國立臺灣大學電機資訊學院資訊工程學系

學士班學生論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

**Bachelor's Thesis**

# 基於內容可定址記憶體優化之樹狀集成模型加速方法

# RETENTION: Resource-Efficient Tree-Based Ensemble Model Acceleration with Content-Addressable Memory

廖奕鈞

Yi-Chun Liao

指導教授：郭大維 博士

Advisor: Tei-Wei Kuo, Ph.D.

中華民國 114 年 4 月

April, 2025

# 國立臺灣大學學士班學生論文
# 口試委員會審定書

## 基於內容可定址記憶體優化之樹狀集成模型加速方法
## RETENTION: Resource-Efficient Tree-Based Ensemble Model Acceleration with Content-Addressable Memory

本論文係廖奕鈞君（學號 B10902034）在國立臺灣大學資訊工程學系完成之學士班學生論文，於民國一百一十四年四月二十一日承下列考試委員審查通過及口試及格，特此證明

口試委員(3 位)：

郭大維 （簽名）

（指導教授）

張原豪 劉永達

系 主 任 ： 陳祝嵩 （簽名）

# 摘要

儘管近年來深度學習於非結構化資料方面取得極大的進展，現代樹狀集成模型在從結構化資料中擷取相關資訊與學習方面仍佔據優勢。然而，目前針對樹狀模型加速相關的研究相對稀少，且此類模型本質上的特性限制了傳統加速器的效能。內容可定址記憶體被視為一種極具潛力的樹狀模型加速器，惟其極高的記憶體容量需求與低使用率限制了其實用性。為了解決上述的問題，本文提出了 RETENTION，一種端對端框架來顯著減少加速樹狀模型推理所需的內容可定址記憶體容量。此框架為基於引導聚集演算法的模型（如隨機森林）提供全新的剪枝準則，並設計迭代剪枝演算法以確保在給定的正確率損失範圍內最小化模型。同時，此架構提供一套樹狀結構映射機制，並結合兩種資料擺放策略，以減輕伴隨記憶體內搜尋的而來的記憶體冗餘問題。實驗結果表明 RETENTION 可大幅減少對內容可定址記憶體的容量需求，進一步提升樹狀模型在資源受限環境中加速的可行性。

**關鍵詞：**樹狀機器學習、隨機森林、極限梯度提升、內容可定址記憶體、記憶體內運算、剪枝演算法

# Abstract

Although deep learning has demonstrated remarkable capabilities in learning from unstructured data, modern tree-based ensemble models remain superior at extracting relevant information and learning from structured datasets. However, limited research has focused on accelerating tree-based models, whose inherent characteristics pose challenges for conventional accelerators. Content-addressable memory (CAM) offers a promising solution for accelerating tree-based models, yet it suffers from excessive memory consumption and low utilization. This work addresses these challenges by introducing RETENTION, an end-to-end framework that significantly reduces CAM capacity requirement for tree-based model inference. We propose an iterative pruning algorithm with a novel pruning criterion for bagging-based models (e.g., Random Forest) to minimize model complexity while ensuring controlled accuracy degradation. Additionally, we present a tree mapping scheme with two innovative data placement strategies to alleviate the memory redundancy associated with in-memory search. Experimental results demonstrate that RETENTION effectively reduces CAM capacity requirement, making tree-based model acceleration more feasible in resource-constrained environments.

**Keywords:** Tree-based machine learning, Random Forest, XGBoost, content-addressable memory, in-memory computing, pruning algorithm

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1
# Introduction

Structured (i.e., tabular) data is one of the most prevalent formats in data science. It is typically represented as a matrix, where each row corresponds to an instance and all instances share the same set of features across columns. This format is widely used in fields such as finance, medicine, and scientific research, as its structured nature enables efficient processing. For example, sensors generate data in tabular format to support real-time analysis for AI-driven closed-loop control. Despite significant advancements in deep learning for unstructured data (e.g., text, images, and speech), studies [1] [2] have shown that modern tree-based ensemble models consistently outperform deep learning on structured data. In fact, tree-based models remain the state-of-the-art for classification and regression tasks involving medium-sized structured datasets [2], and are often favored over deep learning due to their high interpretability [3], particularly in sensitive applications where understanding model decisions is critical [4]. Tree-based models are widely applied in domains such as scientific research [5], credit card fraud detection [6], and machinery fault diagnosis [7]. A recent survey [8] found that over 74% of data scientists prefer tree-based models, whereas fewer than 40% opt for neural networks, underscoring their continued importance.

However, despite their widespread adoption and effectiveness, tree-based models have received less attention in recent years, and the inefficiency of tree-based model inference remains a critical yet unresolved challenge. Since inference requires multiple

tree traversals, and the number of possible paths grows exponentially with tree depth, predicting and prefetching data becomes challenging. This leads to high inference latency, which is particularly unacceptable for real-time applications. The issue is further exacerbated as modern tree-based models (e.g., XGBoost [9]) can contain thousands of trees, and are often deployed in resource-constrained environments [10]. Although various accelerators have been proposed in recent years [11] [12], researchers [13] have found that conventional accelerators, such as multi-core CPUs, GP-GPUs, and FPGAs, offer limited effectiveness for tree-based model acceleration due to the non-deterministic memory access patterns and irregular tree structures. One promising approach to improving tree-based model inference is in-memory computing, which integrates data storage and processing within the same location, thereby eliminating latency and energy costs associated with data access and transfer. While conventional SRAM-based in-memory accelerators can effectively accelerate model inference, they come at the cost of high energy consumption and extensive area overhead [14], making it impractical for resource-constrained environments.

Recently, there has been significant interest in developing emerging non-volatile-memory-based (NVM-based) accelerators [15] [16] [17], as the characteristics of emerging NVM (e.g., high density, low power consumption, and low cost) make it well-suited for resource-constrained environments. Among potential candidates, non-volatile ternary content-addressable memory (nvTCAM) [18] [19] exhibits the best fit for accelerating tree-based models due to its capability to perform sequence matching with extremely high parallelism. Since every root-to-leaf path within tree-based models can be encoded into a binary sequence, nvTCAM can traverse all the paths in one shot, enabling unprecedented acceleration for model inference. Nevertheless, to support in-memory search, data must be organized in a specialized format. Although nvTCAM effectively

accelerates tree-based model inference in a cost- and energy-efficient manner, a considerable portion of memory cells is allocated for format alignment rather than storing actual model data. This leads to excessive memory consumption with substantial redundancy, which is highly inefficient and requires further optimizations.

To achieve resource-efficient acceleration, the enormous CAM capacity requirement with considerable redundancy is the major obstacle that must be addressed before practical implementation. Since model complexity is highly correlated with memory consumption, pruning models can significantly mitigate this issue. Modern tree-based models often employ pruning algorithms, such as limiting maximum depth or minimum impurity decrease [20], to reduce overfitting and simplify model structure. While these techniques are well-suited for boosting-based tree ensemble models (e.g., XGBoost), where each tree in the ensemble is trained to correct the errors of the previous ones, applying them to bagging-based tree ensemble models (e.g., Random Forest [21]) can lead to severe accuracy degradation due to the independent training of each tree. On the other hand, memory redundancy arises from structuring data for in-memory search, suggesting that a tailored data placement strategy could further reduce CAM capacity requirement. While several studies [22] [23] [24] [25] have explored accelerating tree-based models using non-volatile CAM, little attention has been paid to the issue of memory redundancy, and existing data placement strategies fail to address this challenge effectively and efficiently. Mitigating redundancy requires a solution that not only reduces model complexity but also optimizes memory utilization, paving the way for a more resource-efficient acceleration.

Based on the above observations, we propose RETENTION, an end-to-end framework that minimizes memory consumption for accelerating tree-based models with CAM. RETENTION introduces a pruning algorithm with a novel pruning criterion,

applied iteratively during out-of-bag (OOB) estimation [26] for bagging-based models, which effectively reduces model complexity while ensuring controlled accuracy degradation. In addition, after analyzing the tradeoffs between different data placement strategies, RETENTION incorporates a tree mapping scheme with two innovative data placement strategies, tailored for different optimization criteria, to alleviate memory redundancy and further reduce CAM capacity requirement. To the best of our knowledge, this is the first work that systematically optimizes data placement strategies for tree-based model acceleration with CAM, explicitly considering the tradeoffs between memory redundancy and processing overhead. RETENTION is evaluated on Random Forest and XGBoost using five datasets. Although performing experiments on these two models, the framework can be generalized to other ensemble models with similar structures. Experimental results show that our tree mapping scheme alone achieves 1.46× to 21.30× better space efficiency, while the complete framework improves space efficiency by 4.35 × to 207.12× with less than 3% accuracy degradation. These results demonstrate that RETENTION effectively reduces CAM capacity requirement, making tree-based model acceleration with nvTCAM more resource-efficient and feasible for resource-constrained environments.

The rest of this paper is organized as follows: Chapter 2 presents the background, observation, and motivation of this work. Chapter 3 introduces the philosophy and detailed design of RETENTION. Chapter 4 evaluates RETENTION's effectiveness and compares it with existing works. Finally, Chapter 5 provides concluding remarks.

