UNIVERSITY OF CALIFORNIA,
IRVINE

Resistive Content Addressable Memory Design for Decision Tree Acceleration

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF ENGINEERING

in Electrical and Computer Engineering

by

Mariam Rakka

2022

DEDICATION

To my parents, Mustafa Rakka and Zainab Haragli.
To my grandmas, my siblings, the love of my life, my friends, and my mentors.
This would not have been possible without their support, prayers, and blessings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

One step closer...

This thesis would not have been possible without the help of Allah and so many people. It is the fruit of my academic efforts and my discussions with my advisors and mentors.

I would like to express my utmost gratitude to Professor Fadi J. Kurdahi, my advisor and mentor, who has been supporting my research at UC Irvine since Summer 2019 and up-til now. His rich experience in the field, suggestions, and guidance were pivotal in my learning journey. His constant support throughout the ups and downs motivated me to keep pushing further.

I would also like to thank my undergraduate advisor and my all time mentor, friend, and idol, Professor Rouwaida Kanj for her constant support and encouragement. It was under her mentorship that I first discovered my passion for research, and for that I will always be grateful. This thesis would not have been possible without her academic and non-academic support.

I would also like to acknowledge Dr. Mohammed Fouda for his invaluable feedback on my research work and for his insights on many of the projects I participated in. His guidance was significant in helping me become a better researcher.

I would like to thank Professors Rainer Dömer and Al Faruque for being on my committee. Their invaluable feedback on this Thesis was of great help and benefit. Even though I did not conduct research with them, my observations of their contributions will be a good guide in my academic life.

I would also like to thank all my other colleagues and friends, especially Naji Tarabay and Rachid Karami for making my experience at UC, Irvine more enjoyable.

I would also like to acknowledge the use of materials in this work in Chapter 4 is a reprint of the material as it appears in IEEE Transactions on Circuits and Systems II: Express Briefs. The co-authors Fadi Kurdahi and Mohammed Fouda listed in this publication directed and supervised research which forms the basis for the thesis.

I would also like to acknowledge Mr. Samer Sleem, my high school teacher that believed in my abilities and suggested I should pursue this degree from the start.

I would like to express my gratitude to Mahmoud Ajami. I would like to thank him for being there for me, for never giving up on me, and for baring with me. This would not have been possible without him being in my life.

Finally, and most importantly, I would like to thank my family, who encouraged me and supported me to pursue this dream of mine. They have been the constant pushing force throughout all the hard times, and for that I can never be grateful enough.

ABSTRACT OF THE THESIS

Resistive Content Addressable Memory Design for Decision Tree Acceleration

By

Mariam Rakka

Master of Engineering in Electrical and Computer Engineering

University of California, Irvine, 2022

Professor Fadi J. Kurdahi, Chair

In-Memory Computing (IMC) is considered a great candidate to replace the von-Neumann computing architecture to overcome the memory wall. Content Addressable Memories (CAMs) are the main building blocks in IMC-based architectures, such as the associative processors, and they are being used to accelerate machine learning tasks such as inference on Decision Trees (DTs). Decision trees are popular and powerful tools for data classification. Accelerating the decision tree search is crucial for on-the-edge applications that have limited power and latency budget. In this paper, we first present a juxtaposition between the capacitive and resistive sensing schemes in 2 Transistor-2 Resistive (2T-2R) Ternary CAMs (TCAMs). A Figure of Merit (FOM), function of the dynamic range, latency, and energy, is defined to have a fair comparison between the two sensing techniques. A mathematical model for the transient behavior of both sensing schemes is derived and verified through SPICE simulations. We then study the performance of the two schemes with an in-memory addition application and the results reveal that resistive sensing has an edge in that context. In addition, we propose a CAM Compiler for DT inference acceleration. In particular, we propose a novel "adaptive-precision" scheme that results in a compact implementation and enables an efficient bijective mapping to TCAMs while maintaining high inference accuracies. Also, a Resistive-CAM (ReCAM) functional synthesizer is developed for mapping the decision tree to the ReCAM arrays with the capacitive sensing scheme and performing 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 DT datasets including *Give Me Some Credit*, *Titanic*, and *COVID-19*. Our results reveal up to 42.4% energy savings and up to 17.8x 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.

Chapter 1

Introduction

1.1 Motivation

We live in an era where Machine Learning (ML) is becoming a pivotal part of nearly every domain including cyber-security [2], medicine [3], biology [4], chemistry [5], astronomy [6], and many others. Decision Trees (DTs) are a family of supervised ML models that enable classification and regression, in a simple, interpretable, and comprehensible manner [7]. DT graphs that are used for classification 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, the incoming data should "match" one single complete path to associate it with some output class [8]. While one DT is able to produce highly accurate classifications, a collection or ensembles of DTs can produce best performing classifications [9]. Random forests and boosting are two popular ensemble DT techniques [8].

While we witness the end of Moore's law, continuous efforts are needed to enhance the full software-hardware stack in order to accommodate for the abundant deployment of ML algorithms in the most efficient, cost-friendly, yet effective manner [10]. Von Neumann architectures physically seperate between computation and storage: data are fetched from the memory unit, sent to the computing unit(s) and then sent back to the memory unit for storage. The memory wall is a limitation of classical computers due to the communication rate between the memory and computing units [11]. In-Memory Computing (IMC) architectures are gaining momentum for ML applications for they eliminate the memory wall problem that prevails in von Neumann architectures [12]. Their ability to perform the computation and storage in one place (the memory) eliminates most of the data communication overhead and saves the energy once needed for the data movement [11]. Content Addressable Memories (CAMs) perform massively parallel in-memory searching which boosts the performance in terms of energy and latency [13–17]. The Associative Processor (AP) is an IMC architecture based on CAMs that can perform computations in memory. APs can be used to perform both logical and arithmetic computations inside memory units of the CAM [18]. APs and CAMs are utilized for IP routing [19], databases [20], and accelerating matrix-vector-like operations [21]. As such, they present themselves as attractive candidates for accelerating various ML inference workloads.

Motivated by 1-) the fact that CAMs are flourishing for search purposes, 2-) the fact that each route in a DT can be logically mapped to a CAM row (where the route's feature rules are stored), and 3-) the high search throughput offered by CAMs, we propose DT2CAM: a Decision Tree to Content Addressable Memory framework. DT2CAM simulates the inference of decision trees on CAMs in general and resistive-based Ternary CAMs (TCAMs) in particular. We conduct a study on sensing techniques for CAMs preparatory to introducing DT2CAM, and demonstrate our results on an adder application simulated on an AP. Our contributions are summarized in the following section.

1.2 Contributions

In this paper, we first start by studying two sensing schemes for 2 Transistor-2 Resistive (2T-2R) TCAM, (1) a capacitive sensing scheme with optimal dynamic range considerations, and (2) a resistive sensing scheme that eliminates the precharge phase. Then, motivated by the fact that each route in a DT can be mapped to a TCAM row 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 decision trees on CAMs in general and resistive-based TCAMs with capacitive sensing schemes in particular. Our contributions can be summarized as follows.

1. We explore, for the two sensing schemes, improvements in performance metrics in terms of dynamic sensing range, energy, and latency.
2. We develop a mathematical model for the Memristive TCAM (MTCAM) operation for both sensing schemes and validate the proposed model against SPICE simulations.
3. We study the two sensing schemes in the context of an n-bit adder application.
4. We propose DT2CAM, a framework that bijectively maps any decision tree into TCAM units relying on a novel adaptive precision encoding scheme. DT2CAM comprises two components:
   (a) A decision tree to CAM-based hardware architecture compiler (DT-HW compiler). The DT-HW compiler translates a decision tree graph to a structured Look-Up Table (LUT) comprising 0, 1, and ”don't care” bits. The LUT rows represent encoded DT paths, and they can be mapped into any Ternary CAM architecture.
   (b) A Resistive-CAM functional synthesizer which maps the encoded LUT entries to resistive TCAM cells with capacitive sensing while taking into consideration

design requirements and specifications. It also performs simulations to evaluate energy, latency, and accuracy.

5. We study the robustness of the proposed DT2CAM framework given hardware non-idealities: manufacturing variability, device defects, and noise in the input dataset. Results prove high robustness characterized by a low accuracy drop compared to recognition accuracy.

6. We demonstrate for our proposed framework up to 42.4% reduction in energy dissipation compared to the similar state-of-the-art hardware accelerator on analog CAMs [22], high throughput, and low area overhead. Moreover, we define a Figure of Merit (FOM) that further shows that our proposed framework performs the best compared to the other frameworks. We also validate that the DT2CAM functional accuracy matches that of Python-based DT inference.

1.3 Organization

The rest of this paper is organized as follows. Chapter 2 provides an overview on in-memory computing. Then, chapter 3 elaborates on decision trees and the ways to accelerate their inference. In chapter 4, capacitive sensing and resistive sensing designs are presented along with their mathematical formulations. In addition, MATLAB and SPICE validation and simulation results are presented, along with an adder application. In chapter 5, we explain the proposed DT2CAM framework with the implementation details, and chapter 6 elaborates on the results and compares the framework against other hardware accelerators. Finally, chapter 7 concludes the work and presents the future work.

Chapter 2

In-Memory Computing

In von-Neumann architectures, processing data requires retrieving the data first from the memory units to compute units. This creates huge communication overhead across the system, hence degrading performance. With the rise of data intensive ML workloads, this degradation in performance has become more prominent. One solution to tackle that problem is to shift towards a data-centric computing paradigm by enabling IMC. IMC architectures eliminate the data transfer bottleneck by directly performing some of the desired computation inside memory, hence they eradicate the ”memory-wall” problem [12]. IMC architectures that rely on CMOS-based Static Random Access Memories (SRAMs) for logical and arithmetic computations have been explored in various research works [23, 24]. CMOS-based memory technologies are suitable candidates for performing Boolean and low-precision arithmetic computations. However, they suffer from scaling trends and limited density, which motivated the exploration of IMC architectures based on emerging Non-Volatile Memories (NVMs) [25]. Some examples of NVMs include Resistive Memories (ReRAMS or RRAMs) [26], Phase-Change Memories (PCMs) [27], and Spintronics (STT-MRAM) [28]. In this chapter, we focus on ReRAMs and some IMC architectures based on ReRAMs as these form the foundation of our work in this thesis.

2.1 Resistive Memories

ReRAMs are 2-port devices that have a Metal-Insulator-Metal (MIM) structure, as depicted in Fig. 2.1. By changing the applied voltage across the ReRAM device, a soft breakdown of the dielectric occurs whereby conducting filaments are created/destroyed. This results in generating multi-level conductance levels [29]. ReRAMs can provide ON/OFF resistance ratios and high density, and they have been used in 128x128 crossbars [30]. ReRAMs, on the other side, suffer from low endurance, high write energy, and high latency [25]. We note that in this thesis, we assume the memristor-based ReRAM device (unless mentioned otherwise), but the concepts can be easily applied for other MIM device based ReRAM designs. The memristor is a two-terminal element, where the magnetic flux between its terminals is a function of the amount of electric charge passing. The memristance has the Ohm unit, same as resistance. The memristor switches between two stable states when an external voltage is applied across its terminals. If a positive voltage above a specific threshold is applied, the memristor switches into the ”OFF state” with high resistance (this is known as the RESET operation). If a negative voltage of the same magnitude is applied, the memristor switches into the ”ON state” with low resistance (this is known as the SET operation) [31,32]. It is worth noting that low resistance state and high resistance state are referred to as LRS and HRS respectively.

2.2 Content Addressable Memories

Content Addressable Memories, CAMs, are considered one realization of IMC architectures that have proven to boost performance in terms of energy and latency by performing massively parallel search operations [13–17]. A CAM, typically depicted in Fig. 2.3, is capable of performing search on a large number of words in a single shot (i.e. in one cycle) [33].

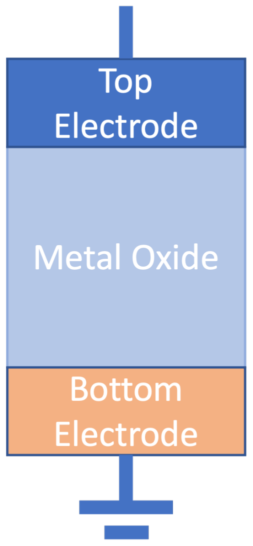


Figure 2.1: ReRAM structure.

CAM memories search the content (i.e. the data word) stored inside the memory to decide on its existence and location (address). This is unlike traditional memories which search for the stored data by their address. Due to the potential of CAMs, different implementations have been explored utilizing either SRAM or emerging devices such as resistive or phase change memories [34,35]. SRAM-based CAMs suffer from high power consumption and low-density [34], and to address these concerns, memristor-based CAMs were introduced in [35]. We will elaborate on the memristor-based (or resistive-based) CAM next.

2.2.1 The 2T-2R Cell: A Typical Building Block of Resistive-based CAMs

Each row of the CAM is composed of several CAM cells (see Fig. 2.3). A typical memristive (resistive)-based CAM cell structure consists of two memristors (resistive elements), two transistors, bitlines, and a matchline [36, 37]. Since it has 2 resistive elements and 2 transistors, it is known as 2T-2R cell. The 2T-2R cell is illustrated in Fig. 2.2. As the resistive elements in this case are assumed to be memristors, a CAM cell can store one of two bits: "0" or "1". By convention, a "0" bit is stored in the cell when the memristors

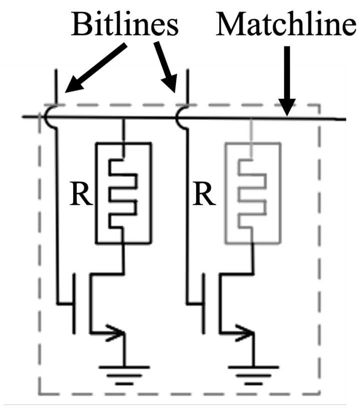


Figure 2.2: A typical 2T-2R structure. SA= Sense amplifier.

are LRS-HRS, while a "1" bit is stored when the memristors are in HRS-LRS. The cell can store an additional "don't care" state, whereby both memristors are in high resistance state. CAMs that allow the cells to have the "don't care" state are known as Ternary Content Addressable Memories (TCAMs). In TCAMs, partial searches are feasible by enabling "don't care" conditions on potions of the searched data word [37]. We next elaborate on the operation of TCAMs.

