2

may not always lead to a wrong decision. The bit flips may be tolerated without a change of the nodes traversed to reach a leaf. The resilience values in Table 1 also support this. The values for the CI are the lowest among one dataset.

**RQ2:** In Table 1 we observe for all datasets and for the first three bit flip injection sites (split value, feature value and feature index), that inclusion of more trees in the RF can enable a slightly better resilience. However, in some cases this is a weak improvement or may not hold (e.g. CI in wine-quality, and FI and CI in adult). Furthermore, the variance of the accuracy for child index plots is high, which makes observations challenging. Regarding the influence of the tree depth on resilience, no consistent observations can be made from Fig. 3 and Table 1.

**RQ3:** Among the datasets, a certain amount of bit flips are tolerated without significant accuracy degradation, for small BERs. We also observe that a higher BER directly increases the accuracy drop. Furthermore, the BERs at which the accuracy drops drastically is similar (between  $10^{-1}$  and  $10^1$ ) for the threshold value, feature value, and feature index. For the child index this region

Table 2: Execution times (in seconds) for the MNIST dataset averaged from 40 times on the test set sized 10 000 elements.

Model	TREAM [s]			sklearn [s]		
	average	avg.-min.	max.-avg.	average	avg.-min.	max.-avg.
T5 / K5	3.64	$9 \cdot 10^{-2}$	0.16	$6.5 \cdot 10^{-2}$	$5 \cdot 10^{-3}$	$4 \cdot 10^{-3}$
T5 / K10	7.15	0.22	0.63	$7.5 \cdot 10^{-2}$	0	$3 \cdot 10^{-3}$
T10 / K5	7.29	0.19	0.71	$8.4 \cdot 10^{-2}$	0	0
T10 / K10	13.71	1.64	0.5	0.11	0	$1 \cdot 10^{-3}$

is between  $10^{-5}$  and  $10^{-4}$ . However, we observe that the resilience values vary largely in Table 1. MNIST and sensorless-drive have CI resilience around 20-30 percent, while wine-quality and adult have values around 78-90 percent. The high resilience values for CI for adult and wine-quality are due to the inherent imbalance in these datasets.

To provide a sense on the effect of bit flip injection on the execution time, latency evaluations are provided in Table 2. Each set of values is averaged from five repetitions on the test set performed for eight different BERs, each for 10 000 samples. We observe that TREAM has additional timing overheads compared to the sklearn. This overhead is to be expected, since bit flips are injected into the parameters during the inference phase.

## 6 Conclusion

We present TREAM, an extension of sklearn, for error resilience analysis of tree-based models (DT/RF). TREAM can be configured to inject bit flips into the node data and input data, i.e. split and feature value, and feature and child indices, during the inference phase. The evaluations demonstrate that the child index is the most sensitive parameter of the node to the bit flips. Moreover, more trees in an RF can tolerate more errors and the accuracy of various datasets under bit flip injections can vary. We believe that this tool will help the research community explore the design of efficient systems using ML on the resource-constrained edge.

As a future work, we plan to find solutions for the timing overhead in the TREAM tool, which occurs due to the fact that the bit flip injections are performed every time the data is fetched from the memory. Furthermore, we plan to use TREAM to evaluate tree-based models under specific approximate memories, with the goal of quantifying the efficiency benefits.

**Acknowledgements** This paper has been supported by Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Analysis” (project number 124020371) and project OneMemory (project number 405422836).