# Chapter 2
# Background, Observation, and Motivation

## 2.1 Decision Tree, Bagging, and Boosting

Decision tree [27] is a commonly used supervised machine learning algorithm for classification and regression tasks, valued for its simplicity in training and high interpretability. Typically structured as a binary tree, each internal node represents a decision condition based on a feature and a corresponding threshold, while each leaf node stores a predicted result. The training process involves a series of node-splitting operations, where all instances initially start at the root node. During each split, instances within the current node are evaluated, and the optimal feature-threshold pair that best separates them (i.e., generates maximum impurity decrease) is selected as the node's condition. The instances are then distributed to the appropriate child nodes based on whether they satisfy the condition. This process continues iteratively until all instances are separated or predefined constraints, such as maximum depth or minimum impurity decrease, are reached. The left part of Fig. 1 presents an example of decision tree inference. As shown in Fig. 1, inference follows a similar tree traversal process, where an instance starts at the root node and follows a path determined by the conditions at each encountered node. The prediction result is the value stored in the reached leaf node. Since each root-to-leaf path is determined by a set of conditions that an instance must satisfy during traversal, it can be viewed as a sequence of condition checks leading to the final prediction. Additionally, as there is no contradiction between the conditions within a path,

the order of condition checks is irrelevant to the prediction result, providing an opportunity to accelerate inference with CAM.

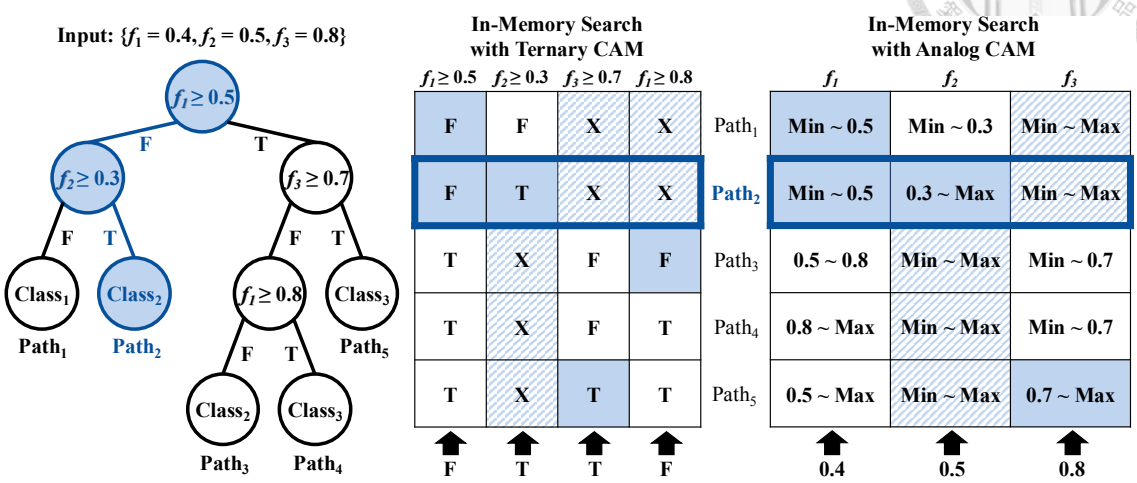


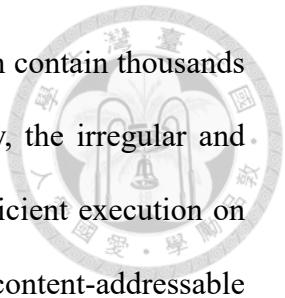

Fig. 1. Visualization of decision tree inference acceleration with TCAM and ACAM.

However, the simplicity in training comes with a major drawback—overfitting, where the model captures noise in the training data, reducing its ability to generalize to unseen instances. To mitigate overfitting and enhance model performance, decision-tree-based ensemble models were introduced.

Bagging (i.e., bootstrap aggregation) [28] is a widely used algorithm for tree ensemble model training. It trains multiple fully-grown decision trees independently, each on a unique subset of the dataset generated through bootstrapping (i.e., sampling with replacement). When making a prediction, each tree produces an individual result, and the ensemble determines the final output via majority voting (if classification) or averaging (if regression). By training trees on different subsets of data, bagging enhances robustness against overfitting and noise. Additionally, bagging enables OOB estimation, an inherent validation method that eliminates the need for a separate validation set. Since each tree is trained on a subset of data, the unused instances can serve as the validation set, which is later passed through the corresponding tree for evaluation. By aggregating predictions

from all trees that did not train on a given instance, OOB estimation provides an unbiased measure of model performance, making it particularly useful when data is limited. Bagging alleviates overfitting, while the ensemble compensates for missing information in individual trees, resulting in improved accuracy compared to a single decision tree. Additionally, since each tree in a bagging-based model is trained independently and contributes equally to the final prediction, models such as Random Forest perform exceptionally well on noisy and small datasets. However, they struggle to capture complex patterns, as they do not fully exploit interactions among trees within the ensemble. Moreover, the averaging mechanism limits the model performance in regression tasks, highlighting the need for an alternative algorithm.

Another class of tree ensemble model primitives is the boosting-based model, which is commonly utilized for both classification and regression tasks. In contrast to bagging, which trains multiple fully-grown decision trees independently, boosting [29] trains shallow decision trees sequentially, with each tree attempting to correct the errors of its predecessors, thereby leveraging the collective power of the ensemble more effectively. During training, boosting fits the first tree on the entire dataset, and each subsequent tree is trained to correct the errors of its predecessors, with errors being reevaluated at each step as new trees are added to the ensemble. When making a prediction, the final result is computed as a weighted sum of individual tree predictions, rather than relying on majority voting or averaging, since trees in boosting-based models contribute with different importance weights. Boosting adapts to task complexity, whereas bagging is constrained by the independence of trees, preventing the ensemble from leveraging mistakes to improve performance. Since boosting iteratively corrects errors, pruning techniques such as limiting depth or impurity decrease have minimal impact on accuracy, while bagging models can experience significant degradation.



However, modern boosting implementations, such as XGBoost, often contain thousands of trees, resulting in substantial memory consumption. Additionally, the irregular and non-deterministic memory access patterns in tree traversal make efficient execution on conventional hardware challenging. To mitigate these inefficiencies, content-addressable memory (CAM) enables massively parallel in-memory search, significantly improving inference speed and efficiency.

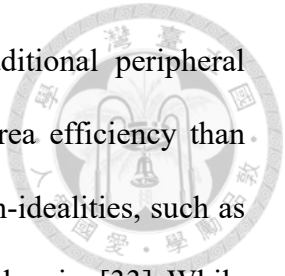
## 2.2 Ternary CAM and Analog CAM

Content-addressable memory (CAM) [14] [30] is a specialized memory architecture widely used in various applications. Unlike conventional memory, which retrieves data based on a given address, CAM allows simultaneous comparison of an input query sequence against all stored binary sequences, and returns the addresses or associated data of matching entries. Data in CAM is stored row-wise, with comparisons performed in parallel on a column basis. This high degree of parallelism makes CAM particularly well-suited for applications requiring rapid lookups, such as network routing, database indexing, and cache systems.

Ternary content-addressable memory (TCAM) [31] extends the functionality of binary CAM by introducing a third state, X, which represents a "don't care" condition. Bits set to the X state are treated as matches during comparison, enabling greater flexibility in search operations. Since each root-to-leaf path in a tree-based model represents a sequence of conditions an input must satisfy to reach a prediction, TCAM can efficiently traverse these paths, making it a compelling solution for tree-based model acceleration. Prior work [22] proposed mapping each path to a TCAM row, where each column represents a unique condition. Unencountered conditions within a path are

assigned the X state as they do not affect the traversal process. When an input instance is received, it is first encoded into a binary sequence that aligns with the condition order stored in TCAM. This encoded sequence is then fed into TCAM for in-memory search. The middle part of Fig. 1 illustrates how TCAM serves as a decision tree accelerator. The decision tree at the left part of Fig. 1 is mapped to the TCAM, where four unique conditions are assigned to separate columns, and each path is represented as a row. Conditions are encoded as 0, 1, and X, corresponding to *False*, *True*, and *don't care*, respectively. Once an input is received, the features are encoded into a binary sequence representing the condition check results, and each bit is then delivered to the corresponding column to perform in-memory search. Since all bits in the second row are matched, *Path_2* is the traversal path for the input instance.

Analog content-addressable memory (ACAM) [32] follows a similar structural design to TCAM, but differs in its data representation. Instead of discrete binary states (0, 1, X), each ACAM cell can store a range of analog values. A cell is considered matched if the given input value falls within the stored range, meaning that a min-to-max range in ACAM functions analogously to the X state in TCAM. Previous research [23] has demonstrated the potential of ACAM for accelerating Random Forest models. Similar to [22], each root-to-leaf path is mapped to an ACAM row. However, unlike TCAM, where each column represents a feature-threshold condition, ACAM columns directly correspond to features. When an input instance is received, a digital-to-analog converter first transforms its features into analog voltages. ACAM then performs in-memory search operations to retrieve the final result. The right part of Fig. 1 provides an overview of ACAM serving as the accelerator for the decision tree in the left part of Fig. 1. In contrast to TCAM, ACAM only requires three columns to represent the features, and each of the cells in ACAM stores an analog range instead.