2.2.2 Operation of TCAMs

As depicted in Fig. 2.3, at the end of each "typical" (we introduce another type in chapter 4) CAM row, a capacitor (not shown in figure), a Sense Amplifier (SA), and a tag are connected to the match line as to allow the search operation to take place. The whole $N \times M$ TCAM structure is searched against one input in one clock cycle. In particular, the search operation comprises two stages that span a clock cycle: 1-) precharge and 2-) evaluate.

1. During precharge, all the N capacitors are charged to the supply voltage V_{DD}.
2. During evaluate, the M-bit data stored in the search data register are passed to the bitlines of the M TCAM cells of all rows in parallel. The voltage drops across the capacitors are measured by the SAs. Voltage drops that are below the SAs' sensing

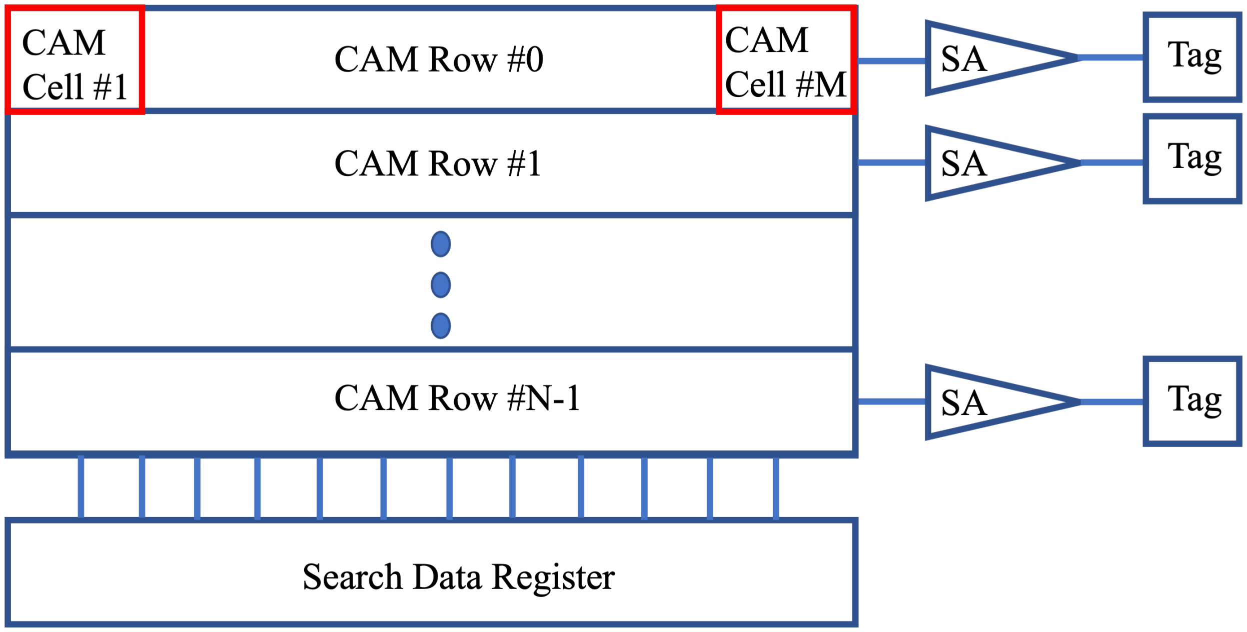


Figure 2.3: A typical content addressable memory structure.

voltage produce a logic-’0’ at the output of the respective SAs (indicating the respective TCAM rows are mismatched). On the other hand, voltage drops that exceed the sensing voltage of the SAs yield a logic-’1’ at the output of those SAs (indicating the respective TCAM rows are matched). SA outputs are latched into the tag in order to indicate the match/mismatch status of the TCAM rows.

The detailed operations of a 2T-2R cell and TCAMs are further elaborated in chapter 4.

2.2.3 Sensing Schemes

Several sensing schemes have been proposed to reduce the power or enhance the latency of the search operation of CAM designs [34, 38–41]. In conventional schemes, the match line (ML) is typically precharged high. During the evaluate phase, only the fully matched rows remain high [38]. The authors in [39] proposed a clocked self-referenced sensing scheme for 2T-2R PCM-based TCAM designs. The scheme dynamically modulates the precharge ML levels to intermediate values that enable power savings while maintaining good noise margins.

A selective hit-ML precharge sensing scheme was proposed for 2T-2R TCAM in [38]. In this scheme, the ML discharges only when there is a full match while the mismatched rows remain high; hence, there is only a need to selectively precharge the fully matched MLs. The authors in [34] proposed a capacitive sensing scheme for 2T-2R with optimal dynamic range considerations. More recently, [41] proposed an evaluate only sensing scheme suitable for nand-type SRAM-based TCAM. This scheme eliminates the precharge phase and performs search in half-cycle compared to other approaches.

2.3 Associative Processors

2.3.1 Architecture

Associative Processors, APs, are IMC architectures proposed back in the seventies, and they utilize associative memories to carry out massively parallel computations [42]. The general architecture of an associative processor is shown in Fig. 2.4. The architecture comprises the following components: 1-) a two-dimensional (2-D) CAM, 2-) an instruction cache, 3-) a controller, 4-) a key register, 5-) a mask register, 6-) an interconnection circuit, and a 7-) tag register. We summarize the functionality of the seven components below [17].

1. 2-D CAM: stores the data that is to be searched or computed. The CAM cell can have different implementations as we have mentioned earlier.
2. Instruction Cache: stores the instructions that will be executed by the AP.
3. Controller: generates a sequence of required mask and key bits for each instruction. The sequence follows the look-up table of the operation being performed. It is worth noting that the LUT of some operation is usually generated automatically by relying on a uni-directional state diagram with no cycles [43]. The sequence issued by the

controller avoids having any cycles that would overwrite the outcome of the earlier sequence pass.

4. Key register: stores the input bits that would be compared to the bits stored in the CAM cells.

5. Mask register: stores bits that dictate which bits (i.e. which CAM cells) are activated during comparison or write. Masked CAM cells are said to be inactive.

6. Interconnection circuit: a switching matrix that facilitates either bitwise or wordwise parallel communication between the AP rows (i.e. CAM rows).

7. Tag register: Each CAM row has an associated tag register, which stores the result of the performed comparison between the bits stored in the CAM rows and those passed by the key.

Completing an operation on the AP can be broken down into two phases: 1-) compare and 2-) write. We summarize both phases below [17, 18].

1. Compare: In this phase, the unmasked values in the key are compared against the values stored in the CAM rows (all CAM rows are compared in one cycle or shot), and the matched rows are selected via the tag. That means the tag will store logic-'1' for matching rows. The compare phase itself comprises two stages: a precharge stage where the matchline capacitor is precharged to the supply voltage, and an evaluate stage where the inputs to be searched are applied to the columns across the rows in parallel. During evaluate and if the stored data does not match the input data, a low resistive path to ground is created through which the precharged capacitor leaks charges. The voltage difference across the capacitor sensed by the sense amplifier installed at the end of each CAM row eventually determines a match from a mismatch. Only rows that have matched invoke the tag to store logic-'1'.

2. Write: During this phase, key values (according to the LUT) are written into tagged CAM words. Some operations performed on the AP do ”in-place” writing while others do ”out-of-place” writing. In-place writing means that the CAM cells of matched rows that are used to store the data during the compare phase are overwritten (by bits dictated by the LUT) in the write phase. Out-of-place writing, on the other hand, indicates that the bits dictated by the LUT will be written in the matched rows, but in cells dedicated for ”results” only rather than the cells that were used for comparison purposes.

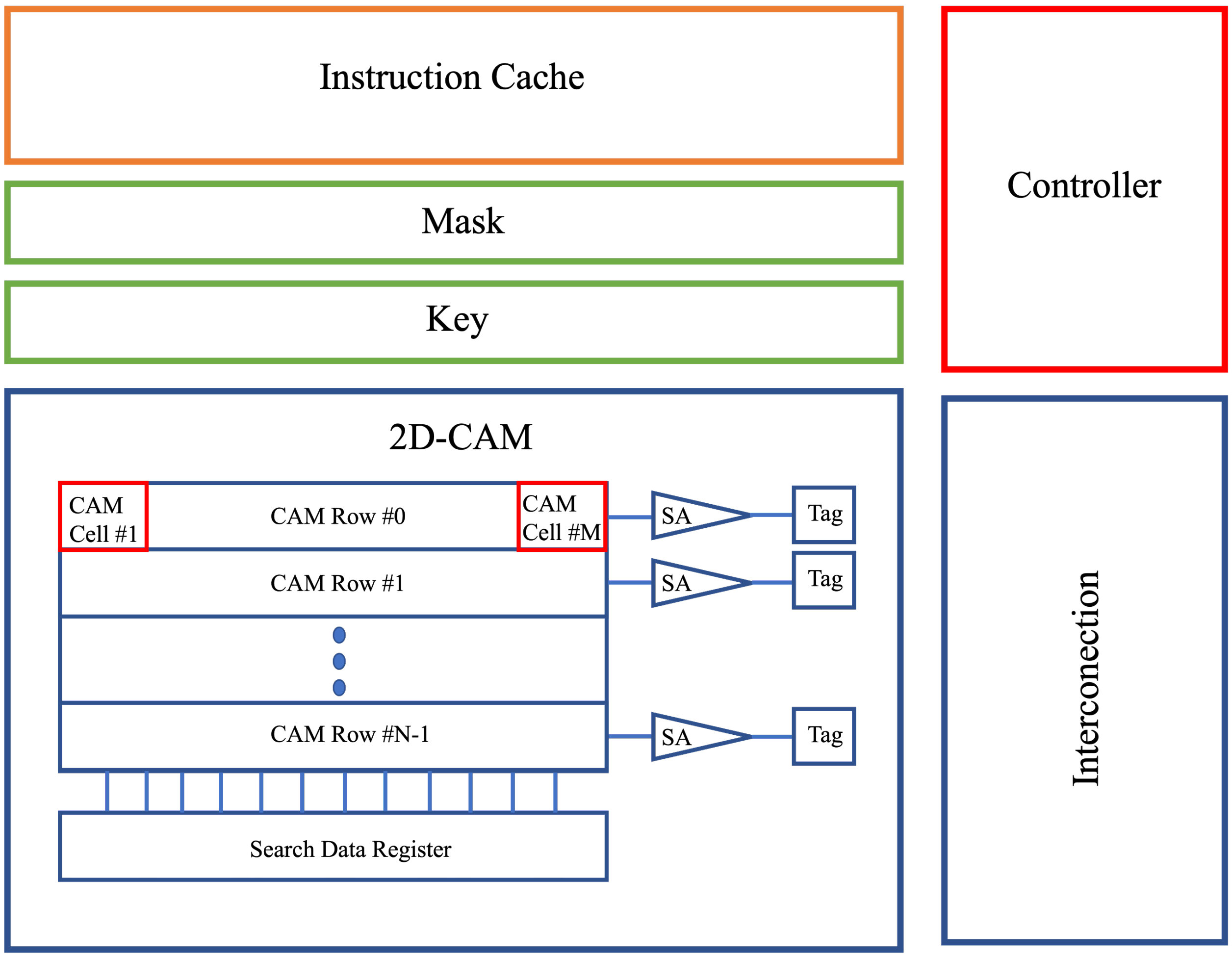


Figure 2.4: Associative Processor Architecture.

We will next elaborate on the supported arithmetic and logical operations that can be performed on the AP.

2.3.2 Supported Operations

The AP can implement arithmetic and logic operations. Assuming that A and B are n-bit vectors, the supported arithmetic operations are in-place (IP) addition ($B[i] <= A[i] + B[i], i \in \{1, ..., n\}$), out-of-place (OOP) ($R[i] <= A[i] + B[i], i \in \{1, ..., n\}$) addition, in-place subtraction ($B[i] <= B[i] - A[i], i \in \{1, ..., n\}$), out-of-place subtraction, two's complement, multiplication ($B[i] <= B[i] \times A[i], i \in \{0, ..., 3\}$), and absolute value. As for the logic operations, the following are supported: and, or, xor and not.

These operations are implemented by relying on their respective LUTs. Example LUTs for addition, subtraction, xor, and or operations are presented in Fig. 2.5. Note that *Cr*, *Br*, and R represent the carry, borrow and output (in OOP operations) respectively. To perform any operation, the AP first compares the $i = 1$ input bits (across all rows; i.e. across all vector entries) to the 1st pass in the LUT (in the compare cycle), and in case of a match, some results could be written either to one of the cells storing input bits or to a separate cell that stores the output according to the LUT (write cycle). The AP repeats this process for all LUT passes and for all remaining $i = \{2, ...n\}$ bits of the input vectors. More detailed explanations and examples can be found in [18].

