DT2CAM: A Decision Tree to Content Addressable Memory Framework

Mariam Rakka, Mohammed E. Fouda, Rouwaida Kanj, and Fadi Kurdahi

Abstract—Decision trees are powerful tools for data classification. Accelerating the decision tree search is crucial for on-the-edge applications with limited power and latency budget. In this paper, we propose a content-addressable memory compiler for decision tree inference acceleration. We propose a novel "adaptive-precision" scheme that results in a compact implementation and enables an efficient bijective mapping to ternary content addressable memories while maintaining high inference accuracies. We also develop a resistive-based functional synthesizer to map the decision tree to resistive content addressable memory arrays and perform functional simulations for energy, latency, and accuracy evaluations. We study the decision tree accuracy under hardware non-idealities including device defects, manufacturing variability, and input encoding noise. We test our framework on various decision tree datasets including *Give Me Some Credit*, *Titanic*, and *COVID-19*. Our results reveal up to 42.4% energy savings and up to $17.8\times$ better energy-delay-area product compared to the state-of-art hardware accelerators, and up to 333 million decisions per sec for the pipelined implementation.

Index Terms—Ternary Content Addressable Memory, Decision Tree, Machine Learning, Hardware Compiler, Synthesizer.

1 Introduction

Machine Learning (ML) continues to play a crucial role in performing complex tasks that are characterized by "learnable" properties. While brain-inspired Deep Neural Networks (DNNs) are nowadays thriving in several fields including computer vision, autonomous driving, the Internet of Things (IoT), and smart industries, they are not applicable where interpretability and domain knowledge are required [1]. Some applications that require integrating hand-crafted solutions (and hence domain expertise and explainability) as part of the learning process include predictive maintenance, risk management, anomaly detection, and image recognition for purposes of medical diagnosis [2]. In particular, Decision Trees (DTs) are popular to perform explainable ML [3], this is known as DT-based ML.

Several hardware accelerators for DT-based ML are proposed in literature. Most of these are CPU, GPU, FPGA, or ASIC-based accelerators [4]–[6]. More recently, hardware accelerators based on emerging memories like In-Memory Computing (IMC) architectures have been proposed for DT-based ML [4], [7]. The need to accelerate DT algorithms by means of IMC is motivated by the promise of orders of magnitude gains in energy efficiency and throughput provided by such architectures. This is particularly pivotal in edge applications that have hard constraints on energy and latency [4], [8]. Ternary Content Addressable Memories (TCAMs) perform massively parallel search operations, and are IMC architectures that have proven to boost performance in terms of energy and latency [8].

DT graphs consist of paths (i.e., routes) that describe some rules on features and that terminate by leaf nodes storing class values. To perform inference on DTs, incoming data should "match" one single path to associate it with some output class. Classical architectures will perform sequential searches on the DT routes to find the matching one. Motivated by the fact that each route in a DT can be mapped to a TCAM row (where the route's feature rules are stored) and by the high search throughput offered by TCAMs, we propose *DT2CAM: a Decision Tree to Content Addressable Memory framework*. DT2CAM simulates the inference of DTs on CAMs in general and Resistive CAMs (ReCAMs) in particular. We summarize our contributions as follows: 1) We propose DT2CAM, a framework that bijectively maps any DT into TCAM units relying on a novel adaptive precision encoding scheme. 2) DT2CAM demonstrates high robustness characterized by a low accuracy drop in presence of hardware non-idealities. 3) Results show up to $1.7\times$ and $3.8\times$ reduction in energy dissipation and area respectively, as well as 1.6x gain in throughput compared to the similar SOTA hardware accelerator on analog CAMs [4].

The rest of the paper is organized as follows. In section II, we explain the proposed DT2CAM framework. Section III presents the implementation details, and Section IV elaborates on the results and compares the framework against other hardware accelerators. Section V concludes the work.

This work was partially supported by the National Science Foundation under award ECCS-2028782, as well as King Abdullah for Science and Technology under award ORA-2021-CRG10-4704.
M. Rakka and F. Kurdahi are with the Center for Embedded & Cyber-physical Systems, University of California-Irvine, Irvine, CA, USA 92697-2625
M. Fouda is with the Center for Embedded & Cyber-physical Systems, University of California-Irvine, Irvine, CA, USA 92697-2625 and is also with Nanoelectronics Integrated Systems Center (NISC), Nile University, Giza, Egypt.
R. Kanj is with the ECE Dept., American University of Beirut, Lebanon, 1107 2020
Manuscript received xxxx xx, xxxx; revised xxxx xx, xxxx.

2 Proposed DT2CAM Framework

DT2CAM comprises: 1-) DT-HW compiler and 2-) ReCAM functional synthesizer. The DT-HW compiler translates a DT graph to a structured Look-Up Table (LUT). The ReCAM functional synthesizer maps the LUT into ReCAM arrays and evaluates energy, latency, and accuracy via simulations. The detailed analysis and experiments can be found in [9].