However, despite its space efficiency, ACAM requires additional peripheral circuits, and its larger cell size does not necessarily offer better area efficiency than TCAM. Furthermore, analog computing is susceptible to various non-idealities, such as stuck-at faults, IR drop, thermal noise, shot noise, and random telegraph noise [33]. While mitigation techniques exist [34], they often introduce additional memory overhead and offer limited effectiveness. Due to these reliability challenges, ACAM is not yet viable for real-world implementation. Therefore, this work opts for TCAM as the accelerator. Still, RETENTION remains compatible with ACAM after slight modifications.

## 2.3 Observation

To accelerate tree-based model inference using TCAM, each root-to-leaf path is mapped to a TCAM row, with each column representing a unique condition within the ensemble. Since a single path encounters only a minimal fraction of the conditions within the ensemble, the majority of TCAM cells store the X state for alignment, leading to substantial redundancy. In fact, mapping a tree-based model to TCAM without optimizations requires at least *#paths* × *#unique_conditions* bits, far exceeding the memory capacity available in resource-constrained environments. As shown in Fig. 1, mapping a decision tree to a CAM with perfect capacity leads to over one-third of cells storing the X state. Moreover, due to the extremely high write latency of nvTCAM, real-time writing severely degrades performance, making it critical to fit the entire model into nvTCAM. Table 1. presents an experiment that highlights the substantial memory requirement and the overwhelming redundancy caused by supporting in-memory search in TCAM. In this experiment, Random Forest models with 100 decision trees required up to 700MB of memory, and all exhibited redundancy of more than 96%. These results

highlight that directly deploying nvTCAM as a tree-based model accelerator is both infeasible and inefficient in resource-constrained environments.

| Dataset | #Paths | Avg. length | #Unique_conditions | Size (MB) | Redundancy (%) |
|---|---|---|---|---|---|
| Adult | 206152 | 16.91 | 29436 | 723.40 | 99.94 |
| CreditApproval | 6426 | 7.39 | 1934 | 1.44 | 99.61 |
| DryBean | 52663 | 12.11 | 41752 | 262.12 | 99.97 |
| Letter | 190377 | 15.59 | 421 | 9.55 | 96.30 |
| Wine | 107772 | 13.53 | 5467 | 70.24 | 99.75 |

Table 1. Memory redundancy caused by supporting in-memory search.

Previous work DT2CAM [22] introduced a framework for mapping decision trees to TCAM. However, instead of addressing memory redundancy, it only proposed an energy-saving precharge mechanism that disables unmatched rows to conserve power. Pedretti et al. [23] proposed mapping Random Forests to ACAM, incorporating techniques such as limiting tree depth and reordering features while eliminating empty rows to mitigate memory redundancy. However, restricting maximum depth in bagging-based models can lead to severe accuracy degradation, while alternative pruning strategies could achieve better accuracy within the same memory budget. Furthermore, although feature reordering groups redundant cells and empty-row elimination reduces memory consumption, the associated computational overhead for result retrieval is relatively high. In fact, comparable computational overhead could yield even greater memory savings. A follow-up study by Pedretti et al. [24] proposed assigning individual trees to separate ACAMs, improving space efficiency. While this approach effectively reduces redundancy across different trees, redundancy within each separate tree remains, leaving room for further optimization. Addressing this residual redundancy could lead to more reductions in memory consumption while maintaining computational efficiency.

## 2.4 Motivation

This work is driven by the challenges identified in the observations and previous works. While nvTCAM holds great promise for accelerating tree-based model inference in resource-constrained environments, the excessive CAM capacity requirement remains a critical challenge that must be addressed before practical implementation. Therefore, this paper proposes RETENTION, an end-to-end framework that significantly reduces CAM capacity requirement, enabling resource-efficient tree-based ensemble model acceleration.

# Chapter 3
# RETENTION

## 3.1 Overview

nvTCAM enables high-speed, parallel inference for tree-based models, making it a promising solution for acceleration. However, substantial memory consumption and redundancy remain key barriers to real-world deployment, limiting its feasibility in resource-constrained environments. In this section, we introduce RETENTION, an end-to-end framework serving as a crucial component to enable practical and resource-efficient acceleration of tree-based model inference with nvTCAM. It addresses the challenges by (1) minimizing model complexity through *purity threshold pruning* and (2) enhancing memory utilization by introducing an optimized tree mapping scheme with two data placement strategies, namely *occurrence-based double reordering (ODR)* and *similarity-based path clustering (SPC)*. Fig. 2. presents an overview of RETENTION, detailing its key components and their interactions. The following sections elaborate on the philosophy and implementation.

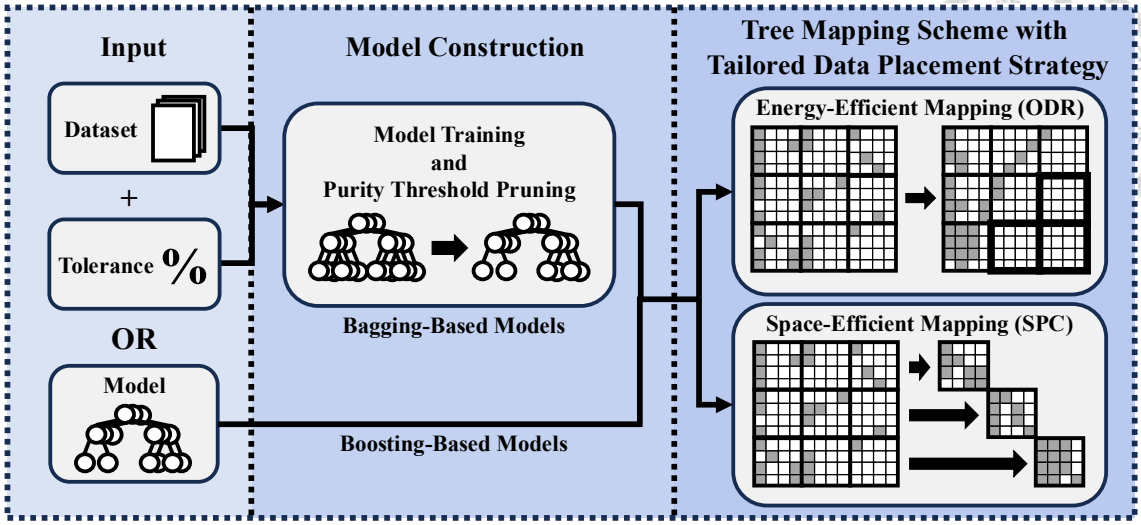



Fig. 2. Overview of RETENTION.

### 3.2 Purity Threshold Pruning

As the required memory capacity is correlated to both *#paths* and *#unique_conditions*, pruning models can effectively improve space efficiency. Existing pruning techniques for reducing model complexity fall into two main categories: (1) structural restriction, which constrains the tree's shape using parameters such as *maximum_depth*, and (2) split-based pruning, which prevents further splits when subsequent splits are deemed worthless (e.g., minimum impurity decrease). While both methods reduce model complexity, neither takes into account the class distribution of instances within nodes, potentially leading to significant accuracy degradation. For example, in complex datasets, instances from different classes may remain entangled even at the maximum depth, resulting in serious performance loss when forced truncation occurs. Similarly, if an early split does not effectively separate instances, later splits might still provide meaningful differentiation, yet split-based pruning prematurely terminates such opportunities. Although these pruning techniques have minimal impact on boosting-based models, where trees iteratively correct previous errors, bagging-based models can

suffer severe accuracy degradation due to their independently trained trees, which lack an error correction mechanism.

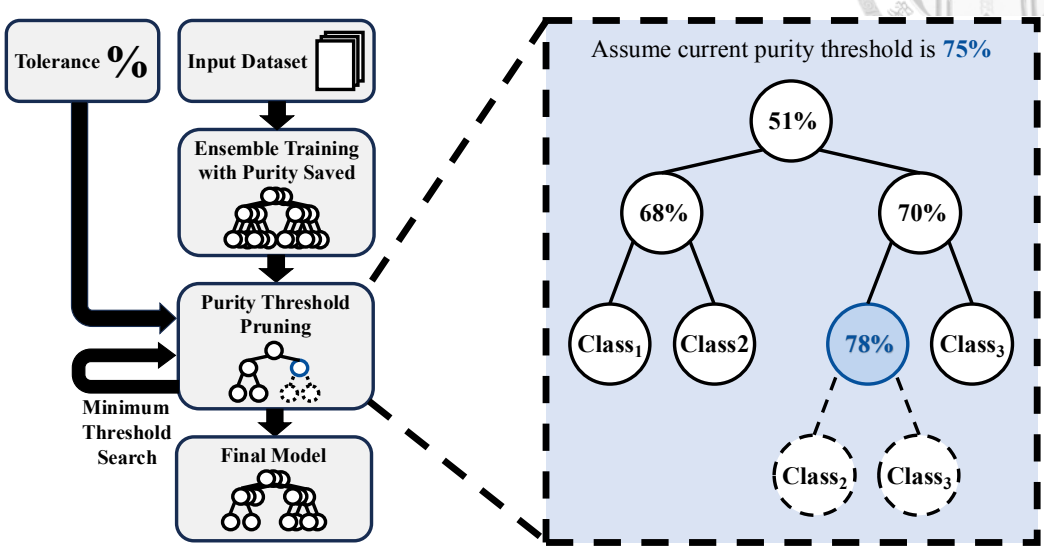



Fig. 3. Workflow of bagging-based model construction with purity threshold pruning.