			In-place			Out-of-place		
Cr	**B**	**A**	**Cr**	**B**	**Comment**	**Cr**	**R**	**Comment**
0	0	0	0	0	**NC**	0	0	**NC**
0	0	1	0	1	**2nd Pass**	0	1	**1st Pass**
0	1	0	0	1	**NC**	0	1	**2nd Pass**
0	1	1	1	0	**1st Pass**	1	0	**5th Pass**
1	0	0	0	1	**3rd Pass**	0	1	**3rd Pass**
1	0	1	1	0	**NC**	1	0	**NC**
1	1	0	1	0	**4th Pass**	1	0	**NC**
1	1	1	1	1	**NC**	1	1	**4th Pass**

(a)

			In-place			Out-of-place		
Br	**B**	**A**	**Br**	**B**	**Comment**	**Br**	**R**	**Comment**
0	0	0	0	0	**NC**	0	0	**NC**
0	0	1	1	1	**1st Pass**	1	1	**1st Pass**
0	1	0	0	1	**NC**	0	1	**2nd Pass**
0	1	1	0	0	**2nd Pass**	0	0	**NC**
1	0	0	1	1	**4th Pass**	1	1	**3rd Pass**
1	0	1	1	0	**NC**	1	0	**NC**
1	1	0	0	0	**3rd Pass**	0	0	**4th Pass**
1	1	1	1	1	**NC**	1	1	**5th Pass**

(b)

B	**A**	**R**	**Comment**
0	0	0	**NC**
0	1	1	**1st Pass**
1	0	1	**2nd Pass**
1	1	0	**NC**

(c)

B	**A**	**R**	**Comment**
0	0	0	**NC**
0	1	1	**1st Pass**
1	0	1	**2nd Pass**
1	1	1	**3rd Pass**

(d)

Figure 2.5: LUTs of (a) addition (in-place and out-of-place), (b) subtraction (in-place and out-of-place), (c) xor, and (d) or operations.

Chapter 3

Decision Trees

Machine learning continues to play a crucial role in performing complex tasks that are characterized by ”learnable” properties. While brain-inspired Deep Neural Networks (DNN) 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 [44–49]. 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 [50–53]. In particular, Decision trees, DTs, are popular to perform explainable ML [7,54], this is known as DT-based ML. DTs are supervised ML models that can be utilized for classification and regression tasks [55]. Classification DTs (an example DT is depicted in Fig. 3.1) combine a group of simple tests sequentially, where each test juxtaposes a numeric (nominal) attribute and a threshold value (set of candidate values) [56]. Attributes are also known as features. As such, classification DTs formulate a set of decision rules by relying on a labeled training dataset. After being constructed, the classification DTs are used to classify unseen data points from the test dataset [8]. In the next section we highlight popular DT training algorithms used for classification/regression.

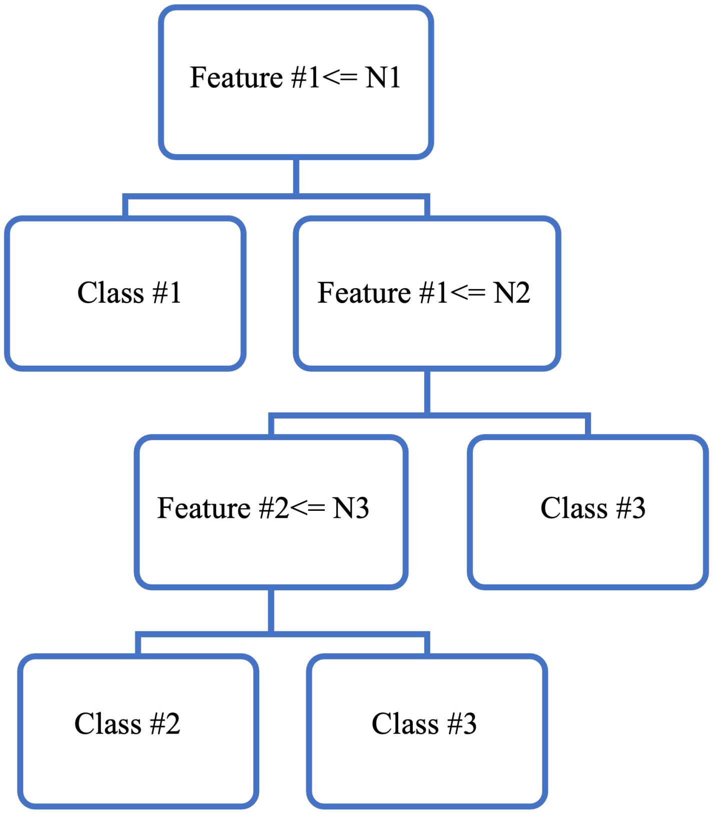


Figure 3.1: A classification DT example. Internal nodes represent the decision rules. Each decision rule has an attribute (feature) and a threshold value. Leaf nodes represent classes

3.1 Training Algorithms

Decision trees can predict very accurately, given the fact that they are trained on high quality data [8]. We discuss hereon some popular DT training algorithms used for classification. A DT algorithm can have either a serial or a parallel implementation depending on the data size, available memory resources, and the scalability of the algorithm itself [57]. We summarize the following DT algorithms in the below subsections: ID3, C4.5, C5.0 and CART.

3.1.1 ID3

Also known as Iterative Dichotomiser 3, the ID3 algorithm is a simple DT algorithm proposed by Quinlan Ross back in 1986. ID3 is implemented in a serial fashion based on Hunt's algorithm, whereby the DT is constructed by relying on a greedy search. In particular, the search is conducted on the training dataset samples in order to test each attribute (categorical) at each tree node. The splitting attribute is chosen based on the information gain measure. Data is usually preprocessed before building the DT model with ID3 since this algorithm is not robust against noise in the dataset [57].

3.1.2 C4.5

Developed by Quilan Ross in the 1990s, C4.5 is a successor of ID3. Similar to ID3, it is serially implemented based on Hunt's algorithm. C4.5 reduces the error rate since it replaces internal nodes with leaf nodes. In addition to categorical attributes, this algorithm also supports continuous attributes. The splitting attribute is chosen based on the gain ratio impurity method [57, 58].

3.1.3 C5.0

C5.0 is also developed by Quinlan Ross, and it is a successor of C4.5. C5.0 utilizes less memory, and it constructs smaller sets of rules compared to C4.5. Moreover, C5.0 is more accurate than C4.5 [59].

3.1.4 CART

Short for Classification and Regression Trees, CART is developed by Breiman in 1984. It is based on Hunt's algorithm, and it is capable of constructing trees for purposes of classification and regression. To construct a classification DT, CART relies on binary splitting of attributes. The splitting attribute is chosen based on the gini index splitting measure [57,60]. It is worth noting that scikit-learn (the Python library we rely on for extracting the DT in our proposed DT2CAM in chapter 5) relies on an optimized version of the CART algorithm [59].

3.2 Hardware Accelerators

3.2.1 ML-based Hardware Accelerators

The need for hardware accelerators, also known as special purpose engines [61], is increasing with the surge of ML applications deployed with strict hardware constraints and the end of Moore's law, Denard's scaling law, and Koomey's law [62, 63]. Hardware accelerator designs trade-off performance and flexibility [64]. Several works in literature have explored ML processors and accelerators for accelerating training, inference, or both [65, 66]. More recently, many ML hardware accelerators rely on neuromorphic architectures, memory-based analog acceleration, and/or computing with light [64]. We summarize some hardware accelerators below.

1. FPGA-based accelerators: Bittware/Achronix VectorPath S7t-VG6 accelerator is an ML accelerator that relies on int8 Multiply-Accumulate Units (MAC) [64, 67]. Microsoft Brainwave project is an Intel Stratix 10 280 FPGA that is implemented as part of the Catapult project. It is used for accelerating inference [64, 68, 69].

2. GPU-based accelerators: The Volta architecture V100 and the Ampere architecture A100 are two NVIDIA computation cards that can accelerate inference and training [64, 70–72]. MI8 and MI60 GPUs are two AMD/ATI computation cards that support acceleration for both inference and training workloads [64, 73, 74].

3. Neuromorphic-based accelerators: Programmable Ultra-efficient Memristor-based Accelerator (PUMA) is a simulator that enables acceleration of a wide variety of ML inference workloads, and it is based on in-memory computing and analog circuitry [75]. Brainchip's Akida Spiking Neural Network (SNN) processor (to be released) is a ”single hardware platform that can perform as an inference engine for the Convolutional Neural Networks (CNNs) of today and support SNNs of tomorrow with its unique on-chip learning algorithms” [76]. It supports 1024 neurons per chip like the IBM TrueNorth research chip [64].

For more details about the different state-of-the-art hardware accelerators in literature, we invite the readers to refer to [64].

3.2.2 DT-based Hardware Accelerators

Several hardware accelerators for DT-based ML are proposed in [22,77–81]. Most of these are CPU (e.g., Intel X5560), GPU (e.g., Nvidia Tesla M2050), FPGA (e.g., Xilinx Virtex-6), or ASIC-based accelerators. More recently, hardware accelerators based on emerging memories like IMC architectures have been proposed for DT-based ML [22, 82, 83].

Chapter 4

MTCAM Designs and Mathematical Formulations

A CAM cell can be implemented in a variety of forms. One popular implementation consists of two transistors and two memristors, which is referred to as 2T-2R [36]. A Memristor-based TCAM (MTCAM) row consists of several of such cells, and each row is equipped with a sensing circuit to distinguish between the full match and the mismatch states as illustrated in Fig. 4.1. The MTCAM row either evaluates to a full match state (fm) where all of its cells are matched with the input bits or to a mismatch state otherwise. Hereon, we refer to the full mismatch/one mismatch state (fmm/1mm) as the state where all/one cell(s) are/is mismatched.

4.1 Capacitive Sensing MTCAMs

In the capacitive sensing scheme, a capacitor is used to distinguish between the fm, and the different mismatch states. During the precharge phase, the capacitor C is charged, and

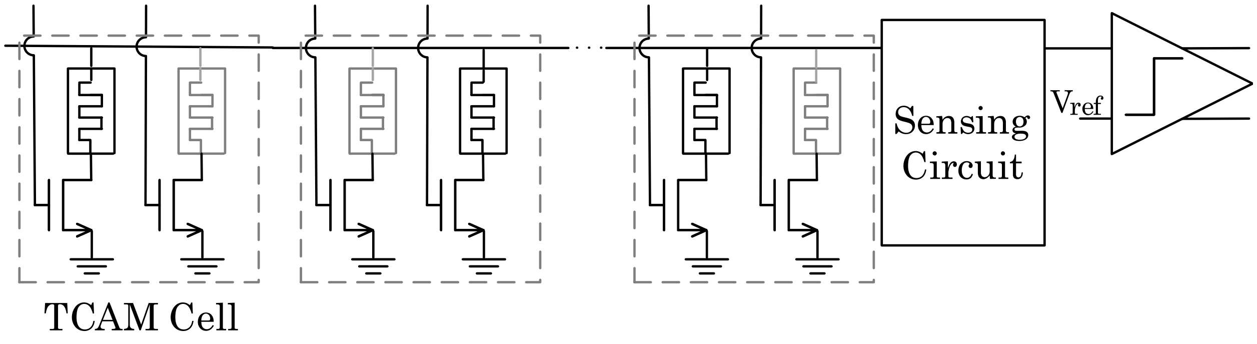
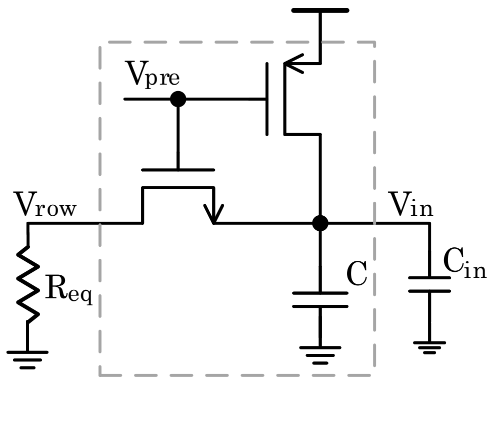


Figure 4.1: 2T-2R TCAM structure.

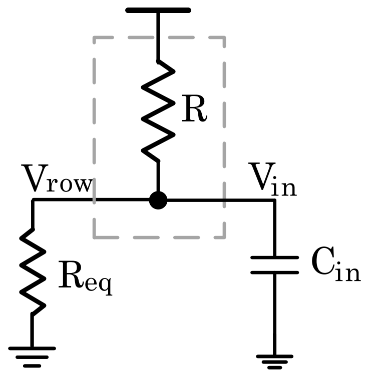
then in the evaluation phase, it discharges through a resistor equivalent to the effective resistance of the MTCAM row of cells shown in Fig. 4.2a. The voltage across C_{in} is used to determine evaluation into a fm or mismatch. In the case of fm, the capacitor discharges slowly, and in case of fmm, it discharges quickly to ground. Figs. 4.3a and 4.3b (in blue) show the simulated voltage and current waveforms across the capacitor of the capacitive sensing MTCAM row. Three cycles are presented corresponding to the fmm, fm, and 1mm states. Each cycle consists of a precharge phase and an evaluation phase.

4.2 Resistive Sensing MTCAMs

Another sensing approach uses a resistor as a voltage divider for the 2T-2R CAM, as shown in Fig. 4.2b. Hence, the voltage that distinguishes the mismatch states from the fm state is depicted as a voltage divider across the equivalent resistor of the MTCAM cells. In this design, there is no need for a precharge phase, and only an evaluation phase is necessary to assess among the different states. Fig. 4.2b shows the MTCAM row based on this design during the evaluation phase. Figs. 4.3c and 4.3d (in blue) demonstrate the output voltage and current of the transient analysis carried out on the resistive sensing MTCAM row. Three cycles are considered: fmm, fm, and 1mm. In this design, each cycle consists of an evaluation phase only. Clearly, this design evaluates much faster while maintaining a good voltage drop difference between the fm and 1mm states. For more information about the setup of the transient analysis, we refer the reader to chapter 3.



(a)



(b)

Figure 4.2: Sensing techniques with the equivalent circuits of MTCAM and comparator, represented in R_{eq} and C_{in}, respectively; a) capacitive sensing circuit and b) resistive sensing circuit, shown in the dashed box.