2.1 DT-HW Compiler

DT-HW compiler maps a DT graph into a structured LUT in four steps: DT graph generation, tree parsing, column reduction, and ternary adaptive encoding step.

Decision Tree Graph Generation: For some dataset, a supervised DT model capable of performing multi-class classification is trained by relying on the Classification and Regression Trees (CART) algorithm [10]. The DT model is represented by a DT graph where internal nodes represent rules on the attributes or features, branches represent the decisions for the rules, and leaf nodes represent classes.

Tree Parsing: The DT-HW compiler parses the DT into its equivalent table of conditions; each row in the table represents a path in the DT from root to leaf, and $\#rows = \#TreePaths$. Subsequently, each row consists of condition(s) applied to at least one feature.

Column Reduction: The DT-HW compiler reduces the conditions on each feature to one single condition (or rule) per row. The incoming input features can then be easily compared against their respective features' rules. The single rule for some feature f_i in row j, $rule_{ij}$, specifies the range for f_i. We note that by construct, the DT enforces a continuous range for the rule definition in a given path (row). The rule can be defined using a comparator $\in$ {'0', '1', '2', 'NaN'} and two thresholds: ($Th1_{ij}$) and ($Th2_{ij}$). The comparator states '0', '1', '2', and 'NaN' represent a-) less than or equal, b-) greater than, c-) in-between, and d-) no rule for this feature in this row, respectively. In particular, if the comparator is '0' in a row for some feature f_i, an incoming input feature, f_{in_i}, should be less than or equal to $Th1_{ij}$ (equivalently, $f_{in_i} \in (-Inf, Th1_{ij}]$) to match f_i's rule in row j. When the comparator is '1', f_{in_i} should be greater than $Th1_{ij}$ to match the rule on f_i. In these two cases, $Th2_{ij}$ is ignored and hence represented as "NaN" in the reduced table. When the comparator is '2', f_{in_i} should belong to $(Th1_{ij}, Th2_{ij}]$ to match the rule.

Ternary Adaptive Encoding: In the final step, the DT-HW compiler encodes each feature rule relying on an "adaptive-precision" unary encoding scheme suitable for TCAM implementations. Note that the scheme exploits the "don't care" feature of the TCAM as will be explained next. The "adaptive-precision" technique optimizes the area by setting a feature-dependent encoded string length. Thus, the number of bits varies for the different features but remains constant for a specific feature across all rows. This ensures that the encoding scheme is compact and efficient. We refer to it as Ternary Adaptive Encoding. The number of encoding bits for a specific feature is determined by the number of respective unique threshold values identified in the preceding column reduction step. In particular, for a given feature f_i out of N features ($i \in 1, 2, ..., N$), the number of bits, n_i, needed to encode f_i depends on the number of unique thresholds over the m rows, $T_i = |\cup_{j=1}^{m} \{Th1_{ij}, Th2_{ij}\}|$, as $n_i = T_i + 1$. Hence, for N features, the total number of bits (n_{total}) that are eventually needed to encode the whole DT (excluding the leaf nodes that store the class labels) is $n_{total} = N_{branches} * \sum_i (n_i)$, where $N_{branches} = m$ is the number of branches or paths from the root to leaf nodes in the DT (or the number of leaf nodes). The encoding scheme employs unary codes in the 'normal' form [11]. The encoded bits belong to the basis $\{0, 1, x\}$; x denotes a "don't care". This facilitates bijective mapping of the rules into TCAM(s). The encoding for a given feature f_i is explained as: **1-** Sort the elements of $Th^{f_i} = \cup_{j=1}^{m} \{Th1_{ij}, Th2_{ij}\}$ in ascending order. **2-** Construct $n_i =' T_i + 1'$ exclusive ranges defined in the set $R_i = \{r_1 = (-Inf, min(Th^{f_i})], ..., r_n =]max(Th^{f_i}), +Inf)\}$; **3-** map the ranges in R_i to ascending unique normal unary codes, $u_{r_1}^{f_i}, ..., u_{r_{n_i}}^{f_i}$, each comprising n_i bits starting with the code '00...01' and ending with '11...11'. We encode input features relying on the same scheme, and each will be represented by one of the unique feature codes based on the exclusive ranges they satisfy. We rely on the above encoding to construct an LUT. Recall that the rule range is continuous for a given path and thus can be interpreted in terms of the union of a set of multiple consecutive exclusive ranges. When a feature spans multiple exclusive ranges, we rely on "don't care" bits denoted as "x" to encode the new union range. Hence, inputs belonging to the different exclusive ranges that construct the rule will result in a match in the TCAM. As such, for each rule $rule_{ij}$ of f_i in row j, we perform those two steps: 1) Find the set of exclusive ranges, $\{r_{LB}, r_{UB}\}$, spanned by $rule_{ij}$. $LB, UB \in \{1, .., n\}$. Then, 2) encode $rule_{ij}$ as $Idx = Find_{idx}(XOR(u_{r_{LB}}, u_{r_{UB}}) == 1)$, $u_{rule_{ij}} = Replace(u_{r_{LB}}, Idx, "x")$. $Find_{idx}(.)$ returns a list of indices satisfying a condition. $Replace(u, Idx, "c")$ replaces all the characters of string u in positions Idx by the character "c".