To avoid potential catastrophic accuracy degradation, this work introduces *purity threshold pruning*, a novel pruning algorithm designed for bagging-based models. The workflow is depicted in the left part of Fig. 2. Unlike conventional training, which only takes a dataset as input, our approach incorporates a user-specified tolerance for OOB accuracy loss. During training, each node records its majority class and the corresponding purity (i.e., proportion). Once the ensemble is constructed, rather than concluding with OOB estimation, we iteratively determine the minimum purity threshold that maintains accuracy within the specified tolerance. Nodes exceeding this threshold are converted into leaf nodes and assigned their respective majority class, as illustrated in the right part of Fig. 2.

The core principle of *purity threshold pruning* is to reduce model complexity while preserving accuracy at the ensemble level. Since individual trees are trained on different subsets with distinct splits, a minority class overlooked in one tree can still be

classified correctly in others. For highly imbalanced datasets, applying class weighting helps prevent the model from favoring majority classes. Additionally, by explicitly considering OOB accuracy during pruning, the method ensures that performance degradation remains controlled. Unlike standard pruning techniques that may create leaf nodes with evenly distributed classes, potentially leading to misclassifications, *purity threshold pruning* ensures that each leaf node maintains sufficient class purity to guarantee the desired level of accuracy. Moreover, when early splits already satisfy the target accuracy, further splits are omitted, significantly reducing both *#paths* and *#unique_conditions* compared to conventional pruning methods.

Furthermore, recent pruning techniques focus on removing entire trees from bagging-based models if they exhibit substantial redundancy or structural resemblance [35] [36]. Since *purity threshold pruning* operates at the node level and assumes trees are diversely grown, it is fully compatible with such whole-tree pruning methods. This compatibility allows further optimization in space efficiency without sacrificing accuracy, making *purity threshold pruning* a highly adaptable approach for resource-efficient tree-based model acceleration. However, the algorithm is not compatible with boosting-based models, as performing pruning after model construction destroys the error correction mechanism, leading to catastrophic accuracy degradation. Nonetheless, the built-in pruning mechanism in boosting-based models sufficiently reduces model complexity without compromising accuracy.

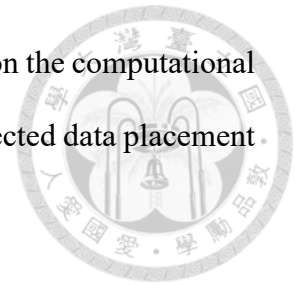
### 3.3 Input Preprocessing during Inference

To accelerate tree-based model inference with CAM, raw input data must be transformed into the specific query format for individual search operations. This

transformation involves two key steps: (1) feature encoding and (2) query packing. For feature encoding, the thresholds of each feature must be sorted before inference. When raw input data for a feature is received, binary search can be employed to accelerate condition checks. This step can also be performed on the sensor side, as modern smart sensors are capable of handling such lightweight tasks [37]. Additionally, since each feature is typically associated with a dedicated sensor, preprocessing at the sensor level is feasible and efficient. Once features are encoded into binary sequences, the query packing step organizes condition check results into the query format, ensuring alignment with the order of TCAM columns. These queries are then transmitted to the corresponding TCAM via a network-on-chip (NoC) [38], following the same implementation as in [24]. In contrast, the input preprocessing for ACAM differs slightly. Since ACAM performs search operations using analog values for each feature, feature encoding is omitted. Instead, digital-to-analog conversions are required after ACAM receives the query.

While tree traversal is essentially a sequence of condition checks, encoding input features into sequences and performing in-memory search may initially seem redundant. However, due to the shared conditions across the ensemble, the actual number of unique conditions is significantly lower than expected. Moreover, binary search can further reduce the number of condition checks. For instance, in a Random Forest model with 100 trees and an average path length of 17.38, traversing the forest requires about 1738 condition checks. However, experimental results reveal that as there are 5446 unique conditions within the model, applying binary search can reduce the number of condition checks to approximately 120 ($14\,\mathit{features} \times \log_2 \frac{5446}{14}$), which is a significant improvement. Additionally, feature encoding is highly efficient compared to conventional CPU-based tree traversal, because all data accesses are deterministic and easily cached. Given that the computational overhead for feature encoding is constant when using TCAM as an

accelerator and negligible when utilizing ACAM, this work focuses on the computational overhead of query packing, which is primarily determined by the selected data placement strategy.



## 3.4 Tree Mapping Scheme

### 3.4.1 Naive Tree Mapping Approaches

To utilize TCAM for acceleration, each root-to-leaf path within the model must be mapped to a TCAM row, with every column of the TCAM corresponding to a unique condition within the paths being searched. There are two naive approaches for mapping: (1) *naive unified mapping* and (2) *naive independent mapping*, with their differences summarized in Table 2, assuming a TCAM size of $S \times S$. In *naive unified mapping*, paths from the entire ensemble are mapped together, so that each row entry corresponds to the same set of conditions. This structure allows for efficient input preprocessing since TCAMs in the same column receive identical input sequences. However, most nodes in the ensemble are not encountered by any single path, resulting in a vast number of cells storing the X state. In fact, mapping 100 trees together can lead to over 96% redundancy, as shown in Table 1. Conversely, *naive independent mapping* maps each tree separately, ensuring that conditions unique to a tree do not introduce unnecessary columns. This approach significantly reduces storage redundancy, as only the conditions relevant to a specific tree are stored, making it far more space-efficient than *naive unified mapping*. However, this independence disrupts input preprocessing because each TCAM now requires a distinct input sequence, increasing preprocessing overhead for query packing.

| | ***Naive unified mapping*** | ***Naive independent mapping*** |
|---|---|---|
| **Mapping unit** | Entire ensemble | Single tree |
| **#Paths** | Number of root-to-leaf paths within a unit | |
| **#Unique_conditions** | Number of unique feature-threshold pairs within a unit | |
| **Width of a unit** | $\lceil \frac{\#unique\_conditions}{S} \rceil$ TCAMs | |
| **Height of a unit** | $\lceil \frac{\#paths}{S} \rceil$ TCAMs | |
| **Total #TCAM required** | $\lceil \frac{\#unique\_conditions}{S} \rceil \times \lceil \frac{\#paths}{S} \rceil$ | $\sum(\lceil \frac{\#unique\_conditions}{S} \rceil \times \lceil \frac{\#paths}{S} \rceil)$ |
| **Key strength** | Lower input preprocessing cost | Higher memory efficiency |
| **Category** | **Energy-efficient mapping** | **Space-efficient mapping** |

Table 2. Comparison of naive unified mapping and naive independent mapping.

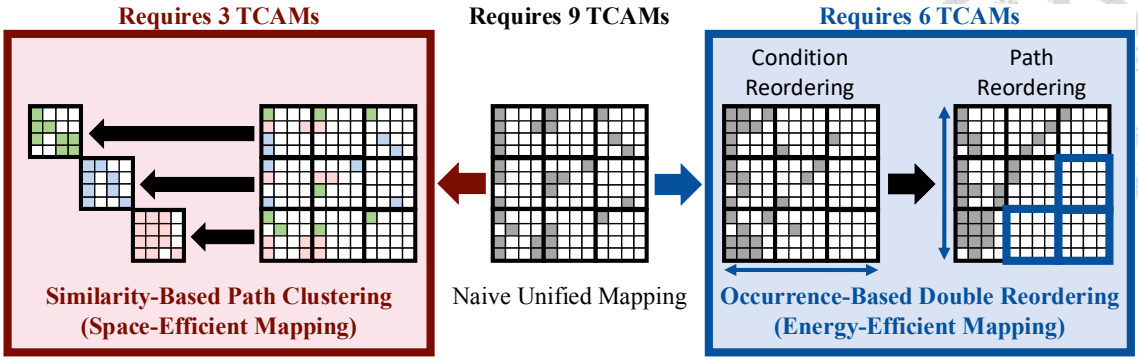
Although *naive unified mapping* generally demands more TCAMs than *naive independent mapping*, the energy-saving precharge mechanism proposed in DT2CAM [22] can help mitigate energy waste by deactivating unmatched rows early. Moreover, input preprocessing dominates energy consumption in inference acceleration, as CPU computations are significantly more power-intensive than in-memory search operations. Given that many nodes are shared among multiple trees, *naive unified mapping* reduces input preprocessing overhead, making it more energy-efficient than *naive independent mapping*. Therefore, based on the above observations, we categorize the optimization criteria of data placement strategies into two types: **energy-efficient mapping** and **space-efficient mapping**. The key difference between these two types is whether to minimize memory usage at the cost of increased computational overhead.