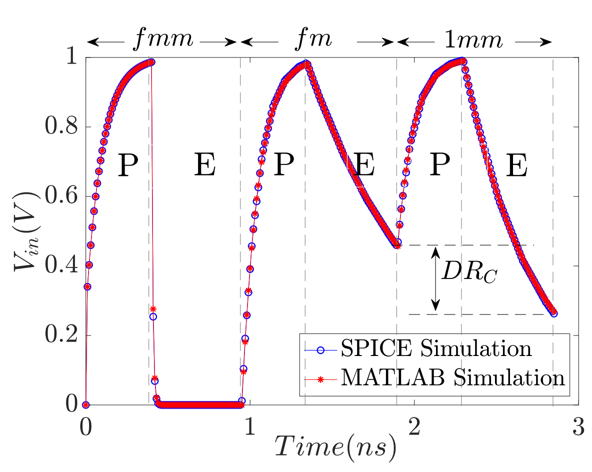
4.3 Mathematical Formulations

4.3.1 MTCAM Row Modeling

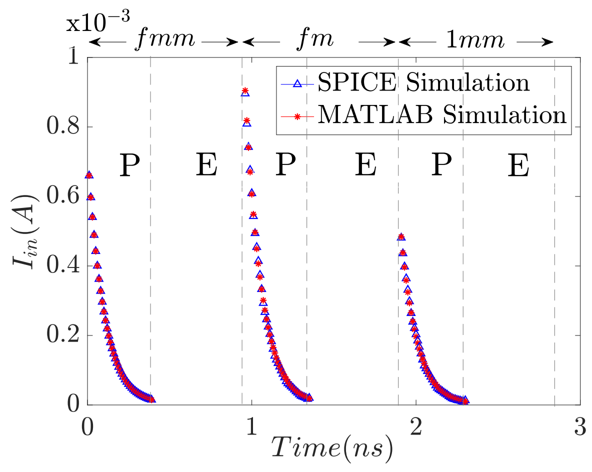
We define the equivalent resistance of a row of MTCAM cells as:

$$R_{eq}(N_{mm}) = R_{mm}//R_m//R_x \tag{4.1}$$

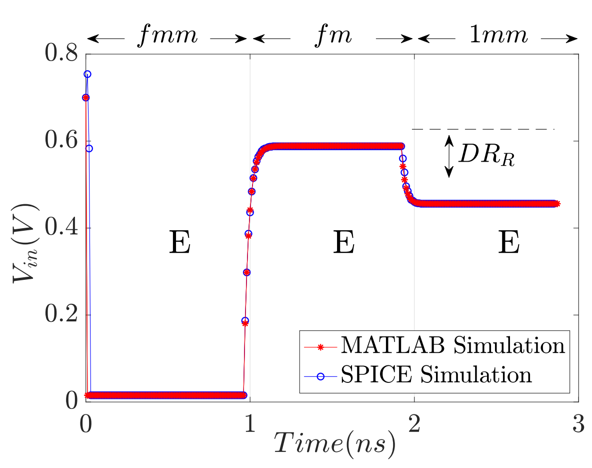
where N_{mm} represents the number of mismatched cells. R_{mm}, R_m and R_x represent the equivalent resistances of the mismatched, matched and 'don't care' cells within the row, respectively, and they are defined as follows.



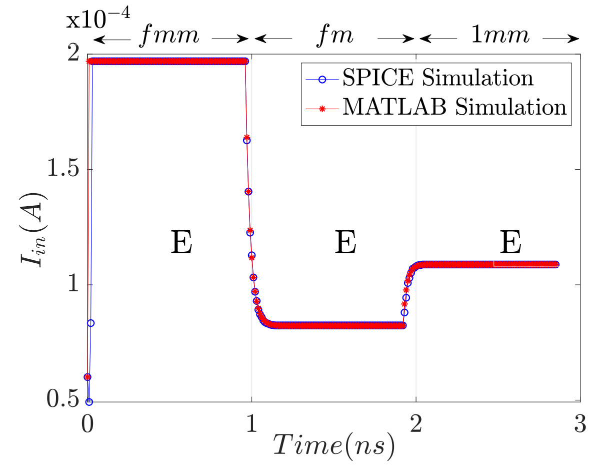
(a)



(b)



(c)



(d)

Figure 4.3: SPICE/MATLAB validation of a TCAM row voltage and current for the fmm, fm, and 1mm states. For the capacitive sensing waveforms (a) and (b), P and E represent the precharge and evaluate phases, respectively. Resistive sensing waveforms, noticeably transitioning faster, are presented over the same time scale for illustration purposes in (c) and (d).

$$R_{mm} = \frac{(R_{on} + R_{LRS})(R_{off} + R_{HRS})}{N_{mm}(R_{on} + R_{off} + R_{LRS} + R_{HRS})} \tag{4.2a}$$

$$R_{m} = \frac{(R_{on} + R_{HRS})(R_{off} + R_{LRS})}{N_{m}(R_{on} + R_{off} + R_{LRS} + R_{HRS})} \tag{4.2b}$$

$$R_{x} = \frac{(R_{on} + R_{HRS})(R_{off} + R_{HRS})}{N_{x}R_{HRS} + N_{x0}R_{off} + N_{x1}R_{on}} \tag{4.2c}$$

where N_{mm}, N_m and N_x are the numbers of mismatched, matched and 'don't care' cells, respectively. The total number of cells includes the number of matches, mismatches and 'don't cares'. N_{x0} and N_{x1} are the numbers of transistors that are OFF and ON of the don't care cells respectively. For computing applications, such as the adder application considered in this paper, N_x is usually set to zero. R_{LRS} represents state '1', R_{HRS} represents state '0'. Moreover, R_{on} and R_{off} represent the effective resistance of the transistors depicted in Fig. 4.1 as indicated in [34].

For the computing applications, such as the adder, which we considered in this paper, we assume no 'don't care' scenarios.

4.3.2 Figure of Merit

In order to have a fair comparison between the two sensing techniques, we define a Figure of Merit (FOM), which is a function of the following three important metrics.

- Sensing Dynamic Range, DR, which is the voltage range between the match voltage and the closest mismatch voltage (i.e., one mismatch state). It is defined as follows.

$$DR(t) = V_{fm}(t) - V_{1mm}(t) \tag{4.3}$$

- Latency, T_L, which is the time needed to distinguish between the fm and mismatch states to establish a maximum or desired dynamic range DR. This time determines the maximum operating frequency of the MTCAM.
- Energy Consumption, E, which is the energy consumed during the search operation by the MTCAM row.

Thus, the Figure of Merit (FOM) is defined as follows.

$$FOM = \frac{DR(T_L)}{T_L * E} \tag{4.4}$$

As one can inspect, the design with the higher FOM presents itself as the better design as it would be more efficient in terms of energy and time, and it would have a relatively high voltage difference suitable for sensing purposes. Generally, the energy can be defined for the circuits in Fig. 4.2 based on Eqns. (4.5) and (4.6).

$$E = \int_{t^i}^{t^i+\Delta t} \frac{V_{dd} * (V_{dd} - V_{in})}{R_0} dt \tag{4.5}$$

$$V_{in}(t) = V_f + (V_{in}(t^i) - V_f) \exp(\frac{-(t - t^i)}{\tau}) \tag{4.6}$$

Assuming a time shift where $t > t^i$, V_f is the final voltage, τ is the corresponding time constant of the RC circuit, and R_0 is the pull-up resistance as defined in Table 4.1. Substituting V_{in} and integrating the previous equation yields the following.

$$E = \frac{\tau V_{dd}}{R_0} \left((V_{dd} - V_f) \frac{\Delta t}{\tau} + \left(V_{in}(t^i) - V_f\right) \left(e^{\frac{-\Delta t}{\tau}} - 1\right) \right) \tag{4.7}$$

For the precharge phase, Δt is selected to be $n\tau$. For purposes of our simulations, we set $n = 3$ throughout the paper, which gives 95% of the steady-state value.

4.3.3 Design Specific Considerations

Three metrics were used for evaluating the performance of the two designs: T_L, Energy, and DR. We herein formulate the corresponding design specific parameters.

Capacitive Sensing Design

(a) Latency: For this design, the optimal evaluation time is derived as $TE_C = (C \ln\left(\frac{R_{fm}}{R_{1mm}}\right) \times \frac{R_{fm} * R_{1mm}}{R_{fm} - R_{1mm}})$ with $R_{fm} = R_{eq}(N_{mm} = 0)$ and $R_{1mm} = R_{eq}(N_{mm} = 1)$ [34]. As such, the latency, T_{CL} can be determined by the worst-case precharge time and TE_C as follows.

$$T_{CL} = n * \tau_{CP} + TE_C \tag{4.8}$$

τ_{CP} is the precharge time constant defined in Table 4.1.

(b) Energy: During the i^{th} cycle, the dissipated energy is the sum of energies dissipated during the precharge and evaluate phases: $E_C^i = E_{CE}^i + E_{CP}^i$. The energy for each phase is derived according to (4.5) while relying on the parameters listed in Table 4.1. Note that the initial conditions for the precharge energy calculations in the i^{th} cycle are obtained from the outcomes of the evaluate phase of the $(i-1)^{th}$ cycle.

(c) Sensing Dynamic Range: The maximum voltage difference between the fm and 1mm states measured at the optimal time TE_C can be defined as [34]:

$$DR_C(TE_C) = V_{dd} * \theta^{\left(\frac{\theta}{1-\theta}\right)} * (1-\theta) \tag{4.9}$$

Table 4.1: Summary of the model parameters for the capacitive and resistive sensing circuits.

	Capacitive Sensing		Resistive Sensing
	Precharge Phase	Evaluation Phase	
τ	$\tau_{CP} = R_{on} * (C + C_{in})$	$\tau_{CE} = (R_{off} \| \| R_{eq}) * (C + C_{in})$	$\tau_R = (R \| \| R_{eq}) * C_{in}$
R_0	R_{on}	R_{off}	R
$V_{in}(t^i)$	$V_{in}(t^{i-1} + TE_{CL})$	V_{dd}	$V_{in}(t^{i-1} + TE_{RL})$
V_f	V_{dd}	0	$V_{dd} \frac{R_{eq}(N_{mm})}{R_{eq}(N_{mm}) + R}$
Δt	$n\tau_{CP}$	TE_{CL}	TE_{RL}

with $\theta = R_{1mm}/R_{fm}$.

Resistive Sensing Design

(a) Latency: This design is evaluate-based only. The latency, TE_{RL} is determined by the worst-case evaluate time as

$$TE_{RL} = n * \tau_{WC,R} \tag{4.10}$$

$\tau_{WC,R}$ is the worst case evaluate time constant for the fm state derived from τ_R in Table 4.1 with $R_{eq}(N_{mm} = 0)$.

(b) Energy: During the i^{th} cycle, the dissipated energy, E_R^i, is equal to the evaluate energy, E_{RE}^i, derived according to (4.5) while relying on the parameters listed in Table 4.1 for the resistive scheme. Specifically, for the i^{th} cycle, $V_{in}(t^i)$ is obtained from the outcomes of the previous cycle, and we expect $V_{in}(t^i + TE_{R_L})$ to reach the final voltage value, V_f, derived according to the voltage divider equation based on the corresponding cycle's N_{mm}.

(c) Sensing Dynamic Range: It can be derived from (4.3) as

$$DR_R(TE_{RL}) = V_{dd}\frac{R(R_{fm} - R_{1mm})}{(R + R_{fm})(R + R_{1mm})} \tag{4.11}$$

The maximum dynamic range can be achieved by taking the first derivative with respect to R and equating it to zero. The optimal R value is as follows.

$$R^* = \sqrt{R_{1mm}R_{fm}} \tag{4.12}$$

It is worth noting that R^* is optimized to maximize the dynamic range not to maximize FOM, which is a monotonically increasing function of R.

4.4 Analysis and Results

4.4.1 Transient Simulations

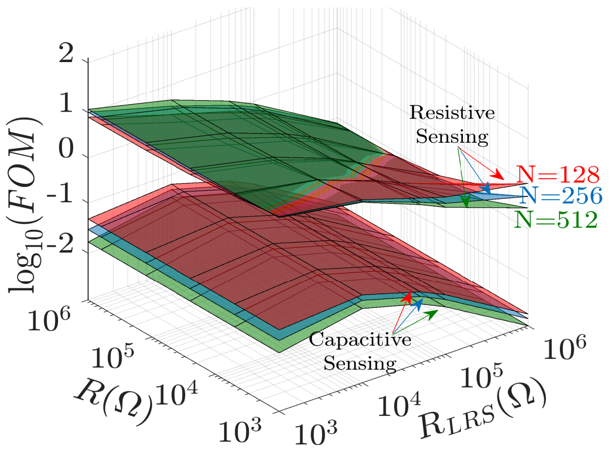
Fig. 4.3 presents precharge and evaluate waveforms for the fm, fmm, and 1mm states for the following parameter combinations: $N = 128$ (number of cells per row), (R_{LRS}, R_{HRS}) = ($1k\Omega$, $\alpha * R_{LRS}$) where $\alpha = 1000$ (representing low and high memristance values), (R_{on}, R_{off}) = ($9k\Omega$, $10M\Omega$) to model the precharge transistor in Fig. 4.2a, and $R = 5k\Omega$. $C_{in} = 10fF$ and $C + C_{in} = 100fF$. For the capacitive scheme, a load of at least $100fF$ is needed for the comparator to properly latch V_{in} and distinguish between the fm and 1mm states due to fast discharge. The simulated SPICE voltage and current waveforms (in blue) are concurrent with the corresponding MATLAB simulated waveforms (in red) (based on (4.6)). Hence, the theoretical results that will be presented in the next section in terms of FOM and design metrics are consistent with the SPICE simulations.

As apparent in Fig. 4.3, the time taken by the resistive sensing design to evaluate all three states is in the order of picoseconds compared to nanoseconds for the capacitive sensing. Moreover, the corresponding energy needed for the resistive sensing design to evaluate during that time is obviously less because of the absence of the need to fully precharge which is required for the capacitive design. Note that the resistive scheme waveforms are presented over the same timescale as the capacitive one for illustration purposes only. While the dynamic range is larger in the capacitive sensing model, the resistive sensing scheme maintains a good dynamic range. This proves that the resistive sensing design has an edge when energy and latency are key concerns. The resistive sensing design is therefore ideal for inference applications where there are consecutive evaluations with no hold intervals. Particularly, the memristive array can be powered off when inference is completed, and hence no static energy will be dissipated in the resistive sensing approach.