2.2 ReCAM Functional Synthesizer

The ReCAM functional synthesizer comprises two steps; **mapping** where the LUT, provided by DT-HW compiler, is mapped into ternary ReCAM arrays and **simulation** where the synthesizer evaluates energy, latency, and accuracy.

2.2.1 Mapping

A bit of "0", "1", or "x" in the LUT is mapped to a "01", "10", or "11", respectively, in the two resistive elements of a TCAM cell as shown in Fig. 2. Ideally, one TCAM array is used, and the total number of TCAM cells needed is equal to n_{total}. However, in practice, the number of TCAM cells depends on the design requirements and limitations in terms of energy efficiency, latency, and dynamic range.

Dynamic Range: Describes the voltage difference between a full match voltage, V_{fm}, and the one mismatch voltage, V_{1mm}. Its equation for a capacitive sensing design is adopted from [12]. Given a dynamic range limit, D_{limit}, we choose a target TCAM row size S to meet D_{limit}.

Organization, Latency and Energy Efficiency: For practical purposes, we also assume that the TCAM width (# of rows) would be S. Hence, multiple TCAMs are needed; specifically, the synthesized TCAM cells of the encoded LUT rules need to be divided among $N_t = N_{cwd} * N_{rwd}$ TCAM arrays (aka tiles) each of size $S \times S$ to guarantee practical correct operation, where $N_{cwd} = \lceil (n_{total}/\#rows\textbf{+1})/S \rceil$ and $N_{rwd} = \lceil \#rows/S \rceil$ represent the number of column-wise and row-wise TCAM tiles respectively. The **'+1'** in N_{cwd} is explained by the reserved decoder column discussed below. **1-** If the original size of the LUT is smaller than $S \times S$ (in that case N_{cwd}=N_{rwd}=1), the functional synthesizer needs to extend the table obtained from the encoding step by padding "don't care" cells to render the LUT size $S \times S$ (1). We reserve the first column of the TCAM array and

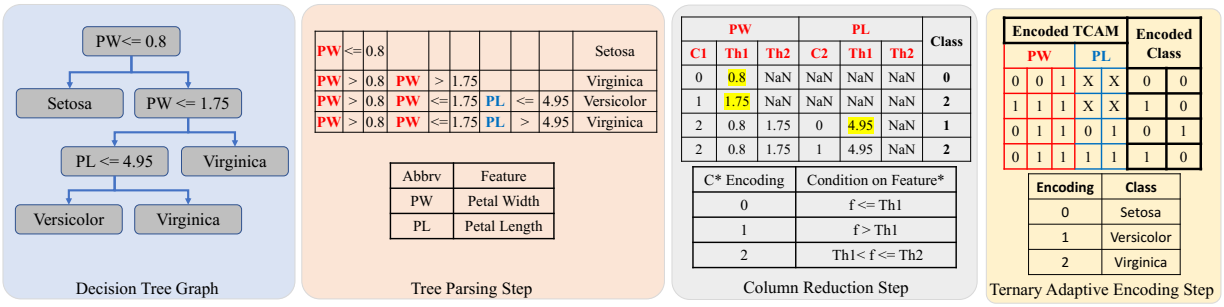


Fig. 1: DT-HW Compiler: Translates a DT graph to a structured LUT. From left to right: it first parses the DT and creates a table, then reduces the columns of the table, and then uses a "ternary adaptive encoding" scheme to create the LUT.