### 3.4.2 Existing Data Placement Strategy Analysis

DT2CAM primarily focused on mapping a single decision tree to TCAM without addressing redundancy or ensemble models. When directly applied to ensembles,

DT2CAM acts as either *naive unified mapping* or *naive independent mapping*, depending on the chosen mapping unit. Pedretti et al. [23] introduced a data placement strategy, referred to as the *feature reordering with row elimination (FR)* algorithm, for ACAM. This method selects the whole ensemble as a mapping unit, and proposes (1) reordering features based on frequency to cluster redundant cells, and (2) eliminating entire path segments when they become redundant. However, due to row elimination, paths were no longer stored in fixed rows, making the result retrieval process substantially more complex and CPU-dependent. As a result, the additional computational overhead is proportional to *#ACAMs*, similar to that of *naive independent mapping*. Thus, we classify *FR* as a space-efficient mapping approach, though its space efficiency may still fall short of *naive independent mapping*. A follow-up study by Pedretti et al. [24] proposed fitting each tree into a separate ACAM without mitigating redundancy, resembling *naive independent mapping*. Consequently, we also categorize this approach under space-efficient mapping.

After a thorough analysis of the tradeoffs among existing data placement strategies, we find that prior approaches fail to effectively mitigate memory redundancy, offering efficiency comparable to naive methods. To bridge this gap, we introduce two novel data placement strategies: one optimized for energy-efficient mapping and the other for space-efficient mapping. The following sections provide a detailed explanation of these strategies, with a visualization presented in Fig. 4.




**Bold:** TCAMs. **Gray/White:** Encountered/Unencountered Conditions. **Colorful:** Similar Encountered Conditions.

Fig. 4. Visualization of the proposed data placement strategies.

### 3.4.3 Occurrence-Based Double Reordering (ODR)

To alleviate memory redundancy in an energy-efficient manner, each path should be stored in a fixed row to avoid additional computational overhead during result retrieval. Moreover, the input sequences for TCAMs in the same column must remain consistent to minimize input preprocessing overhead for query packing. Given these constraints, we propose *ODR* to enhance memory utilization for resource-constrained environments that prioritize energy efficiency. As outlined in Algorithm 1, *ODR* first sorts conditions by occurrence frequency in descending order, similar to *FR*. Instead of eliminating partial sequences when an entire TCAM row is irrelevant to a given path, which introduces sequential computational overhead as seen in *FR*, *ODR* optimizes path ordering based on condition usage. Paths that contain rare conditions are placed at the top, concentrating most of the X-state cells in the bottom-right TCAMs. TCAMs that are entirely filled with X-state cells can then be removed, reducing redundancy without distributing a single path across multiple rows, thereby avoiding additional computational overhead.

Despite its effectiveness, *ODR* still leaves considerable redundancy, as shown in Fig. 4. For example, in the top-right TCAM after applying *ODR*, only a single meaningful cell remains, yet the entire TCAM is required for correct functionality. To further improve

memory efficiency, the next section introduces a more aggressive data placement strategy that reduces CAM capacity requirement at the cost of higher energy consumption. However, its computational overhead is comparable to that of *naive unified mapping*, and is, in fact, even lower in practice.

**Algorithm 1** Occurrence-Based Double Reordering (ODR)

```
Require: pool = root_to_leaf_paths
Require: conditions = pool.union()                ▷ Unique conditions in the pool
Ensure: condition_order, path_order
paths ← {}
sorted_conditions ← conditions.sort()             ▷ Sort conditions by frequency from high to low
for c in sorted_conditions.reverse() do           ▷ Process the rarest conditions first
    for path in pool do
        if path.contains(c) then
            paths.append(path)
            pool.remove(path)
        end if
    end for
end for
return sorted_conditions, paths
```

Algorithm 1. Occurrence-based double reordering (ODR).

### 3.4.4 Similarity-Based Path Clustering (SPC)

Since memory redundancy arises from unencountered conditions, clustering paths with high similarity (i.e., those that share many conditions) can substantially improve space efficiency. Although *naive independent mapping* can be viewed as a basic form of clustering, paths within a tree exhibit limited similarity, and some TCAMs can still remain nearly redundant especially in cases where $\#paths = S + 1$ and the TCAM size is $S \times S$, leading to inefficient capacity utilization. Finding the optimal clusters to minimize TCAM consumption is an NP-hard problem. In this section, we propose *SPC*, a heuristic algorithm that greedily maximizes similarity among paths within TCAM, and thus minimizes CAM capacity requirement. As presented in Algorithm 2, *SPC* starts with a *pool* containing all root-to-leaf paths in the model. While *pool* is not empty, *SPC* evaluates

the similarity between *current_cluster* and each of the paths within *pool*, and designates the path that can fit in *current_cluster* based on (1) the highest number of shared conditions, and (2) the smallest resulting set of unique conditions after being integrated into *current_cluster*, as *candidate*. Since the objective is to accommodate all paths using the smallest number of clusters, and the capacity of each cluster depends on both *#paths* and *#unique_conditions* because we limit a cluster to a single TCAM, the key intuition behind *SPC* is to minimize *#unique_conditions* at every step. This approach allows each TCAM to hold as many paths as possible, resulting in an effective heuristic. Once *current_cluster* reaches capacity or none of the paths is suitable, *current_cluster* is added to *clusters* representing the completed ones, and a new empty cluster is initiated. Otherwise, *SPC* greedily adds the candidate to *current_cluster* and removes it from *pool*, and this process is repeated until all paths have been assigned to clusters.

*SPC* efficiently exploits similarities across paths, diminishing unnecessary X states. By filling TCAMs as fully as possible, it maximizes utilization and minimizes wasted capacity. Since each cluster is constrained to contain at most *S* unique conditions and *S* paths, there is no dependency between the match results of different TCAMs, eliminating any computational overhead associated with intersecting partial match results. Therefore, the only computational overhead lies in the query packing for every TCAM, which is the same as *naive independent mapping*. However, since *SPC* requires far fewer TCAMs than *naive independent mapping*, it emerges as a significantly more resource-efficient approach in terms of both energy and memory consumption.

**Algorithm 2** Similarity-Based Path Clustering (SPC)

**Require:** $S$ = TCAM_size
**Require:** *pool* = root_to_leaf_paths
**Ensure:** *clusters*

```
clusters ← {}, current_cluster ← {}
current_num ← 0
while pool is not empty do
    similarity ← calculate_similarity(pool, current_cluster)
    candidate ← find_best_candidate(similarity, pool)
    if candidate is NULL or current_num == S then
        clusters.append(current_cluster)
        current_cluster ← {}
        current_num ← 0
    else
        current_cluster.append(candidate)
        current_num ← current_num + 1
        pool.remove(candidate)
    end if
end while
if current_cluster is not empty then
    clusters.append(current_cluster)
end if
return clusters
```

Algorithm 2. Similarity-based path clustering (SPC).

# Chapter 4
# Evaluation

## 4.1 Experiment Setup

To evaluate the effectiveness of RETENTION, we compare the number of required TCAMs against those in existing works. Two ensemble models are selected: Random Forest, representing bagging-based methods, and XGBoost, representing boosting-based methods. Both models are trained on five datasets from the widely used UCI Machine Learning Repository [39], as listed in Table 3. Each dataset is divided into training and testing sets in a 7:3 ratio to examine the impact of *purity threshold pruning* on accuracy. For Random Forest, we utilized the open-source library Ranger [40] for forest construction, setting the number of trees to 100. *Purity threshold pruning* is applied on Random Forest to reduce model complexity, with tolerance set to {1%, 2%, 3%, 4%, 5%}. On the other hand, we used the official XGBoost library [9] to construct XGBoost with default parameters, including *max_depth=6* and *num_trees=100*. To achieve resource-efficient acceleration, we implemented *ODR* and *SPC* to further enhance efficiency. As analyzed in Section 3.4.2, the data placement strategies presented in prior works are mostly equivalent to *naive unified mapping* and *naive independent mapping*, which fall under the categories of energy-efficient mapping and space-efficient mapping, respectively. While *FR* incorporates additional optimizations, the associated computational overhead is comparable to *naive independent mapping*, and is therefore categorized into space-efficient mapping. Since the algorithm is originally designed for

ACAM, we make a minor modification, reordering conditions instead of features, to adapt it for TCAM deployment. In this work, energy-efficient mapping and space-efficient mapping are evaluated separately, with *naive unified mapping* and *naive independent mapping* as baselines. All models are mapped to TCAMs with size $64 \times 64$ if not specified. We select the reduced number of TCAMs as our evaluation metric, and therefore the higher value represents better performance.

| Dataset | #Samples | #Features | #Classes | Characteristic |
|---|---|---|---|---|
| Adult [41] | 48842 | 14 | 2 | Prestigious |
| CreditApproval [42] | 690 | 16 | 2 | Small |
| DryBean [43] | 13611 | 16 | 7 | Multi-class |
| Letter [44] | 20000 | 16 | 26 | Multi-class |
| Wine [45] | 4898 | 11 | 11 | Multi-class |

Table 3. Information of datasets.

The subsequent sections present detailed experimental results: Section 4.2 analyzes RETENTION's overall performance. Section 4.3 investigates the effects of *purity threshold pruning* on accuracy and TCAM capacity requirement. Section 4.4 evaluates the contributions of *ODR* and *SPC* individually. Section 4.5 explores the impact of the number of trees. Section 4.6 examines the influence of TCAM size. and Section 4.7 discusses computational overhead, energy consumption, and throughput.