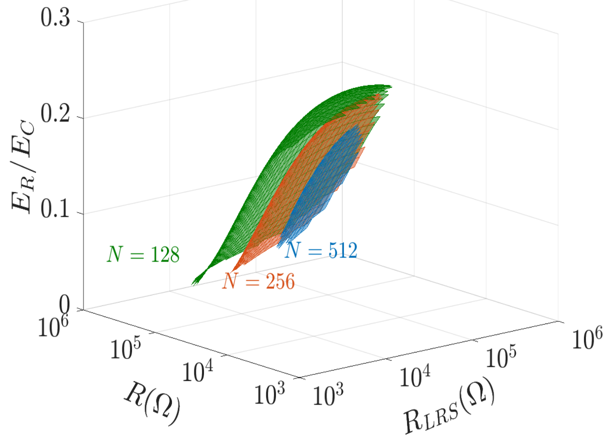
4.4.2 Design Space Exploration: Figure of Merit Analysis

We rely on MATLAB numerical simulations to perform comparative FOM analysis for the two designs. We rely on the following design space parameter combinations to consider different memristor device values for our study [84]: $N \in \{128, 256, 512\}$, $R_{LRS} \in \{1K, 10K, 50K, 100K, 1M\}\Omega$, $R \in \{1K, 5K, 25K, 125K, 625K, 1M\}\Omega$, $R_{on} = 9K\Omega$, $R_{off} = 10M\Omega$, $V_{dd} = 1V$, $C_{in} = 10fF$, $R_{HRS} = \alpha * R_{LRS}$ where $\alpha = 1000$ and $C + C_{in} = 100fF$. We rely on these combinations to compare the two designs over different device types and ranges. In fact, some device LRS can go as low as 100Ω, and α can be as high as 1000 or more [84,85]. While large α values may have implications on the device endurance [85], some metal Oxide RRAMs exhibit good endurance for $\alpha = 1000$ [84]. We also note that these ratios can drop due to variability [1, 85], particularly in HRS, and hence, our assumption for $\alpha = 1000$ represents a median window ratio.

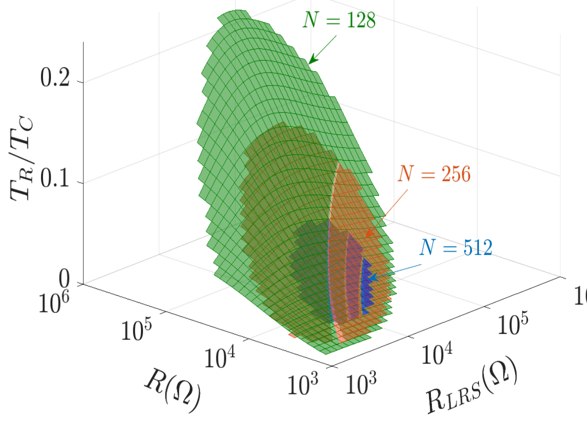
Fig. 4.4a shows 3D plots for FOMs (when considering worst-case energy of the full mismatch case) of the two designs corresponding to the different combinations. One can see that the resistive sensing design shows higher FOM values compared to the capacitive sensing design for all LRS, R, and N values. Furthermore, the energy and latency results of the resistive sensing scheme outperform that of the capacitive sensing one over the design space. Figs. 4.4b, 4.4c and 4.4d present the energy, latency and FOM ratios respectively plotted with the additional constraint on the dynamic range: $DR > 75mV$ to maintain that a sense amplifier can differentiate between the fm and 1mm states. Table 4.2 lists the maximum FOM ratio along with the corresponding energy and latency measures for both designs when $DR > 75mV$. We note upto $260\times$ FOM improvement for the resistive sensing scheme compared to the capacitive sensing scheme for $N = 256$.
While the FOM is a good overall comparator for both designs, some of the metrics may be more significant than others depending on the specific application. For example, for error-tolerant neuromorphic computing applications, energy matters most. For AP applications,



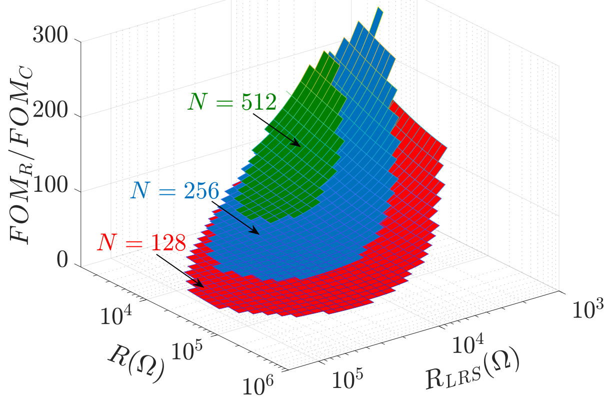
(a)



(b)



(c)



(d)

Figure 4.4: (a) FOM comparison between the capacitive and resistive sensing techniques for different N values. (b) Energy ratio, (c) evaluation time ratio and (d) FOM ratio for different N values with constrained dynamic range to $DR > 75mV$.

with dense TCAM structures, energy as well as accuracy in terms of the DR matter, and latency could be traded-off in favor of energy and/or DR. For IP routing applications, energy is not a concern. Since the achieved search times for both the capacitive and resistive sensing schemes are fast, one may consider DR as the key metric for IP routing. If utmost search speed is under consideration, both latency and DR can be considered as key metrics for IP routing applications. As such, we present in Table 4.3, a summary of the preferred sensing scheme based on the performance of the desired metrics for the specific applications when $DR > 75mV$. It is evident that the resistive sensing scheme is preferred when the energy is a concern.

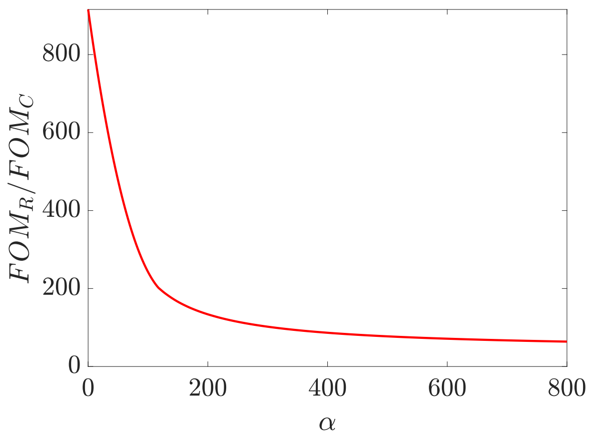
We further swept α values and conducted Monte Carlo simulations to mimic variability in HRS similar to the strong programming values in [1], where we set α~N($\mu = 500$, $\sigma_{lower} = 133$ and $\sigma_{upper} = 83$) where $\alpha = \mu + ((z > 0)?\sigma_{upper} * z : \sigma_{lower} * z)$ and z ~$N(0, 1)$. $R_{LRS} = 5K\Omega$, $R = 10K\Omega$, $N = 128$. The results are illustrated in Fig. 4.5. Fig. 4.5a presents the FOM ratio vs. α (fitted curve plotted). We note that FOM ratio increases as α decreases, with some advantage in terms of the DR for the capacitive sensing scheme as illustrated in Fig. 4.5b (DR vs. α fitted curve plotted). Hence the capacitive sensing scheme sustains a $DR > 75mV$, needed for accurate sensing, at lower alpha values compared to resistive sensing. Fig. 4.5c presents DR_R and DR_C based on random variations in HRS similar to [1]. The two schemes have similar lower tails for their respective DR distributions. As such, in the presence of process variations in HRS, the two schemes are mostly suited for error tolerant applications, with the resistive scheme offering energy/latency savings compared to the capacitive one.

Table 4.2: Energy and latency of points with the highest FOM Ratio with $DR > 75mV$ for each N.

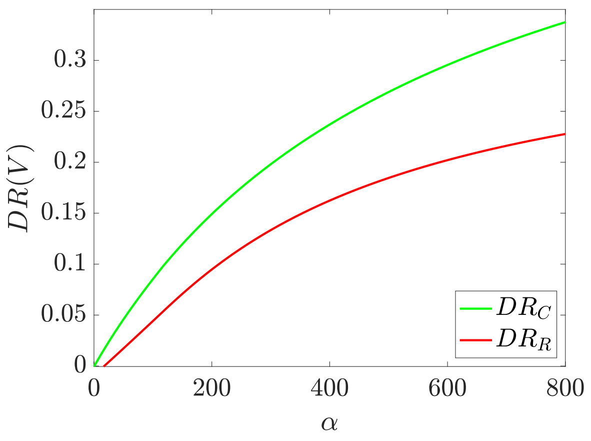
N	Max FOM Ratio	E_R(fJ)	E_C(fJ)	TE_{RL}(ns)	T_{CL}(ns)
128	177.9	24	99	3.1E-2	3.2
256	259.7	17	99	4.7E-2	3.0
512	222.1	18	99	5.1E-2	3.1

Table 4.3: Summary of the preferred sensing scheme based on the performance of the desired metrics for the specific applications for $DR > 75mV$. R:C represents the ratio of the key metric for the resistive sensing scheme to the capacitive sensing scheme

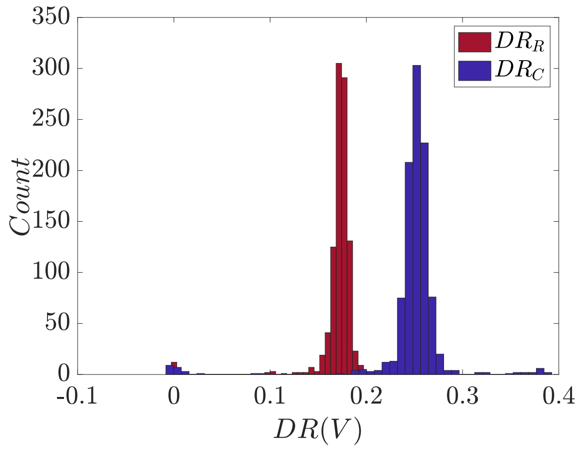
Application	Key Metrics	R:C			Preferred Sensing Scheme
		N=128	N=256	N=512	
Associative Processors	DR/Energy	28.7	14.0	8.4	Resistive
Neuromorphic Computing	1/Energy	41.0	20.0	12.0	Resistive
IP Routing	DR	0.7	0.7	0.7	Capacitive
IP Routing (Fast Search)	DR/Latency	73.4	61.5	46.9	Resistive



(a)



(b)



(c)

Figure 4.5: (a) FOM ratio vs. α (fitted curve plotted), (b) DR vs. α: swept alpha and applied same alpha to all memristors in the TCAM row (fitted curve plotted), and (c) DR_R and DR_C based on random variations in HRS similar to [1].

4.5 In-memory Adder Example

We studied the two design alternatives in the context of n-bit adder application in a $2n \times (2n+1)$-bit TCAM array for $n \in \{16, 32, 64\}$. We implemented the LUT-based approach in [86]. At its core, the algorithm relies on four passes of 3-bit comparisons each as indicated in Table 4.4, operates on the different rows in parallel and consecutively computes addition from the least significant bit (LSB) to the most significant bit (MSB). Since the procedure overwrites one of the inputs, it is necessary to follow the order of the passes as specified in

Table 4.4: Addition look-up table.

Compare			Write		
Cr	B	A	Cr	B	Comments
0	0	0	0	0	NC
0	0	1	0	1	2nd Pass
0	1	0	0	1	NC
0	1	1	1	0	1st Pass
1	0	0	0	1	3rd Pass
1	0	1	1	0	NC
1	1	0	1	0	4th Pass
1	1	1	1	1	NC

Algorithm 1 Addition B=B+A per row of AP

```
0: procedure ADD($A_{bit}, B_{bit}, Carry_{bit}, Previous_{state}, N_{bit}$) {Returns $New_{Carrybit}, Next_{State}, Total_{Energy}$}
1: for $i = 1$ to $n_{bits}$ do
2:    for $pass = 1$ to 4 do
3:       Determine next state
4:       Calculate $E_{s1->s2}$ for resistive sensing or $E_s$ for the capacitive sensing
5:    end for
6: end for
```

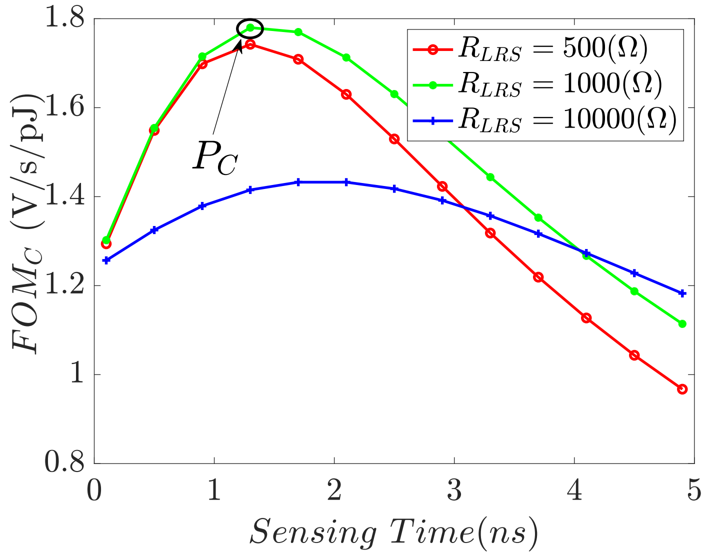
Table 4.4. The adder function pseudo-code is presented in Algorithm 1. We explored the design space to identify and compare the best resistive and capacitive sensing design points.