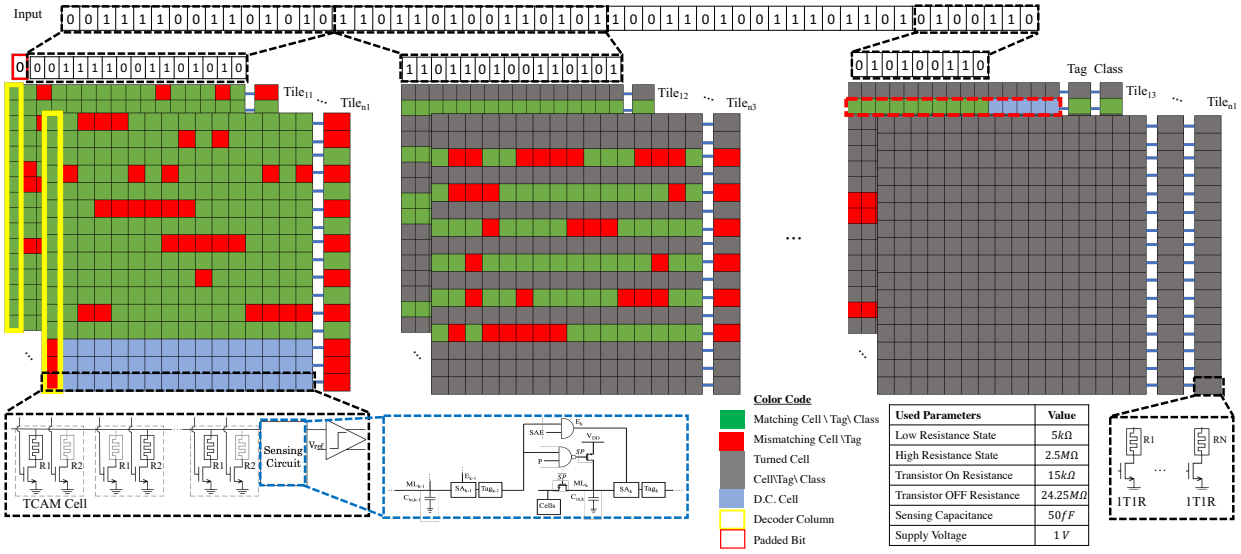


Fig. 2: ReCAM Functional Synthesizer: Maps the encoded LUT into $S \times S$ ReCAM arrays and runs energy, latency, and accuracy evaluations. SP circuits deactivate rows in the following tiles if the respective rows in the previous tiles mismatch. 16nm technology parameters are adapted from [12].

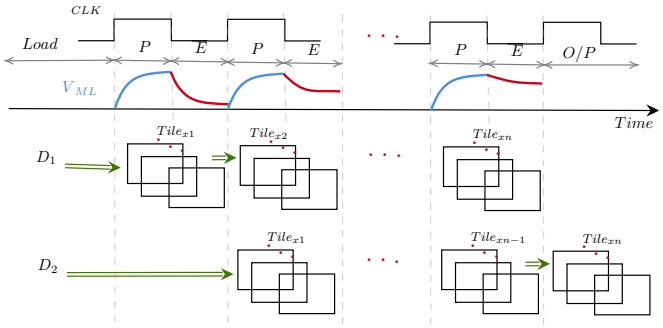


Fig. 3: Timing Diagram: $Tile_{x1}$ and $Tile_{x2}$ represent the row-wise tiles of the 1st and 2nd column-wise TCAM tiles. P and E stand for Precharge and Evaluate respectively. V_{in} is the voltage measured across C_{in}.

refer to it as the decoder column to enforce mismatch for the "rogue" rows that are not part of the original LUT.

2- Otherwise, it needs to divide that table into multiple TCAM tiles of size $S \times S$ as shown in Fig. 2. Tiles that are not completely filled by the LUT are padded by "don't care" cells. We reserve the first column of all TCAM arrays in the first division as decoder columns (see Fig. 2). For energy efficiency, the column-wise TCAM tiles are separated by row-enable bits that deactivate the rows in the following tiles if the respective rows in the previous tiles mismatch. By setting the decoder column bits to '1' for the rogue rows, we enable further energy savings since it forcibly mismatches the rogue rows. Aside from the decoder column, the remaining columns in the rogue rows are stored as "don't care cells". Each one of the $S \times S$ TCAMs has a column of S Sense Amplifiers (SAs) used to determine the match/mismatch status of each row. The class values corresponding to the rogue rows are populated with random values from the set of possible classes. We equip the row-wise tiles of the last column-wise division with an extra column of ReRAM cells, used to store the class bits (or equivalently the encoded leaf nodes' values of the DT). ReRAM cells are made of 1T1R cells, and each binary bit used to encode the classes is saved in one 1T1R cell. So, for a DT that has C possible classes, $\lceil log_2(C) \rceil$ bits (1T1R cells) are needed per row.

Input Processing and TCAM Mode of Operation: A '0' bit is padded at the beginning of the input. This padding along with decoder column bits enforces a mismatch in the rogue rows. For the rows that are part of the original LUT the padded bit matches with the decoder column bit. The original encoded input is then split across row-wise tiles of the column-wise tiles. Input pins that exceed the size of the encoded input may be assigned random inputs or may be masked. For the latter, the extended columns of the last

column-wise division are "masked", and the "masked don't care" cells have a pair of OFF-OFF transistors and do not dissipate energy. To exploit the parallel processing property of TCAMs whereby an input is processed in one shot across all TCAM rows, the row-wise tiles are allowed to operate in parallel. Moreover, to save precharge and evaluate energy we force a sequential operation on the column-wise TCAM tiles where no energy is dissipated in the following tiles upon mismatch in the previous tiles. The mode of operation is depicted in Fig. 3. Eventually, each encoded input must have one matching row in the row tiles of the last column division. We call this row the surviving row.

Selective Precharge: For each input, we evaluate column-wise TCAM tiles sequentially to enable Selective Precharge (SP) (shown in Fig. 2). With the SP circuit, a row that mismatches in the previous column-wise tile for some input is not precharged nor evaluated in the current tile. If an input mismatches a given row in some $Tile_{ij}$ (stage *k-1*), the SP circuit deactivates the precharge circuitry and SA of the corresponding row in $Tile_{ij+1}$ (stage *k*). Deactivating SA_k prevents the floating capacitor voltage residue from falsely flagging a match and activating the following tiles while $\bar{SP}$ preserves the charge to save energy during future precharges of the same tile. The SP circuit reduces the energy-delay product (see Section IV for details). If an input at stage *k-1* matches some row, the SA and precharge circuitry of the corresponding row in stage *k* are activated.