## 4.2 Experiment 1: Overall Performance of RETENTION

To evaluate the overall performance of RETENTION, we compare Random Forest models with tolerance set to {1%, 3%, 5%} against baseline methods. As shown in Fig. 5, RETENTION exhibits substantial reductions, achieving 66.25% to 99.96% fewer TCAMs required (equivalent to 2.96× to 2420.81× better space efficiency) in energy-efficient mapping, and 70.83% to 99.70% reduction (3.43× to 334.48×

improvement) in space-efficient mapping. Specifically, setting tolerance to a moderate value of 3% yields 4.35× to 207.12× better space efficiency. These results demonstrate that RETENTION effectively minimizes CAM capacity requirement, offering a promising solution for resource-efficient tree-based model acceleration with CAM.

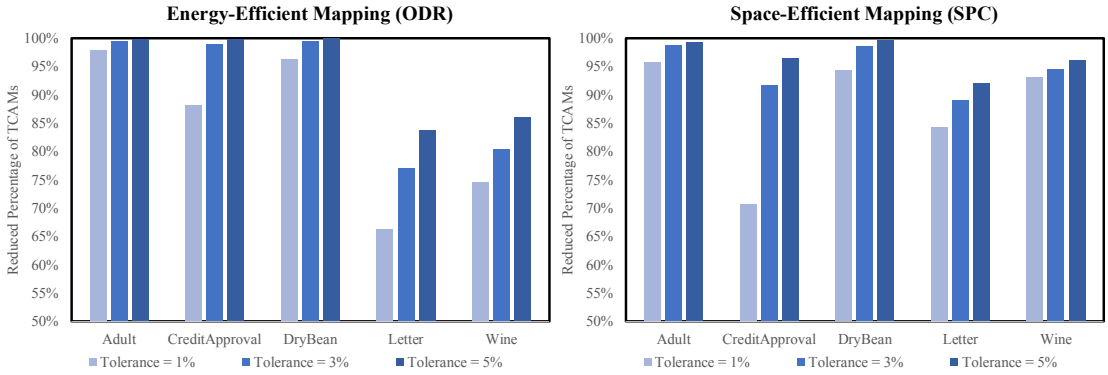



Fig. 5. Overall performance of RETENTION.

### 4.3 Experiment 2: Purity Threshold Pruning

Noticing the extraordinary performance of RETENTION, we break down the framework and evaluate each component separately. This section analyzes the effectiveness of *purity threshold pruning* by mapping Random Forest models with *naive unified mapping*. By setting tolerance to {1%, 3%, 5%}, *purity threshold pruning* alone achieves considerable improvements, reducing CAM capacity requirement by 21.04% to 99.93% (1.27× to 1357.12× improvement) compared to unpruned models. The relatively lower improvement on the Letter and Wine datasets stems from the increased number of classes, which makes it more challenging for nodes to achieve high purity. Pruning such nodes may result in significant accuracy degradation, which the tolerance mechanism prevents. Consequently, only a few higher-purity nodes can be pruned, thereby limiting the impact of the *purity threshold pruning*. We further scale tolerance up to {10%, 15%, 20%} to investigate the correlation between tolerance and different datasets. As shown in

the left part of Fig. 6, the performance of *purity threshold pruning* converges at different levels of tolerance, depending on task complexity. Though all can achieve over 90% reduction, simple tasks easily converge with low tolerance ({1%, 3%, 5%}, refer to the red columns), while the convergence for complex tasks occurs with relatively higher tolerance ({10%, 15%, 20%}, refer to the blue columns). However, for imbalanced datasets such as Adult, higher tolerance may lead to a model always favoring the majority class. This should be carefully dealt with by leveraging techniques such as class weights, as discussed in Section 3.2. The right part of Fig. 6 exhibits that the user-specified tolerance is highly correlated to testing accuracy loss. Since tolerance is the expected accuracy degradation that is sacrificed to achieve lower model complexity, ideally the testing accuracy loss should be lower than or similar to tolerance. Results show that most of the testing accuracy losses are lower than the predefined tolerance, with only one case slightly higher, suggesting that *purity threshold pruning* effectively reduces model complexity while ensuring controlled accuracy degradation. Models pruned with higher tolerance also follow this trend, though we omitted the statistics here for clarity.

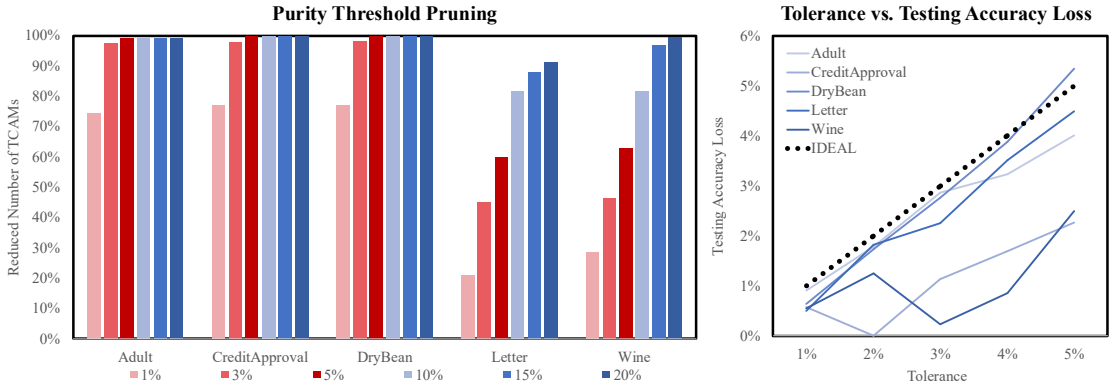



Fig. 6. Effectiveness of purity threshold pruning.

### 4.4 Experiment 3: Tree Mapping Scheme

After validating *purity threshold pruning*, we estimate the effectiveness of the proposed data placement strategies. Fig. 7 presents the results of implementing *ODR* and *SPC* on XGBoost and unpruned Random Forest. Since unpruned Random Forest is usually more complex than XGBoost, the improvement of applying *ODR* and *SPC* on Random Forest is often more obvious. For energy-efficient mapping, *ODR* presents more than 50% reduction in all cases. Since *#unique_conditions* of the models trained on CreditApproval and Letter datasets is an order of magnitude fewer than in other datasets, the improvement observed in the Random Forest trained on these two datasets is relatively limited. For space-efficient mapping, *SPC* exhibits notable improvement, with over 30% reduction in every case, while *FR* performs worse than the baseline in half of the cases. The underlying reason is that *FR* only eliminates completely redundant path segments within a TCAM, whereas significant redundancy is still left behind. Thus, *FR* beats *naive independent mapping* only when the redundancy within a single tree is overwhelming. On the other hand, by greedily clustering similar paths together, *SPC* minimizes redundancy while incurring less runtime computational overhead, consistently outperforming *FR* and *naive independent mapping* in all cases. The performance of applying *SPC* to the Random Forest trained on the CreditApproval dataset is nearly optimal (*SPC* requires 99 TCAMs, while the theoretical minimum is 98 for 6246 paths). However, the exceptionally strong performance of *naive independent mapping* on this model limits the potential for further improvement. Experimental results show that implementing the tree mapping scheme alone yields 1.46× to 21.30× better space efficiency, highlighting the effectiveness of both *ODR* and *SPC*.

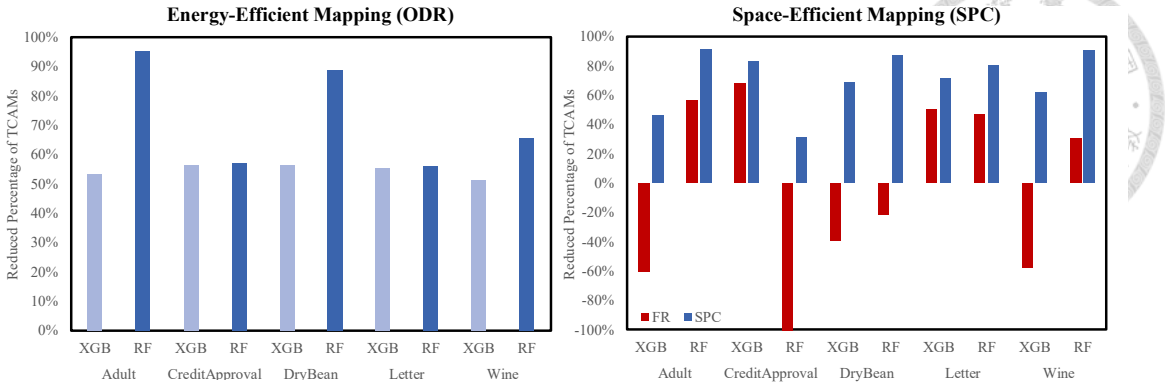



Fig. 7. Comparison of data placement strategies (RF: Random Forest, XGB: XGBoost).

## 4.5 Experiment 4: Number of Trees

Prior experiments demonstrate that RETENTION is extremely effective in minimizing CAM capacity requirement. In this section, we study the impact of number of trees on RETENTION's performance. Since XGBoost can achieve higher accuracy with an increased number of trees and appropriate hyperparameter tuning, XGBoost with *num_trees* set to {100, 300, 500} are evaluated in this experiment. As shown in Fig. 8, the performance of RETENTION tends to improve as *num_trees* increases because the inefficient issue is amplified by *num_trees*. However, since the training phase incorporates randomness, and the selected conditions for splitting nodes can strongly affect RETENTION's performance, the result may not always follow the trend.