4.5.1 Design Space Exploration

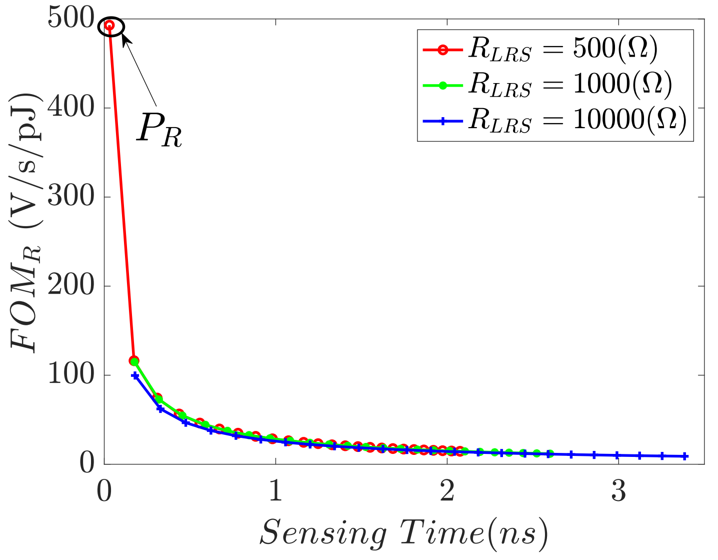
We conducted a design space exploration with a MATLAB setup similar to that in section 4.4.2, but with: (i) $N = 3$, (ii) $R_{LRS} \in \{500, 1K, 10K\}\Omega$. Since the adder's functionality comprises a 3-bit comparison per row, we note that one can achieve a reasonable DR_C, with a time interval that is less than the optimal TE_C. As such, (iii) we swept TE as a parameter for the capacitive sensing design over the range [0.0298-5]ns. This range is obtained based on the resistive sensing evaluate latency corresponding to the different R combinations for a given R_{LRS}. Hence for consistency, we use this TE range as our reference sensing time axis for our study. Note that for the capacitive sensing design, we still need to include the precharge time in the FOM analysis. From Fig. 4.6, it is evident that the resistive sensing design offers enhanced FOM values compared to the capacitive sensing design for the space under study. Despite the similarity in the range of the sensing time TE for both designs, $3\tau_{CP}$ is significant and contributes to the FOM degradation for the capacitive sensing approach. As shown in Fig. 4.6b, we identify the best resistive sensing design point P_R to correspond to (R_{LRS}, TE_{RL}) = (500Ω, 0.02ns (i.e., $R = 1K\Omega$)). Note that while R value in (4.12) provides the best dynamic range, it does not correspond to the best FOM. P_R yielded $DR_R = 100mV$, $TE_{RL} = 0.02ns$, $E_R = 6fJ$ and $FOM_R = 493$. For the capacitive sensing design, the best FOM was achieved at the point P_C corresponding to (R_{LRS}, TE_C) = $(1k\Omega, 1.3ns)$ as shown in Fig. 4.6a. DR_C was around $700mV$, latency $T_{CL} = 4ns$, $E_C = 97fJ$ and $FOM_C = 1.78$ for this design point. This point offered a good trade-off between the different metrics. The FOM ratio at the best operating points is around 277.

4.5.2 Energy Saving

Herein, we focus on the energy savings at the design points P_R and P_C. For the capacitive sensing, we assumed the current evaluate will incur known energy, E_s in the following precharge, where $s \in \{fm, 1mm, 2mm, 3mm\}$. These states correspond to the full match, one, two, and full mismatch energies, respectively. For example, if the LSBs of integers A and B and the carry, $A_0B_0C = 000$, then the 1st pass of the LUT-based algorithm will compare $'000'$ to $'110'$. This will result in an energy dissipation equivalent to E_{2mm} during the following precharge phase. For the resistive sensing design, on the other hand, there is no precharge phase, as indicated earlier, and the evaluation leads to a charge or discharge based on the previous state. For the j^{th} bit, the previous state is saved from either the previous pass or the last pass of $(j-1)^{th}$ bit addition. So, it depends on a number of transitional energies of interest of the form: $E_{s_1-s_2}$, where $s_{1,2} \in \{fm, 1mm, 2mm, 3mm\}$. Hence, if $A_0B_0C = 000$, the 2nd pass will compare $'000'$ against $'100'$ and will dissipate an energy equal to $E_{2mm->1mm}$. In both designs, the energy for a given bit per row is computed based on the energy consumed in the four passes of the lookup table [86]. The total energy represents the sum of energies consumed due to the addition of the n-bit integers across all rows. For a word size of 16, 32, and 64 bits, we ran 1000 add operations. The resulting average



(a)



(b)

Figure 4.6: FOM versus sensing time; a) for $N = 3$ in capacitive sensing MTCAM, b) for $N = 3$ in resistive sensing MTCAM.

energy ratios demonstrated around 14× energy savings for the resistive sensing TCAM when compared to the capacitive sensing TCAM for all n.

4.6 Summary

Up-til now, we presented a comparative study between two sensing designs of 2T-2R TCAMs. Our comparison was carried out theoretically and using SPICE simulations, where the resistive sensing MTCAM proved to have the edge over its capacitive sensing counterpart design based on the adopted figure of merit, which incorporates the dynamic range, latency, and energy as the design performance metrics. This was particularly true for applications where energy and latency are key concerns. In this context, resistive design showed up-to 260× FOM improvements over the capacitive design, while taking DR constraints into consideration. We also note that for $N = 128$, the resistive sensing design, which is characterized by low energy and latency, satisfied the DR constraints over a wide range of the design parameter space. Finally, we presented an adder application that maintained our theoretical and SPICE results. As such, we have demonstrated that the resistive sensing of MTCAM cell serves as an efficient building block of AP designs and their subsequent applications. In the next chapters, we adapt the capacitive sensing scheme due to its popularity with the current TCAM designs. We will explore the proposed DT2CAM framework with resistive sensing in future work.

Chapter 5

Proposed DT2CAM Framework

Our proposed framework comprises two components: A decision tree to CAM-based hardware architecture compiler (DT-HW compiler), and a Resisitve-CAM functional synthesizer (ReCAM functional synthesizer). The DT-HW compiler translates a decision tree graph to a structured lookup table. The ReCAM functional synthesizer first maps the look-up table into ReCAM arrays and then evaluates energy, latency, and accuracy via simulations.

5.1 DT-HW Compiler

To map a decision tree graph into a structured look-up table, the DT-HW compiler comprises four main steps: decision tree graph generation, tree parsing, column reduction, and ternary adaptive encoding step. We next elaborate on each of these steps.

5.1.1 Decision Tree Graph Generation

In this step and for a given dataset, a supervised decision tree model capable of performing multi-class classification is trained by relying on the Classification and Regression Trees (CART) algorithm [87]. The decision tree model can be represented by a decision tree graph. The internal nodes of the graph represent rules on the attributes or features, the branches represent the decisions for the rules, and leaf nodes represent outcome classes.

5.1.2 Tree Parsing

Starting with a decision tree, the DT-HW compiler parses it into its equivalent table of conditions; each row in the table represents a path in the decision tree from root to leaf, and the number of rows is equal to the number of paths of the tree. Subsequently, each row consists of condition(s) applied to at least one feature.

5.1.3 Column Reduction

After the tree parsing step, 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 decision tree enforces a continuous range for the rule definition in a given path (row). The rule can be defined using a three-state comparator ∈ { '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, $f_{in_i} \in (-Inf, Th1_{ij}]$) in order 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 $(Th1_{ij}, Th2_{ij}]$ in order to match the rule.

5.1.4 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. We 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. Hence, it is referred to 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 follows:

$$n_i = T_i + 1 \tag{5.1}$$

Hence, for N features, the total number of bits (n_{total}) that are eventually needed to encode the whole decision tree (excluding the leaf nodes that store the class labels) is as follows.

$$n_{total} = N_{branches} * \sum_i (n_i) \tag{5.2}$$

where $N_{branches} = m$ is the number of branches or paths from the root to leaf nodes in the decision tree (or the number of leaf nodes).

The encoding scheme employs unary codes in the 'normal' form [88]. The encoded bits belong to the basis $\{0, 1, x\}$; x represents a "don't care". This encoding facilitates bijective mapping of the rules into TCAM(s). The encoding can be best explained as follows for a given feature f_i.

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 note that the input features also rely on the same scheme to be encoded, 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 a 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. In order to accommodate for cases where a feature spans multiple exclusive ranges, we rely on "don't care" bits denoted as "x" to encode the new union range. With this scheme, inputs that belong 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 the following steps.

1. Find the set of exclusive ranges, $\{r_{LB}, r_{UB}\}$, spanned by $rule_{ij}$. $LB, UB \in \{1, .., n\}$.

2. Encode $rule_{ij}$ as follows.

$$Idx = Find_{idx}(XOR(u_{r_{LB}}, u_{r_{UB}}) == 1) \tag{5.3}$$

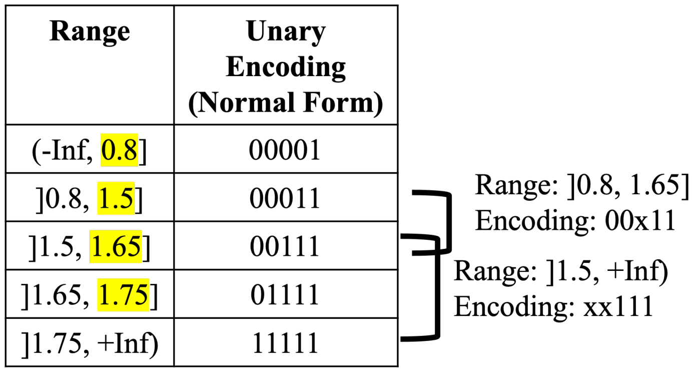
$$u_{rule_{ij}} = Replace(u_{r_{LB}}, Idx, "x") \tag{5.4}$$

   $Find_{idx}(.)$ returns a list of indices satisfying a certain condition. $Replace(u, Idx, "c")$ replaces all the characters of string u in positions Idx by the character "c".

Fig. 5.1 presents an example that illustrates the encoding scheme for some feature f_i. Without loss of generality, we assume that $T_i = 4$ and $Th^{f_i} = \{0.8, 1.5, 1.65, 1.75\}$ as highlighted in yellow in Fig. 5.1. Accordingly, we construct the unary codes, {00001, ..., 11111}, for the five exclusive ranges. We note again that we use five bits to encode each range since there are four unique thresholds. So, if in the column reduction step, $rule_{ij} = "'0', 0.8, NaN"$, i.e., $f_i \leq 0.8$, its range spans the first range $(-Inf, 0.8]$. Accordingly, $u_{rule_{ij}} = 00001$. If $rule_{ij} = "'2', 1.65, 1.75"$, i.e., requiring $f_i \in]1.65, 1.75]$, and hence it will be encoded as $u_{rule_{ij}} = 01111$. To find the encoding of the new range $]0.8, 1.65]$, which spans the second and third ranges (Fig. 5.1), we find XOR(00011, 00111). This results in the string 00100. $u_{rule_{ij}} = Replace(00011, \{'3'\}, 'x') = 00x11$. In a similar manner, a range of $]1.5, +Inf[$, which spans the last three exclusive ranges in the table of Fig. 5.1, is encoded as $xx111$.

5.2 DT-HW Sample Example based on Iris Dataset

Henceforth, we rely on Fig. 5.2 to elaborate on the four steps described above. Starting with a given decision tree, adapted from part of the Iris dataset [89] decision tree, DT-HW parses it into its equivalent table of rules in the tree parsing step. The left most and right most paths in the decision tree graph are parsed into the first and second rows of the table as follows: if "Petal Width", PW, of the input is less than or equal to 0.8, the class output



Range	Unary Encoding (Normal Form)
(-Inf, 0.8]	00001
]0.8, 1.5]	00011
]1.5, 1.65]	00111
]1.65, 1.75]	01111
]1.75, +Inf)	11111

Range:]0.8, 1.65]
Encoding: 00x11

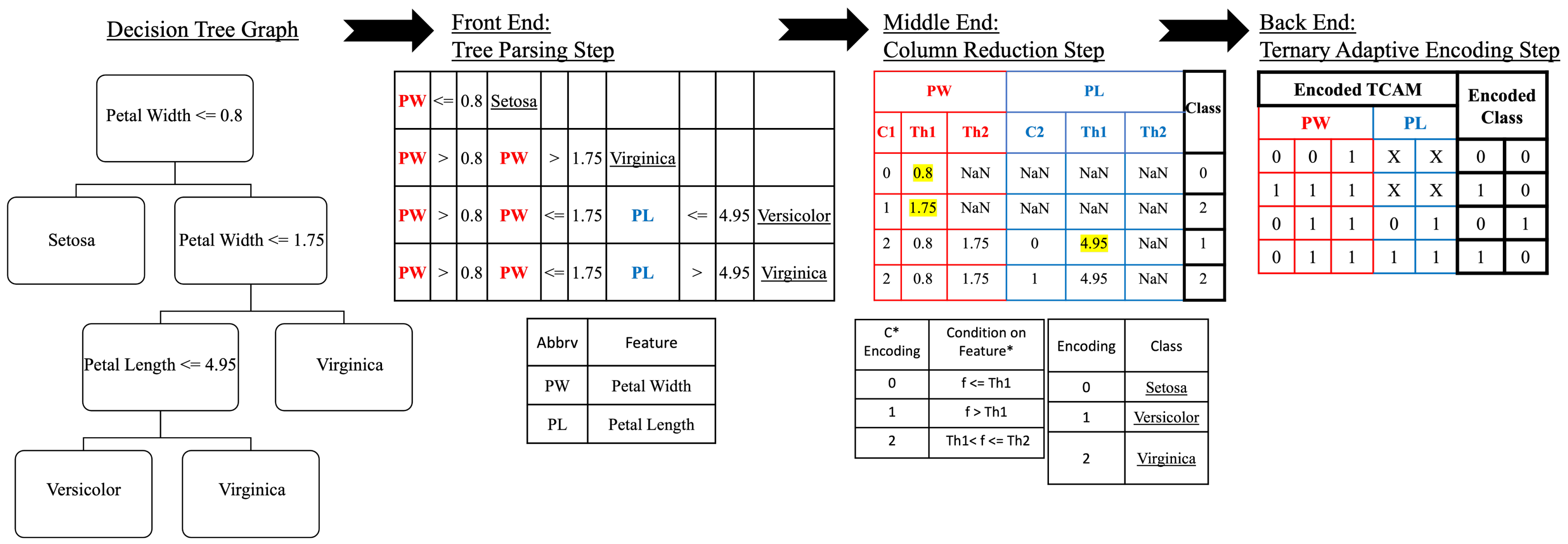
Range:]1.5, +Inf)
Encoding: xx111

Figure 5.1: An example of encoded ranges based on four unique thresholds (highlighted in yellow): 0.8, 1.5, 1.65, and 1.75. Unary codes in normal form are used for exclusive intervals comprising the unique thresholds. For inclusive intervals (union of multiple exclusive intervals), "don't care" bits (denotes as "x") are used to maintain the correctness of the codes of these ranges.

at the leaf is "Setosa" (row1). Otherwise, if the input "Petal Width" is greater than 0.8 and greater than 1.75, then the class at the leaf is "Virginica" (row2).