2.2.2 Simulation

The synthesizer performs simulations to evaluate energy, latency, and accuracy for the design with/without hardware non-idealities, with the following assumptions.

Technology: To calculate energy, latency, dynamic range, and optimal evaluation time (T_{opt}), we rely on $16nm$ technology parameters shown in Fig. 2.

Target Size: We determine the target size S values of the TCAM for $D_{limit} \in \{0.2, 0.3, 0.4, 0.5, 0.6\}$. For each D_{limit} value, we rely on dynamic range to determine the maximum number of TCAM cells per row allowed to satisfy this value. Finally, we choose a power-of-two target S value close to and lower than the maximum value. For example, for $D_{limit} = 0.2$, $max\#cells/row = 154$, so $S = 128$.

Energy: The total energy per an active TCAM row per input is calculated as $E_{row}^{active} = E_{TCAM} + E_{sa}$ where E_{sa} is the energy of the SA obtained via SPICE simulations. In particular, for a target size S, E_{sa} is the energy dissipated in the SA for a certain reference voltage capable of differentiating between a fully matching row and a row with one mismatch. In addition, E_{TCAM} and the optimal evaluation time, T_{opt}, are derived based on the closed form in [12]. We assume worst-case scenario for energy, where the extended cells in the row-wise tiles of the last column-wise division are treated like regular "don't care cells", hence dissipating energy as opposed to being masked. We note that we maintain the sequential functionality assuming null energy dissipation in rows that have been deactivated by the respective mismatching rows in previous tiles. Since the energy defined above is measured per row per input, the total energy for a given input is $E_{total} = \sum_1^{N_a} E_{row}^{active} + E_{mem}$, where N_a is the number of active rows for that input. E_{mem} is the energy needed to access the class label of the surviving row. We assume that class labels are stored in 1T1R cell(s) (total # of 1T1R cells $= log_2(\#classes)$) followed by a SA adapted from [13]. So, E_{mem} is the energy dissipated in the 1T1R cell(s) and the SA adapted from [13]. The average energy per input can then be computed from all inputs.

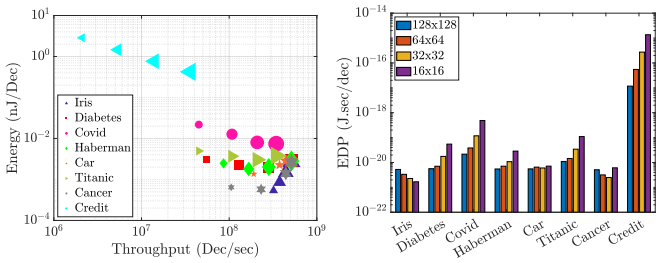
Latency: We define the column-wise latency, T_{cwd}, as the time needed to complete the inference per input per a column-wise tile according to $T_{cwd} = 3\tau_{pchg} + T_{opt} + T_{sa}$, where T_{sa} (determined via SPICE simulations) is the time needed for the SA to sense a match or a mismatch. The average total latency per input, $\bar{T}_{total}$, is then given by $\bar{T}_{total} = N_{cwd}T_{cwd} + T_{mem}$. T_{mem} is the time needed to access the 1T1R cell(s) storing the class label of the surviving row. Note that for multiple 1T1R cells, these are accessed in parallel. In addition, our simulator operates with the maximum frequency (unless otherwise mentioned) which is given as $f_{max} = (max(3\tau_{pchg} + T_{opt} + T_{sa}, T_{mem}))^{-1}$. For instance, the operating frequency for an array width of 128 is 1 GHz under the parameters reported in Fig. 2.

Hardware Non-idealities: We study the accuracy-wise robustness of our DT2CAM framework under device defects. In particular, we focus on a common problem in resistive TCAM cells: the fabrication-induced permanent Stuck-At-Fault (SAF) problem. Such fault cannot be writable as it is stuck at either High-Resistance State (HRS) (equivalently stuck at the bit "0" or $SA0$) or Low-Resistance State (LRS) (equivalently stuck at the bit "1" or $SA1$) [14]. We study the DT2CAM SAF problem in the presence of SA manufacturing variability similar to [15], and input noise. Thus, we perform Monte Carlo analysis to induce bit flips in the encoded TCAM cells. Our independent variables are: a-) SA reference voltage ($V_{ref} \sim N(\mu_{V_{ref}}, \sigma_{sa})$), b-) Stuck-At-Fault (SAF) (each cell is an event that gets stuck at "0" or "1" with probability $SA0/1$), and c-) input noise which is $\sim N(0, \sigma_{in})$. We sweep the probability percentage values as follows: $SA0 = [0, 0.1, 0.5, 1, 5]\%$ and $SA1 = [0, 0.1, 0.5, 1, 5]\%$. We emulate the SA variability by applying random offsets to reference voltage, V_{ref}, of the individual SAs for a given TCAM division where $\sigma_{sa} \in [0,\ 0.03,\ 0.04,\ 0.05,\ 0.1]V$. In addition, we study the effect of input noise where we induce random noise in the normalized input features dataset with $\sigma_{in} \in [0,\ 0.001,\ 0.005,\ 0.01,\ 0.02,\ 0.05,\ 0.1]$.