## References

1. Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D.S., Smith, K.: Cython: The best of both worlds. *Computing in Science & Engineering* **13**(2), 31–39 (2011). <https://doi.org/10.1109/MCSE.2010.118>
2. Buschjager, S., Chen, K.H., Chen, J.J., Morik, K.: Realization of random forest for real-time evaluation through tree framing. In: 2018 IEEE International Conference on Data Mining (ICDM). pp. 19–28. IEEE (2018). <https://doi.org/10.1109/ICDM.2018.00017>
3. Charbuty, B., Abdulazeez, A.: Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends* **2**(01), 20–28 (2021). <https://doi.org/10.38094/jastt20165>
4. Chen, K.H., Su, C., Hakert, C., Buschjäger, S., Lee, C.L., Lee, J.K., Morik, K., Chen, J.J.: Efficient realization of decision trees for real-time inference. *ACM Transactions on Embedded Computing Systems (TECS)* (2022). <https://doi.org/10.1145/3508019>
5. Chen, Z., Li, G., Pattabiraman, K., DeBardeleben, N.: Binfi: an efficient fault injector for safety-critical machine learning systems. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–23 (2019). <https://doi.org/10.1145/3295500.3356177>
6. Chen, Z., Narayanan, N., Fang, B., Li, G., Pattabiraman, K., DeBardeleben, N.: Tensorfi: A flexible fault injection framework for tensorflow applications. In: 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). pp. 426–435. IEEE (2020). <https://doi.org/10.1109/ISSRE5003.2020.00047>
7. Cornetta, G., Touhafi, A.: Design and evaluation of a new machine learning framework for iot and embedded devices. *Electronics* **10**(5), 600 (2021). <https://doi.org/10.3390/electronics10050600>
8. Dua, D., Graff, C.: UCI machine learning repository (2017), <http://archive.ics.uci.edu/ml>
9. Hirtzlin, T., Bocquet, M., Klein, J.O., Nowak, E., Vianello, E., Portal, J.M., Querlioz, D.: Outstanding bit error tolerance of resistive ram-based binarized neural networks. In: 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). pp. 288–292. IEEE (2019). <https://doi.org/10.1109/AICAS.2019.8771544>
10. Hirtzlin, T., Penkovsky, B., Klein, J.O., Locatelli, N., Vincent, A.F., Bocquet, M., Portal, J.M., Querlioz, D.: Implementing binarized neural networks with magnetoresistive ram without error correction. In: 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH). pp. 1–5. IEEE (2019). <https://doi.org/10.1109/NANOARCH47378.2019.181300>
11. Koppula, S., Orosa, L., Yağlıkçı, A.G., Azizi, R., Shahroodi, T., Kanellopoulos, K., Mutlu, O.: Eden: Enabling energy-efficient, high-performance deep neural network inference using approximate dram. In: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 166–181 (2019). <https://doi.org/10.1145/3352460.3358280>
12. Li, G., Hari, S.K.S., Sullivan, M., Tsai, T., Pattabiraman, K., Emer, J., Keckler, S.W.: Understanding error propagation in deep learning neural network (dnn) accelerators and applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–12 (2017). <https://doi.org/10.1145/3126908.3126964>

13. Mahmoud, A., Aggarwal, N., Nobbe, A., Vicarte, J.R.S., Adve, S.V., Fletcher, C.W., Frosio, I., Hari, S.K.S.: Pytorchfi: A runtime perturbation tool for dnns. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). pp. 25–31. IEEE (2020). <https://doi.org/10.1109/DSN-W50199.2020.00014>
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. *the Journal of machine Learning research* **12**, 2825–2830 (2011)
15. Reagen, B., Gupta, U., Pentecost, L., Whatmough, P., Lee, S.K., Mulholland, N., Brooks, D., Wei, G.Y.: Ares: A framework for quantifying the resilience of deep neural networks. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2018). <https://doi.org/10.1109/DAC.2018.8465834>
16. Rodriguez-Galiano, V., Sanchez-Castillo, M., Chica-Olmo, M., Chica-Rivas, M.: Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines. *Ore Geology Reviews* **71**, 804–818 (2015). <https://doi.org/10.1016/j.oregeorev.2015.01.001>
17. Shaik, A.B., Srinivasan, S.: A brief survey on random forest ensembles in classification model. In: International Conference on Innovative Computing and Communications. pp. 253–260. Springer (2019). [https://doi.org/10.1007/978-981-13-2354-6\\_27](https://doi.org/10.1007/978-981-13-2354-6_27)
18. Sun, X., Liu, R., Chen, Y.J., Chiu, H.Y., Chen, W.H., Chang, M.F., Yu, S.: Low-vdd operation of sram synaptic array for implementing ternary neural network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**(10), 2962–2965 (2017). <https://doi.org/10.1109/TVLSI.2017.2727528>
19. Yang, L., Bankman, D., Moons, B., Verhelst, M., Murmann, B.: Bit error tolerance of a cifar-10 binarized convolutional neural network processor. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5. IEEE (2018). <https://doi.org/10.1109/ISCAS.2018.8351255>
20. Yayla, M., Buschjager, S., Gupta, A., Chen, J.J., Henkel, J., Morik, K., Chen, K.H., Amrouch, H.: Fefet-based binarized neural networks under temperature-dependent bit errors. *IEEE Transactions on Computers* (2021). <https://doi.org/10.1109/TC.2021.3104736>