Then, in the column reduction step, the second row ($PW > 0.8$ and $PW > 1.75$ Virginica) is reduced into one rule on PW, $PW > 1.75$ (i.e. comparator is $'1'$, $Th1 = 1.75$, and $Th2 = NaN$). Moreover, in the column reduction step, each unique class in the original decision tree is assigned a natural number.

By inspecting the columns of PW in the column reduction step, one can notice that PW has two unique thresholds so it should be encoded using three bits in the final ternary adaptive encoding step. The actual encoding follows step3 explained above. The same steps are repeated for all other rows and features. When all rules are encoded, binary encoding is further used to represent the class (decision tree leaf nodes).



PW	<=	0.8	Setosa							
PW	>	0.8	PW	>	1.75	Virginica				
PW	>	0.8	PW	<=	1.75	PL	<=	4.95	Versicolor
PW	>	0.8	PW	<=	1.75	PL	>	4.95	Virginica

Abbrv	Feature
PW	Petal Width
PL	Petal Length

PW			PL			Class
C1	Th1	Th2	C2	Th1	Th2	
0	0.8	NaN	NaN	NaN	NaN	0
1	1.75	NaN	NaN	NaN	NaN	2
2	0.8	1.75	0	4.95	NaN	1
2	0.8	1.75	1	4.95	NaN	2

C* Encoding	Condition on Feature*
0	f <= Th1
1	f > Th1
2	Th1< f <= Th2

Encoding	Class
0	Setosa
1	Versicolor
2	Virginica

Encoded TCAM					Encoded Class	
PW			PL			
0	0	1	X	X	0	0
1	1	1	X	X	1	0
0	1	1	0	1	0	1
0	1	1	1	1	1	0

Figure 5.2: Decision Tree Hardware (DT-HW) Compiler: Translates a decision tree graph to a structured lookup table. In particular (from left to right), it first parses the decision tree and creates a table, then reduces the columns of the table, and finally, it uses a ”ternary adaptive encoding” scheme to create the look-up table.

5.3 ReCAM Functional Synthesizer

The ReCAM functional synthesizer comprises two steps:

- Mapping step: maps the look-up table, provided by the DT-HW compiler, into Ternary Resistive-CAM arrays. It takes into consideration hardware and functional limitations.
- Simulation step: After that, the synthesizer then evaluates energy, latency, and accuracy via simulations while maintaining certain specifications or limitations.

5.3.1 Mapping

A bit of "0", "1", or "x" in the look-up table is mapped to a "01", "10", or "11", respectively, in the two resistive elements of a TCAM cell as shown in Fig. 5.3. 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.

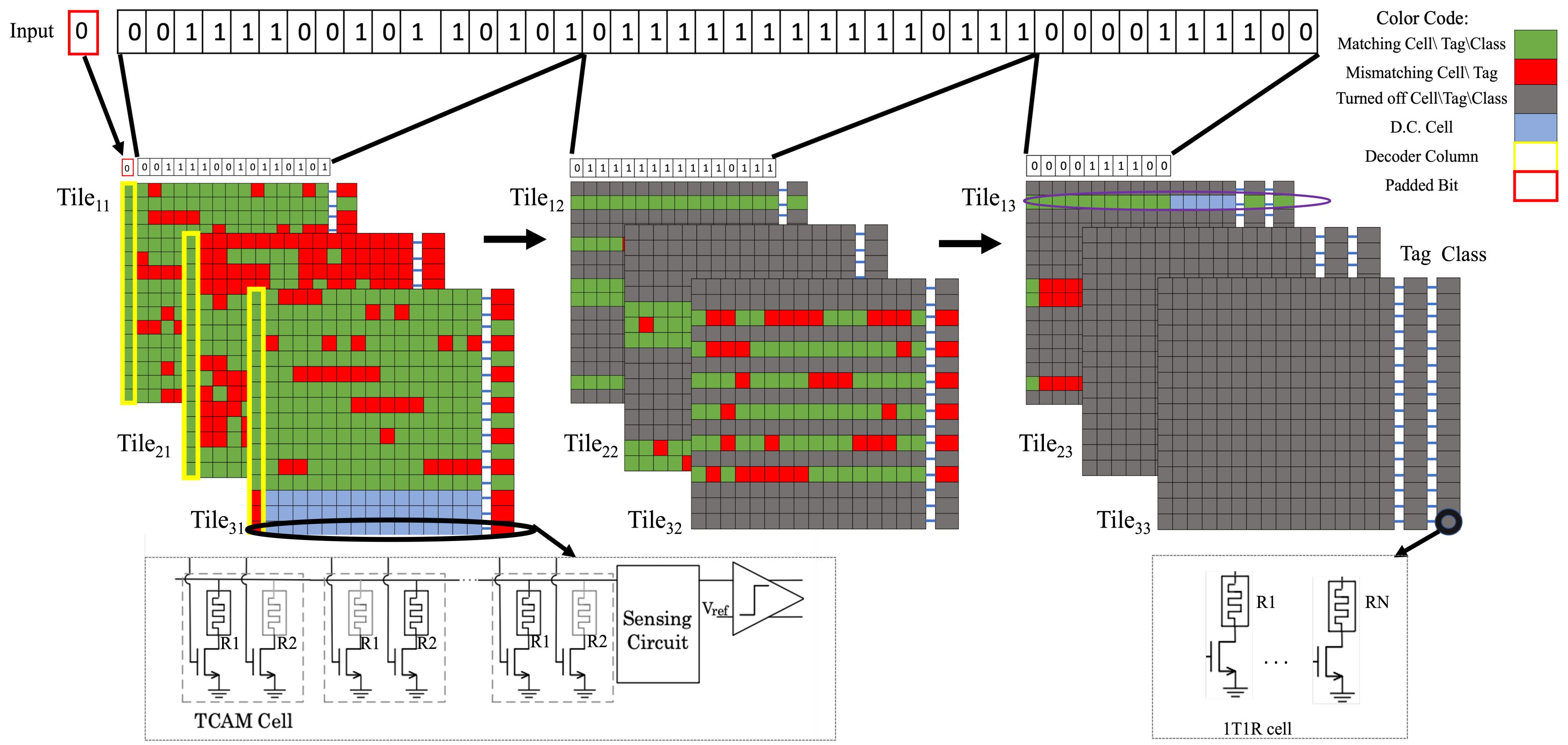


Figure 5.3: ReCAM Functional Synthesizer: Maps the look-up table from the encoding step into ReCAM arrays and runs energy, latency, and accuracy evaluations. In other words, it breaks up, if needed, the table from the encoding step into multiple tables that can be mapped into Resistive TCAMs of regular size "$S \times S$". For purposes of 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.

Dynamic Range: The dynamic range of the TCAM is a limitation that needs to be satisfied to guarantee correct functionality. The dynamic range of a TCAM describes the voltage difference between a full match voltage, V_{fm}, and the one mismatch voltage, V_{1mm}, and it needs to be a "measurable difference" for the Sense Amplifier (SA) to detect it and differentiate between a full matching row (fm) scenario and a one mismatching row one (1mm). The dynamic range, D, is defined as follows.

$$D = V_{fm} - V_{1mm} \tag{5.5}$$

Furthermore, we remind the reader that the dynamic range for a capacitive sensing design, D_{cap}, measured at optimal time T_{opt} is defined as follows [37, 90].

$$D_{cap}(t = T_{opt}) = V_{DD} * \gamma^{(\frac{\gamma}{1-\gamma})} * (1 - \gamma) \tag{5.6}$$

where V_{DD} is the supply voltage, and $\gamma = \frac{R_{1mm}}{R_{fm}}$, with R_{1mm} and R_{fm} being the equivalent resistances of the TCAM row in the cases of one mismatch and full match respectively.

The dynamic range is affected by the TCAM row size (equivalently, the size of the row of the encoded table of Fig. 5.2) which affects the equivalent resistances. So given a certain limit on the dynamic range, D_{limit}, (to render it a "measurable difference"), we calculate relying on Eqn. (5.6) a target TCAM row size S beyond which D_{limit} cannot be met.

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 $N_{cwd} = \lceil (n_{total}/\#rows + \mathbf{1})/S \rceil$ 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 in the scenarios below.

- The original size of the LUT is smaller than $S \times S$ (in that case N_{cwd}=N_{rwd}=1), and the functional synthesizer needs to extend the table obtained from the encoding step by padding "don't care" cells to render the LUT size $S \times S$ (5.2). We reserve the first column of the TCAM array and refer to it as the decoder column to enforce mismatch for the rows that are not part of the original LUT (denoted as rogue rows).

- Otherwise, it needs to divide that table into multiple TCAM tiles of size $S \times S$. 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. 5.3. For purposes of 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. Furthermore, 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 SAs that are 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, that are used to store the class bits (or equivalently the encoded leaf nodes' values of the decision tree). 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 decision tree that has C, possible classes, $\lceil log_2(C) \rceil$ bits or 1T1R cells are needed for each row.

Without loss of generality, we further elaborate on the latter scenario with the aid of Fig. 5.3.

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. 5.4. 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 because it is the one that has matched in all corresponding previous row and column-wise tiles.

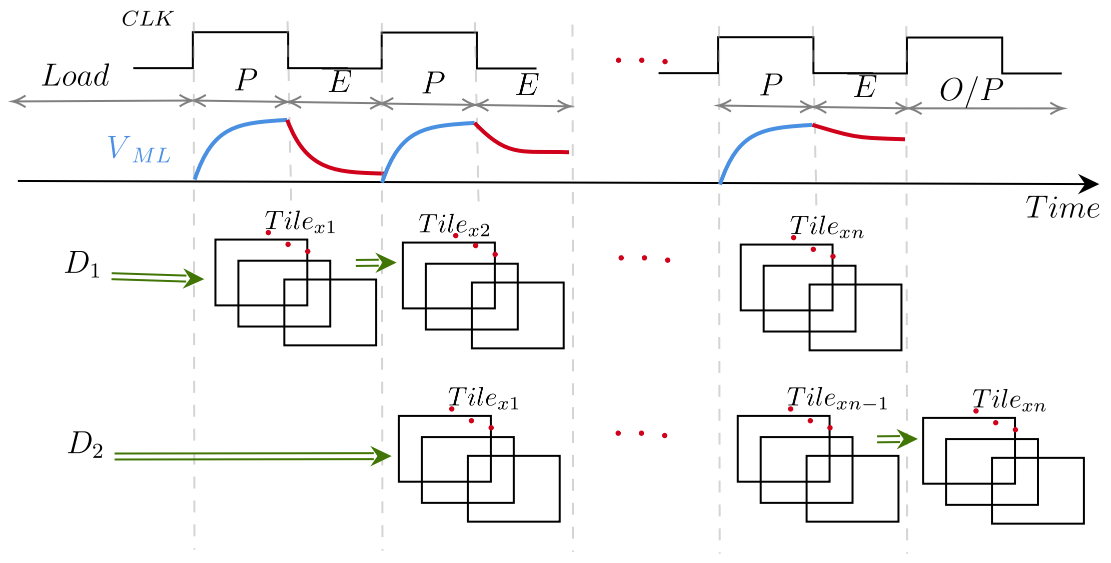


Figure 5.4: Timing Diagram: $Tile_{x1}$ and $Tile_{x2}$ represent the row-wise tiles of the first and second column-wise TCAM tiles. P and E are short for Precharge and Evaluate respectively. V_{in} is the voltage measured across C_{in}. First, data is loaded into the TCAM tiles. During precharge cycles, tiles are precharged, and during evaluate, the data is searched across the tiles. Column-wise TCAM tiles operate sequentially while row-wise tiles operate in parallel.

Selective Precharge: In this work, we adopt sequential evaluation across multiple column-wise TCAM tiles for each input to enable Selective Precharge (SP). By relying on the proposed SP circuit presented in Fig. 5.5, a row that mismatches in the previous column-wise tile for a given input is not precharged nor evaluated in the current tile. In particular, if an input mismatches a given row in some $Tile_{ij}$ (stage *k-1*), the SP circuit deactivates the precharge circuitry and the 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. As such, the advantage of using the SP circuit is depicted in the reduction of the energy-delay product presented in Fig. 6.1c (see following chapter). We note that if an input at stage *k-1* matches some row, the SA and precharge circuitry of the corresponding row in stage k are activated.

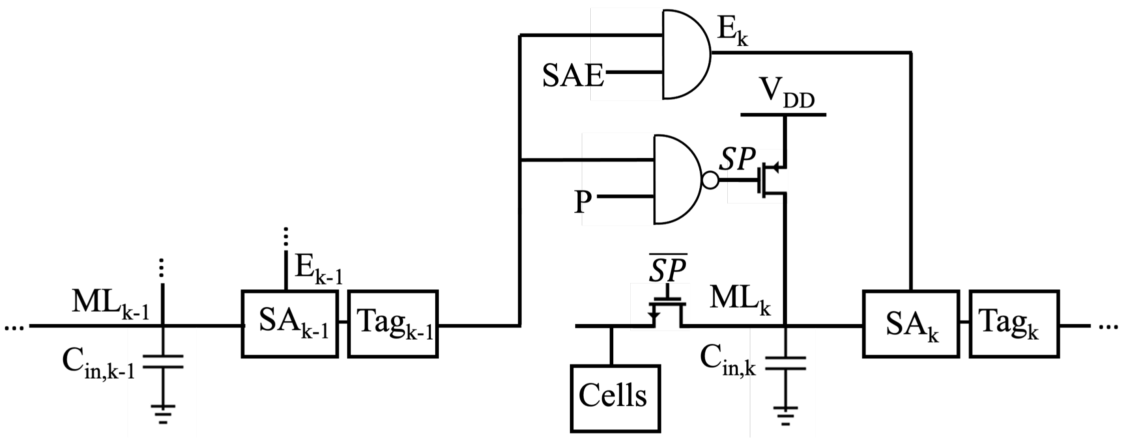


Figure 5.5: Selective Precharge Circuit.