3 Implementation Details

We develop the DT2CAM framework in Python/MATLAB. Our Python-based compiler extracts and parses the DT model and further reduces it and produces an encoded LUT as shown in the last step of Fig. 1. Then, our MATLAB-based ReCAM functional synthesizer takes the LUT as an input and performs the mapping and hardware simulations. For testing, we use six datasets from UCI Repository and Kaggle [16], [17]. Particularly, we use the *Fisher's Iris* (Iris), *Haberman's Survival*, *Car Evaluation*, and *Breast Cancer Wisconsin (Diagnostic)* datasets from the UCI repository. *Give Me Some Credit (training)* and *Pima Indian Diabetes* datasets are taken from Kaggle. From Stanford's CS109 website [18], we utilize the *Titanic* dataset. We also evaluate our framework on a more recent dataset, *COVID-19* compiled by [19]. In some datasets, we omit some incomplete instances and some features that are unique for each data instance. We use

the same split percentage of the data in the aforementioned datasets to generate the DTs: 90%/10% for training/testing.



(a) (b)

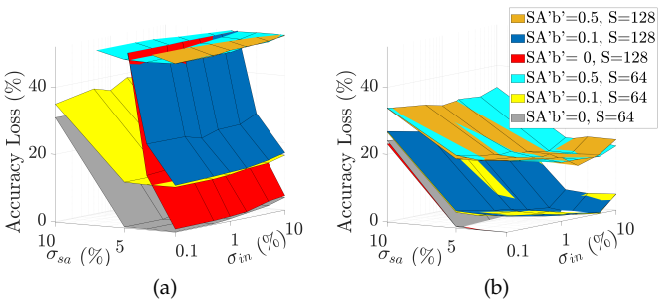
Fig. 4: Per inference decision: (a) Energy vs throughput for the different datasets. From small to large shapes: $S = 16 \times 16$, 32×32, 64×64, and 128×128. (b) Energy-Delay-Product.

4 Results and Comparison

Here, we discuss the ReCAM functional synthesizer results and compare DT2CAM to other hardware accelerators.

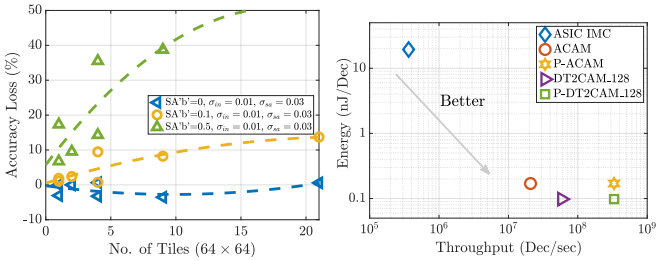
4.1 Energy/Throughput/EDP Analysis

Fig. 4a shows the energy per decision (dec) vs throughput for all datasets where $S \times S \in \{16 \times 16, 32 \times 32, 64 \times 64, 128 \times 128\}$. Larger markers indicate larger S values. Inference on Credit (largest dataset) consumes the highest energy and has the lowest throughput, while inference on Iris (smallest dataset) consumes almost the lowest energy and yields the highest throughput. This is expected as energy and throughput are dataset-size dependent. For Credit, Covid, Titanic, and Diabetes (relatively large datasets), increasing S results in reducing the per decision energy consumption (nJ/Dec) and increasing the throughput in terms of the number of decisions per second (Dec/sec). The energy reduction is due to a decrease in the number of switching blocks and SAs. The throughput improvement is attributed to the fact that the number of TCAM tiles operating sequentially for these datasets decreases with increasing S. Accordingly, the Energy-Delay Product (EDP) demonstrates improvement with increasing S as illustrated for these datasets in Fig. 4b. For the other datasets, the throughput (Dec/sec) improves with the target size demonstrating similar behavior as the previous ones. However, the energy consumption (nJ/Dec) increases with S. This is because small datasets are represented by at most two tiles when $S = 128$ thereby not benefiting from deactivated rows due to mismatching



(a) (b)

Fig. 5: % Accuracy loss due to hardware non-idealities for (a) Covid and (b) Cancer. $SA'b' = x$ means $SA0 = SA1 = x\%$.



(a) (b)

Fig. 6: (a)% Accuracy loss versus # of tiles. $SA'b' = x$ means $SA0 = SA1 = x\%$. (b) Energy vs. Throughput for our proposed DT2CAM and other SOTA hardware accelerators.

TABLE 1: Comparison with SOTA hardware accelerators. P= pipelined. We normalize to the P-DT2CAM.