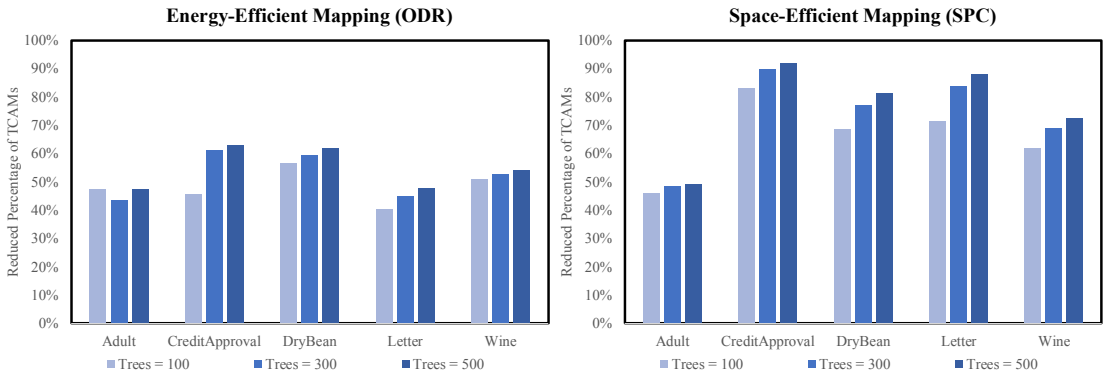
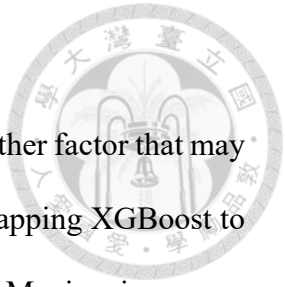



Fig. 8. Impact of number of trees on the performance of data placement strategies.



### 4.6 Experiment 5: Different TCAM Sizes

In addition to the number of trees, the size of the TCAM is another factor that may affect RETENTION's performance. Fig. 9 illustrates the impact of mapping XGBoost to TCAMs with three different sizes. Results suggest that as TCAM size increases, RETENTION yields gradually diminishing improvements in energy-efficient mapping, while offering notable gains in space-efficient mapping. Although the designs of both *naive unified mapping* and *ODR* are not directly correlated with TCAM size, it becomes increasingly difficult for *ODR* to eliminate TCAMs as the size increases, since only those entirely filled with X states can be removed. In some edge cases, such as mapping the models trained on the CreditApproval or Letter datasets to 256 × 256 TCAMs, *ODR* may offer no improvement because none of the TCAMs are fully redundant even after reordering. Conversely, increasing the TCAM size can sometimes create partially redundant TCAMs with only a few non-X-state cells. In such cases, *ODR* can effectively mitigate this issue, allowing these TCAMs to be removed and improving space efficiency. For example, mapping the Letter model to 128 × 128 TCAMs may benefit from this effect. On the other hand, although improvement increases with TCAM size when RETENTION is applied for space-efficient mapping, CAM utilization actually decreases because larger TCAMs require more X states per row, which leads to greater inefficiency. However, since *naive independent mapping* assigns each tree to a separate TCAM, larger capacities often fail to be fully exploited, leading to greater performance degradation compared to *SPC*. Consequently, a smaller TCAM is recommended for RETENTION regardless of the optimization objective, though it must still be large enough to accommodate the longest path in the model.

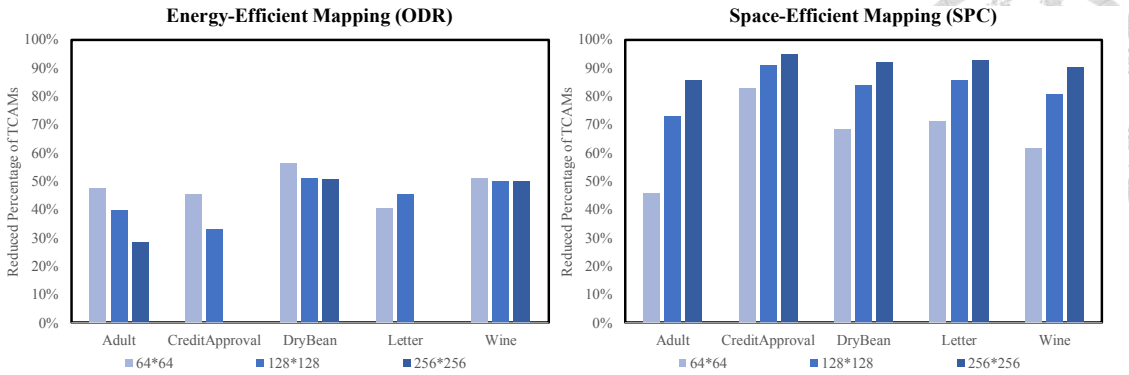


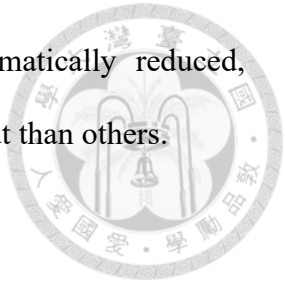

Fig. 9. Impact of TCAM sizes on the performance of data placement strategies.

### 4.7 Computational Overhead, Energy Consumption, and Throughput

For energy-efficient mapping, both *naive unified mapping* and *ODR* share the same input sequence among TCAMs in the same column, and a path is stored in a single row to prevent additional computational overhead for result retrieval. For space-efficient mapping, *naive independent mapping* and *SPC* require a unique input sequence for each TCAM, leading to higher computational overhead. However, as *naive independent mapping* stores paths to independent rows, and *SPC* stores paths in a single TCAM, none of them require further processing for result retrieval. Although *FR* prevents additional input preprocessing by sharing input sequences among TCAM columns, the paths are no longer stored in the same row after the row elimination process, and the associated computational overhead for result retrieval is proportional to *#TCAMs*. Therefore, considering computational overhead, the complexity of both *ODR* and *SPC* is comparable to previous works.

In terms of energy consumption, the reduction in the number of TCAMs is equivalent to energy savings. Furthermore, the energy-saving precharge mechanism proposed in DT2CAM is compatible with *ODR*, enabling additional reductions in energy consumption. Considering throughput, since RETENTION does not require any hardware modification, and the computational overhead is comparable to or even slightly lower

than previous works as the required number of TCAM is dramatically reduced, RETENTION is supposed to offer a similar or even higher throughput than others.



# Chapter 5
# Conclusion

nvTCAM shows great promise to accelerate tree-based model inference effectively and efficiently, yet the CAM capacity requirement is unacceptably high, and most of the cells are storing the X state for format alignment, which is redundant. In this work, we introduce RETENTION, an end-to-end framework designed to reduce the CAM capacity requirement for tree-based model inference. RETENTION introduces *purity threshold pruning* to minimize model complexity while ensuring controlled accuracy degradation for bagging-based models. A tree mapping scheme with two data placement strategies, *occurrence-based double reordering* and *similarity-based path clustering*, is proposed to further alleviate memory redundancy for energy-efficient mapping and space-efficient mapping. Experimental results show that implementing the tree mapping scheme alone achieves 1.46× to 21.30× better space efficiency, while the full RETENTION framework yields 4.35× to 207.12× improvement with less than 3% accuracy loss. These results demonstrate that RETENTION is extremely effective in reducing CAM capacity requirement, offering a resource-efficient solution for tree-based model acceleration. By implementing RETENTION, resource retention becomes feasible, bringing CAM-based acceleration closer to practicality in resource-constrained environments.

# References


[1] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. Inf. Fusion 81, C (May 2022), 84–90. https://doi.org/10.1016/j.inffus.2021.11.011

[2] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In Proceedings of the 36th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS '22). Curran Associates Inc., Red Hook, NY, USA, Article 37, 14 pages.