Accelerator	Technology/f_{clk} (nm)/(GHz)	Normalized Throughput	Normalized Energy	Normalized Area	Normalized Area/bit	Normalized FOM
ASIC [6]	65/0.2	$9.01 * 10^{-8}$	$1.91 * 10^{+6}$	-	-	-
ASIC [20]	65/0.25	$1.80 * 10^{-7}$	$4.69 * 10^{+6}$	-	-	-
ASIC IMC [7]	65/1	$1.09 * 10^{-3}$	$1.98 * 10^{+2}$	-	-	-
ACAM [4]	16/1	$6.25 * 10^{-2}$	1.73	3.80	$1.76 * 10^{+1}$	$1.01 * 10^{+2}$
P-ACAM [4]	16/1	1	1.73	3.80	$1.76 * 10^{+1}$	6.33
DT2CAM_128	16/1	$1.77 * 10^{-1}$	1	1	1	5.67
P-DT2CAM_128	16/1	1	1	1	1	1

rows in previous tiles. Nevertheless, the throughput improvement is larger than the energy degradation (increase), and the EDP improves (decreases) with larger S values (Fig. 4b). Only the Iris dataset favors smaller S values when it comes to EDP due to its extremely small LUT size. From Fig. 4b, for all datasets where at least two column-wise tiles are required for different target size S, we see a reduction in the EDP when SP is used compared to when it is not. Particularly, the Credit dataset with SP circuit achieves the highest reduction in EDP (around 90%) because it is the largest dataset with the largest produced LUT, which in turn yields a large number of column-wise tiles. Column-wise tiles benefit from SP by evaluating few rows in each tile.

4.2 Analysis with Hardware Non-idealities

We study the accuracy loss in DT2CAM for different target sizes S and under the mentioned hardware non-idealities. Without loss of generality, we focus on Cancer and Covid datasets. For all the datasets under study, the accuracy evaluated by the ReCAM synthesizer for ideal hardware matches the *inference accuracy obtained in Python* (hereon denoted as *golden accuracy*). The accuracy loss of each dataset is measured compared to the corresponding golden accuracy. From Fig. 5, the target size S does not impact the accuracy loss in the presence of non-idealities for Cancer. For Covid (has a large number of tiles) a smaller S is more robust against non-idealities as the drop in accuracy is lower. This is clear for the case when $SA'b' = 0.1\%$ and $S = 64$ (yellow plane) and $S = 128$ (dark blue plane). The same holds for the case of $SA'b' = 0\%$. We truncate the cases for $SA'b' = 0.5\%$ for better illustration. Note that the probability of a defect falling in a division decreases with S. The variability induced in SAs affects the accuracy more severely compared to the noise in the input test datasets. In some cases, the input noise reduces the accuracy loss, and this is due to the test dataset itself, and how it changes with the input noise. The SAF problem affects the accuracy the most, as it can increase the % accuracy loss up to 50% (in the absence of other non-idealities), especially for large S.

4.3 Comparison with Other Hardware Accelerators

In Table 1 and Fig. 6b, we summarize the (normalized) per decision throughput and energy for our framework and other hardware accelerators for DT inference ([4], [6], [7], [20]). For DT2CAM, we assume a 2000×2048 original TCAM size, divided into 128×128 ($S = 128$) tiles to mimic inference on the traffic dataset problem. In particular, we take into consideration the 2000 rows by 256 features reported for the traffic dataset in [4], and further assume that each feature will require eight bits of storage (overestimation). We report the values for the sequential case (column-wise tiles operate sequentially) and pipelined case (column-wise tiles are pipelined). Compared to other SOTA accelerators, our proposed DT2CAM achieves up to seven orders of magnitude gain in throughput. Particularly, the pipelined version of DT2CAM can achieve $1.6 * 10\times$, $1.11 * 10^7\times$, $5.56 * 10^6\times$, $917\times$ throughput enhancement compared to [4] (non-pipelined), [6], [20], and [7] respectively. Furthermore, our pipelined design achieves $1.73\times$, $1.91 * 10^6\times$, $4.69 * 10^6\times$, and $1.98 * 10^2\times$ energy reduction compared to [4] (pipelined and non-pipelined), [6], [20], and [7]. [7] (SRAM-based IMC ASIC) offers advantages (energy and throughput) over traditional ASIC implementations [6], [20], where a local database is utilized. However, as mentioned in [4], achieving low energy and high throughput inference is still a challenge. [4] brings forth the advantages of memristive-based CAM arrays in terms of low power and high computational density. Our proposed design capitalizes on binary TCAM that does not undergo static currents, employs a ternary adaptive encoding scheme, and benefits from a selective precharge scheme. This results in energy savings compared to [4]. Non-pipelined DT2CAM has higher throughput than non-pipelined ACAM ($2.8\times$) because in our case we need 16 TCAM arrays compared to 29 ACAM arrays required by [4], which would give us at least 29/16 improvement given that we operate at the same frequency (1GHz). Moreover, they use extra periphery circuits which we do not use. Furthermore, the area efficiency of ACAM-based implementation has been shown in [4] to be much better than [7] (SRAM-based IMC) so we hereon compare to [4]. As shown in Table 1, compared to the area reported for the analog CAM framework [4], we achieve about $3.8\times$ and $17.6\times$ reduction in area overhead and area/bit respectively due to relying on 2T2R cells as opposed to analog CAM cells (whose area is larger). We define a figure of merit, FOM as Energy-Delay-Area (EDP) product to better compare the accelerators' performances. Accordingly, the lower the FOM, the better the performance. Our DT2CAM/P-DT2CAM framework has $17.79\times/6.33\times$ better FOM compared to the ACAM/P-ACAM realization.