[3] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. Nature Machine Intelligence 2, 1 (2020), 56–67. https://doi.org/10.1038/s42256-019-0138-9

[4] DARPA, 2017. Explainable Artificial Intelligence (XAI). https://www.darpa.mil/research/programs/explainable-artificial-intelligence

[5] Yizhen Zheng, Huan Yee Koh, Jiaxin Ju, Anh T. N. Nguyen, Lauren T. May, Geoffrey I. Webb, and Shirui Pan. 2025. Large language models for scientific discovery in molecular property prediction. Nature Machine Intelligence (2025). https://doi.org/10.1038/s42256-025-00994-z

[6] Jonathan Kwaku Afriyie, Kassim Tawiah, Wilhemina Adoma Pels, Sandra Addai-Henne, Harriet Achiaa Dwamena, Emmanuel Odame Owiredu, Samuel Amening Ayeh, and John Eshun. 2023. A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions. Decision Analytics Journal 6 (2023), 100163. https://doi.org/10.1016/j.dajour.2023.100163

[7] Rongtao Zhang, Binbin Li, and Bin Jiao. 2019. Application of XGboost Algorithm in Bearing Fault Diagnosis. IOP Conference Series: Materials Science and Engineering 490, 7 (apr 2019), 072062. https://doi.org/10.1088/1757-899X/490/7/072062

[8] Kaggle. 2021. Kaggle Machine Learning & Data Science Survey 2021. https://www.kaggle.com/kaggle-survey-2021

[9] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (San Francisco, California, USA) (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[10] Francesco Daghero, Alessio Burrello, Enrico Macii, Paolo Montuschi, Massimo Poncino, and Daniele Jahier Pagliari. 2024. Dynamic Decision Tree Ensembles for Energy-Efficient Inference on IoT Edge Nodes. IEEE Internet of Things Journal 11, 1 (2024), 742–757. https://doi.org/10.1109/JIOT.2023.3286276

[11] Zhen Xie, Wenqian Dong, Jiawen Liu, Hang Liu, and Dong Li. 2021. Tahoe: tree structure-aware high performance inference engine for decision tree ensemble on GPU. In Proceedings of the Sixteenth European Conference on Computer Systems (Online Event, United Kingdom) (EuroSys '21). Association for Computing Machinery, New York, NY, USA, 426–440. https://doi.org/10.1145/3447786.3456251

[12] Adrián Alcolea and Javier Resano. 2021. FPGA Accelerator for Gradient Boosting Decision Trees. Electronics 10 (01 2021), 314. https://doi.org/10.3390/electronics10030314

[13] Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. 2012. Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?. In 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines. 232–239. https://doi.org/10.1109/FCCM.2012.47

[14] Kostas Pagiamtzis and Ali Sheikholeslami. 2006. Content-addressable memory (CAM) circuits and architectures: a tutorial and survey. IEEE Journal of Solid-State Circuits 41, 3 (2006), 712–727. https://doi.org/10.1109/JSSC.2005.864128

[15] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In Proceedings of the 43rd International Symposium on Computer Architecture (Seoul, Republic of Korea) (ISCA '16). IEEE Press, 14–26. https://doi.org/10.1109/ISCA.2016.12

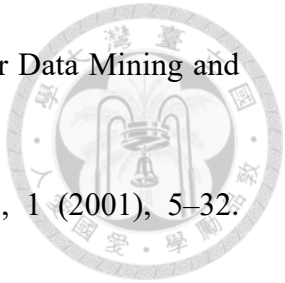
[16] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). 27–39. https://doi.org/10.1109/ISCA.2016.13

[17] An Chen. 2016. A review of emerging non-volatile memory (NVM) technologies and applications. Solid-State Electronics 125 (07 2016). https://doi.org/10.1016/j.sse.2016.07.006

[18] Le Zheng, Sangho Shin, Scott Lloyd, Maya Gokhale, Kyungmin Kim, and Sung-Mo Kang. 2016. RRAM-based TCAMs for pattern search. In 2016 IEEE International Symposium on Circuits and Systems (ISCAS). 1382–1385. https://doi.org/10.1109/ISCAS.2016.7527507

[19] Ke-Ji Zhou, Chen Mu, Bo Wen, Xu-Meng Zhang, Guang-Jian Wu, Can Li, Hao Jiang, Xiao-Yong Xue, Shang Tang, Chi-Xiao Chen, and Qi Liu. 2022. The trend of emerging non-volatile TCAM for parallel search and AI applications. Chip 1, 2 (2022), 100012. https://doi.org/10.1016/j.chip.2022.100012

[20] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the



scikit-learn project. In ECML PKDD Workshop: Languages for Data Mining and Machine Learning. 108–122.

[21] Leo Breiman. 2001. Random Forests. Machine Learning 45, 1 (2001), 5–32. https://doi.org/10.1023/A:1010933404324

[22] Mariam Rakka, Mohammed E. Fouda, Rouwaida Kanj, and Fadi Kurdahi. 2023. DT2CAM: A Decision Tree to Content Addressable Memory Framework. IEEE Transactions on Emerging Topics in Computing 11, 3 (2023), 805–810. https://doi.org/10.1109/TETC.2023.3261748

[23] Giacomo Pedretti, Catherine E. Graves, Sergey Serebryakov, Ruibin Mao, Xia Sheng, Martin Foltin, Can Li, and John Paul Strachan. 2021. Tree-based machine learning performed in-memory with memristive analog CAM. Nature Communications 12, 1 (2021), 5806. https://doi.org/10.1038/s41467-021-25873-0

[24] Giacomo Pedretti, John Moon, Pedro Bruel, Sergey Serebryakov, Ron Roth, Luca Buonanno, Archit Gajjar, Lei Zhao, Tobias Ziegler, Cong Xu, Martin Foltin, Paolo Faraboschi, Jim Ignowski, and Catherine Graves. 2024. X-TIME: Accelerating Large Tree Ensembles Inference for Tabular Data With Analog CAMs. IEEE Journal on Exploratory Solid-State Computational Devices and Circuits PP (01 2024), 1–1. https://doi.org/10.1109/JXCDC.2024.3495634

[25] Chieh-Lin Tsai, Chun-Feng Wu, Yuan-Hao Chang, Han-Wen Hu, Yung-Chun Lee, Hsiang-Pang Li, and Tei-Wei Kuo. 2023. A digital 3D TCAM accelerator for the inference phase of Random Forest. In 2023 60th ACM/IEEE Design Automation Conference (DAC). 1–6. https://doi.org/10.1109/DAC56929.2023.10247695

[26] Leo Breiman. 1996. OUT-OF-BAG ESTIMATION. https://api.semanticscholar.org/CorpusID:17166335

[27] Leo Breiman, Jerome Friedman, R. A. Olshen, and Charles J. Stone. 1984. Classification and Regression Trees (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

[28] Leo Breiman. 1996. Bagging predictors. Mach. Learn. 24, 2 (Aug. 1996), 123–140. https://doi.org/10.1023/A: 1018054314350

[29] Robert E. Schapire. 1999. A brief introduction to boosting. In Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2 (Stockholm, Sweden) (IJCAI'99). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1401–1406.

[30] Robert Karam, Ruchir Puri, Swaroop Ghosh, and Swarup Bhunia. 2015. Emerging Trends in Design and Applications of Memory-Based Computing and Content-Addressable Memories. Proc. IEEE 103, 8 (2015), 1311–1330. https://doi.org/10.1109/JPROC.2015.2434888

[31] Igor Arsovski, Trevis Chandler, and Ali Sheikholeslami. 2003. A ternary content-addressable memory (TCAM) based on 4T static storage and including a current-race sensing scheme. IEEE Journal of Solid-State Circuits 38, 1 (2003), 155–158. https://doi.org/10.1109/JSSC.2002.806264

[32] Can Li, Catherine E. Graves, Xia Sheng, Darrin Miller, Martin Foltin, Giacomo Pedretti, and John Paul Strachan. 2020. Analog content-addressable memories with memristors. Nature Communications 11, 1 (2020), 1638. https://doi.org/10.1038/s41467-020-15254-4

[33] Zhezhi He, Jie Lin, Rickard Ewetz, Jiann-Shiun Yuan, and Deliang Fan. 2019. Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping. In Proceedings of the 56th Annual Design Automation Conference 2019 (Las Vegas, NV, USA) (DAC '19). Association for Computing Machinery, New York, NY, USA, Article 57, 6 pages. https://doi.org/10.1145/3316781.3317870

[34] Yao-Wen Kang, Chun-Feng Wu, Yuan-Hao Chang, Tei-Wei Kuo, and Shu-Yin Ho. 2020. On Minimizing Analog Variation Errors to Resolve the Scalability Issue of ReRAM-Based Crossbar Accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39, 11 (2020), 3856–3867. https://doi.org/10.1109/TCAD.2020.3012250

[35] Fan Yang, Wei hang Lu, Lin kai Luo, and Tao Li. 2012. Margin optimization based pruning for random forest. Neurocomputing 94 (2012), 54–63. https://doi.org/10.1016/j.neucom.2012.04.007

[36] Heping Zhang and Minghui Wang. 2009. Search for the smallest random forest. Statistics and its interface 2 (01 2009), 381. https://doi.org/10.4310/SII.2009.v2.n3.a11

[37] Deepti Sehrawat and Nasib Singh Gill. 2019. Smart Sensors: Analysis of Different Types of IoT Sensors. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI). 523–528. https://doi.org/10.1109/ICOEI.2019.8862778

[38] Tobias Bjerregaard and Shankar Mahadevan. 2006. A survey of research and practices of Network-on-chip. ACM Comput. Surv. 38, 1 (June 2006), 1–es. https://doi.org/10.1145/1132952.1132953

[39] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. 2024. The UCI Machine Learning Repository. https://archive.ics.uci.edu

[40] Marvin Wright and Andreas Ziegler. 2015. ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. Journal of Statistical Software 77 (08 2015). https://doi.org/10.18637/jss.v077.i01

[41] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20

[42] John Ross Quinlan. 1987. Credit Approval. UCI Machine Learning Repository. https://doi.org/10.24432/C5FS30

[43] 2020. Dry Bean. UCI Machine Learning Repository. https://doi.org/10.24432/C50S4B

[44] David Slate. 1991. Letter Recognition. UCI Machine Learning Repository. https://doi.org/10.24432/C5ZP40

[45] Cortez, Paulo, Cerdeira, A., Almeida, F., Matos, T. and Reis, J., 2009. Wine Quality. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T