5 Conclusion

In conclusion, we proposed DT2CAM, a DT to ReCAM framework that evaluates energy, latency, and accuracy of DT inference using (resistive) TCAMs with/without hardware non-idealities. DT2CAM has two phases: The DT-HW compiler maps a DT graph into an LUT, and the ReCAM functional synthesizer maps the LUT into ReCAM arrays and performs simulations. Experiments on various datasets show that the ternary adaptive encoding scheme adopted by the DT-HW compiler is robust against noise and efficient in terms of energy and latency. Compared to other SOTA hardware accelerators, DT2CAM achieves the lowest energy, highest throughput, lowest area overhead, and lowest FOM (preferred). As future work, we want to extend the framework to include other ReRAM cell topologies (like ACAM) and to support deep decision ensembles.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] N. Bussmann, P. Giudici, D. Marinelli, and J. Papenbrock, "Explainable machine learning in credit risk management," *Computational Economics*, vol. 57, no. 1, pp. 203–216, 2021.

[3] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable ai for trees," *Nature machine intelligence*, vol. 2, no. 1, pp. 56–67, 2020.

[4] G. Pedretti, C. E. Graves, S. Serebryakov, R. Mao, X. Sheng, M. Foltin, C. Li, and J. P. Strachan, "Tree-based machine learning performed in-memory with memristive analog cam," *Nature communications*, vol. 12, no. 1, pp. 1–10, 2021.

[5] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?" in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2012, pp. 232–239.

[6] T.-W. Chen, Y.-C. Su, K.-Y. Huang, Y.-M. Tsai, S.-Y. Chien, and L.-G. Chen, "Visual vocabulary processor based on binary tree architecture for real-time object recognition in full-hd resolution," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 12, pp. 2329–2332, 2011.

[7] M. Kang, S. K. Gonugondla, S. Lim, and N. R. Shanbhag, "A 19.4-nj/decision, 364-k decisions/s, in-memory random forest multi-class inference accelerator," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 7, pp. 2126–2135, 2018.

[8] M. E. Fouda, H. E. Yantir, A. M. Eltawil, and F. Kurdahi, "In-memory associative processors: Tutorial, potential, and challenges," *arXiv preprint arXiv:2203.00662*, 2022.

[9] M. Rakka, "Resistive content addressable memory design for decision tree acceleration thesis," Master's thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2022.

[10] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees. belmont, ca: Wadsworth," *International Group*, vol. 432, pp. 151–166, 1984.

[11] S. Kak, "Generalized unary coding," *Circuits, Systems, and Signal Processing*, vol. 35, no. 4, pp. 1419–1426, 2016.

[12] M. Rakka, M. E. Fouda, R. Kanj, A. Eltawil, and F. J. Kurdahi, "Design exploration of sensing techniques in 2t-2r resistive ternary cams," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 2, pp. 762–766, 2020.

[13] X. Sun, S. Yin, X. Peng, R. Liu, J.-s. Seo, and S. Yu, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1423–1428.

[14] I. Yeo, M. Chu, S.-G. Gi, H. Hwang, and B.-G. Lee, "Stuck-at-fault tolerant schemes for memristor crossbar array-based neural networks," *IEEE Transactions on Electron Devices*, vol. 66, no. 7, pp. 2937–2945, 2019.

[15] P.-F. Chiu, B. Zimmer, and B. Nikolić, "A double-tail sense amplifier for low-voltage sram in 28nm technology," in *2016 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2016, pp. 181–184.

[16] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[17] "Kaggle." [Online]. Available: https://www.kaggle.com

[18] "A titanic probability." [Online]. Available: https://web.stanford.edu/class/archive/cs/cs109/cs109.1166

[19] B. Xu, B. Gutierrez, S. Mekaru, K. Sewalk, L. Goodwin, A. Loskill, E. L. Cohn, Y. Hswen, S. C. Hill, M. M. Cobo *et al.*, "Epidemiological data from the covid-19 outbreak, real-time case information," *Scientific data*, vol. 7, no. 1, pp. 1–6, 2020.

[20] K. J. Lee, G. Kim, J. Park, and H.-J. Yoo, "A vocabulary forest object matching processor with 2.07 m-vector/s throughput and 13.3 nj/vector per-vector energy for full-hd 60 fps video object recognition," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1059–1069, 2015.