5.3.2 Simulation

The synthesizer relies on simulations to carry out energy, latency, and accuracy evaluations for the design with and without hardware non-idealities. Herein, we adopt the following assumptions.

Technology: For energy, latency, dynamic range (D_{cap}), and optimal evaluation time (T_{opt}) calculations, the ReCAM functional simulator relies on $16nm$ technology parameters summarized in Table 5.1.

Table 5.1: $16nm$ predictive technology models parameters used for the ReCAM arrays.

Parameter	Definition	Value
R_{LRS}	Low Resistance State	$5k\Omega$
R_{HRS}	High Resistance State	$2.5M\Omega$
R_{ON}	ON Transistor Resistance	$15k\Omega$
R_{OFF}	OFF Transistor Resistance	$24.25M\Omega$
C_{in}	Sensing Capacitance	$50fF$
V_{DD}	Supply Voltage	1V

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 Eqn. (5.6) 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 the maximum value found as shown in Table 5.2.

Table 5.2: D_{cap} values and the chosen target TCAM size S.

D_{cap} Upper Bound	Max # of Cells/Row	Chosen S
0.2	154	128
0.3	86	64
0.4	53	32
0.5	33	32
0.6	21	16

Energy: For energy calculation purposes, the total energy per an active TCAM row per an input is calculated as follows.

$$E_{row}^{active} = E_{TCAM} + E_{sa} \tag{5.7}$$

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} is derived based on the closed form in chapter 4 (see section 4.3.2). We remind the reader that the evaluation duration is T_{opt}, where T_{opt} is the time used to sense the match line for evaluation purposes and is defined as follows.

$$T_{opt} = C_{in} * ln(\frac{R_{fm}}{R_{1mm}}) * \frac{(R_{fm} * R_{1mm})}{(R_{fm} - R_{1mm})} \tag{5.8}$$

We assume the worst-case scenario for the energy calculations, 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 the specific 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 number of 1T1R cells needed is $log_2(\#classes)$) followed by a SA adapted from [91]. Accordingly, E_{mem} is the energy dissipated in the 1T1R cell(s) and the SA adapted from [91]. The average energy per input can then be computed from all the input data points.

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 (5.9). The average total latency, per input, $\bar{T}_{total}$, is then given by $\bar{T}_{total} = N_{cwd} * (T_{cwd}) + T_{mem}$.

$$T_{cwd} = 3 * (\tau_{pchg}) + T_{opt} + T_{sa} \tag{5.9}$$

where T_{sa} (determined via SPICE simulations) is the time needed for the SA to sense a match or a mismatch, and T_{mem} is the time needed to access the 1T1R cell(s) storing the class label of the surviving row. We note that in the case of multiple 1T1R cells, these are accessed in parallel. In addition, our simulator operates with the maximum frequency (unless otherwise mentioned) which is given as follows.

$$f_{max} = \frac{1}{max((3 * (\tau_{pchg}) + T_{opt} + T_{sa}), T_{mem})} \tag{5.10}$$

This equation is used to determine the operating frequency for any array size. For instance, operating frequency for an array width of 128 is 1 GHz under the parameters reported in Table 5.1.

Sense Amplifier Reference Voltage: We utilize two different reference voltages, $V_{ref,1}$ and $V_{ref,2}$, for the SAs. $V_{ref,1}$ is used for the SAs of all TCAM tiles except the row-wise tiles of the last column-wise division. For these tiles, $V_{ref,2}$ is utilized instead to accommodate for the presence of "masked don't care" cells that result in different values for V_{fm} and V_{1mm}. We note that the sense amplifier design adopted for sensing the match line is based on the double-tail sense amplifier proposed in [92], and is implemented in the 16nm technology node.

Device Defects 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 $SA1$) [93]. We study the SAF problem by inducing bit flips in the encoded TCAM cells as indicated in Table 5.3 and using the following probability percentage values: $SA0 = [0, 0.1, 0.5, 1, 5]\%$ and $SA1 = [0, 0.1, 0.5, 1, 5]\%$.

Table 5.3: TCAM induced bit flips due to SAF.

Target Encoded Bit	{R1, R2}	Encoded Bit w/ SA0	Encoded Bit w/ SA1
0	{HRS, LRS}	x or 0	0 or {LRS, LRS}
1	{LRS, HRS}	x or 1	1 or {LRS, LRS}
x	{HRS, HRS}	x	x or 0 or 1 or {LRS, LRS}

Manufacturing Variability DT2CAM is studied (accuracy-wise) under the effect of manufacturing variability in the SAs similar to [92]. We emulate this variability by applying random offsets to $V_{ref1,2}$ of the individual SAs for a given TCAM division. $V_{ref1,2} = \mu_{V_{ref1,2}} \pm \sigma_{sa} * z$ where $\sigma_{sa} \in [0, 0.03, 0.04, 0.05, 0.1]V$ and $z\ N(0,1)$.

Input Encoding Noise To study the effect of input noise, we induce random noise in the normalized input features dataset with the following variability:
$\sigma_{in} \in [0, 0.001, 0.005, 0.01, 0.02, 0.05, 0.1]$ and observe the change in recognition accuracy.

Area We estimate the average area, A, of the proposed synthesizer design according to the following formula.

$$A = N_t * (S^2 * A_{2T2R} + S * (A_{SA} + A_{DFF} + A_{SP})) + S * \log_2(N_c) * (A_{1T1R} + A_{SA2}) \quad (5.11)$$

where A_{2T2R}, A_{DFF}, A_{SP}, and A_{1T1R} are the areas of 2T2R (i.e. TCAM cell), D-flipflop (i.e. tag), selective precharge circuit (Fig. 5.5), and 1T1R (for storing class labels for the surviving row) respectively. Moreover, A_{SA} and A_{SA2} represent the areas of the double-tail SA (used for sensing the match line) and the SA adopted from [91] (used along with the 1T1R cell(s)). N_c is the number of class labels used for some dataset.

Table 5.4: Description of the utilized datasets.

Dataset	# Instances	# Features	# Classes
Iris	150	4	3
Diabetes	768	8	2
Haberman	306	3	2
Car	1728	6	4
Cancer	569	30	2
Credit	120269	10	2
Titanic	887	6	2
Covid	33599	4	2

5.4 Implementation Details

Our framework is built in Python where we extract and parse the decision tree, then reduce it as shown in the column reduction step (Fig. 5.2). This is followed by ReCAM functional synthesizer to perform the mapping and hardware simulations. To test DT2CAM, we rely on eight datasets, six of which are from the UCI Repository and Kaggle [89, 94, 95]. In particular, we utilize the *Fisher's Iris* (denoted as Iris), *Haberman's Survival* (denoted as Haberman), *Car Evaluation* (denoted as Car), and *Breast Cancer Wisconsin (Diagnostic)* (denoted as Cancer) datasets from the UCI repository. The *Give Me Some Credit (training)* (denoted as Credit) and *Pima Indian Diabetes* (denoted as Diabetes) datasets are taken from Kaggle. In addition, from Stanford's CS109 website [96], we utilize the *Titanic* dataset. To evaluate our framework on a more recent dataset, we also test it on the *COVID-19* (denoted as Covid) dataset compiled by [97]. The details of the used datasets are summarized in Table 5.4.

Note that in some datasets, we omit some incomplete instances and some features that are unique for each data instance (like ID or name etc.), and these modifications are reflected in the table. Without loss of generality, we use the same split percentage of the data in the aforementioned datasets to generate the decision trees: 90% and 10% for training and testing, respectively.

Chapter 6

Results and Comparison

In this section, we discuss the results collected by the ReCAM functional synthesizer, and then compare DT2CAM to other state-of-the-art hardware accelerators.

6.1 Energy/Throughput/EDP Analysis

In Fig. 6.1a, we plot the energy per decision (dec) vs throughput for all eight datasets and with different target S value, 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, being the largest dataset, consumes the highest energy and has the lowest throughput, while inference on Iris, being the 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 as shown in Table 6.1), 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. 6.1b.

For the remaining 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 attributed to the fact that small datasets are represented by at most two tiles when $S = 128$ thereby not benefiting from deactivated rows due to mismatching 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. 6.1b). Only the Iris dataset favors smaller S values when it comes to EDP due to its extremely small LUT size.

In addition, in Fig. 6.1c, we present the % reduction of EDP when SP circuit is used compared to when it is not. For all datasets where at least two column-wise tiles are required for different target size S, we see a reduction in the EDP. This shows the advantage of using the SP circuit as it saves energy. In particular, the Credit dataset with SP circuit achieves the highest reduction in EDP (around 90%). This is expected as it is the largest dataset with the largest produced LUT, which in turn yields a large number of column-wise tiles. The large number of column-wise tiles benefits from the SP circuit by evaluating only few rows in each tile.

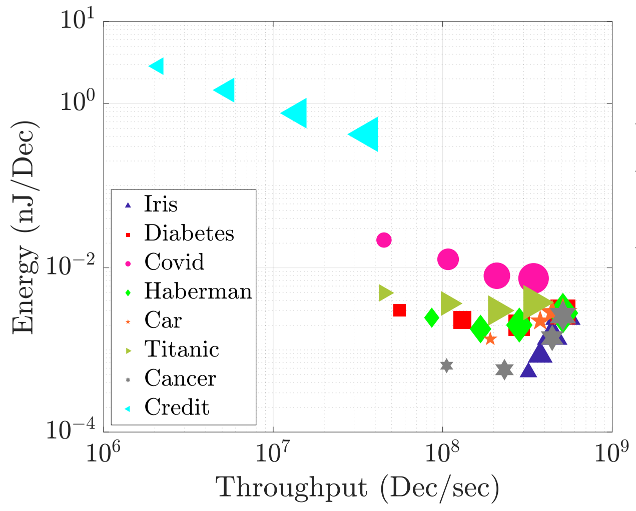
6.2 Analysis with Hardware Non-idealities

We study the DT2CAM framework in the context of accuracy loss for different target size S, and under the described hardware non-idealities before. Without loss of generality, we focus on the following datasets: Diabetes, Cancer, and Covid. We note that for all the

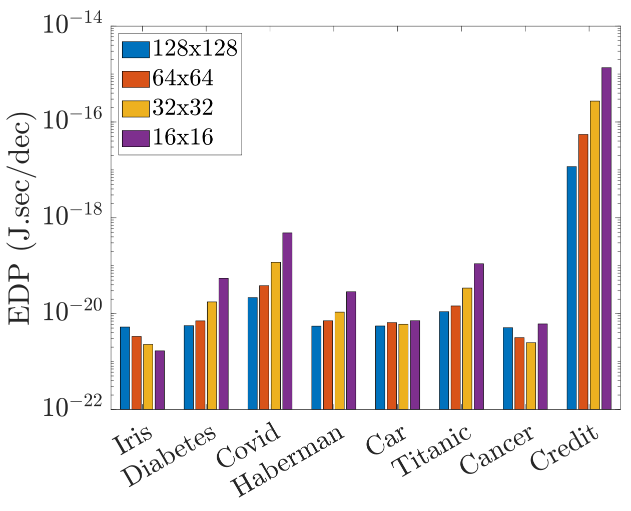
Table 6.1: Number of TCAM tiles for the different datasets.

		# TCAM Tiles: $\mathbf{N_{rwd} \times N_{cwd}}$			
Dataset	**LUT Size**	**$\mathbf{16 \times 16}$**	**$\mathbf{32 \times 32}$**	**$\mathbf{64 \times 64}$**	**$\mathbf{128 \times 128}$**
Iris	9×12	1×1	1×1	1×1	1×1
Diabetes	120×123	8×8	4×4	2×2	1×1
Haberman	93×71	6×5	3×3	2×2	1×1
Car	76×20	5×2	3×1	2×1	1×1
Cancer	23×52	2×4	1×2	1×1	1×1
Credit	8475×3580	530×224	265×112	133×56	67×28
Titanic	191×150	12×10	6×5	3×3	2×2
Covid	441×146	28×10	14×5	7×3	4×2

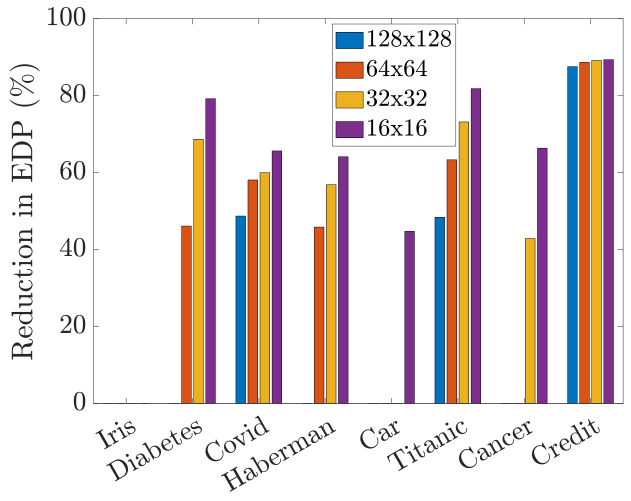
datasets under study, the accuracy evaluated by the ReCAM synthesizer for ideal hardware (without non-idealities) matches the *accuracy obtained in Python* (hereon denoted as *golden accuracy*) when inference is performed. Hence the accuracy loss of each dataset is measured concerning the corresponding golden accuracy. From Fig. 6.2, the target size S does not impact the accuracy loss in the presence of non-idealities for Diabetes and Cancer. For the Covid dataset, which 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 $SA'b' = 0\%$. Note that the cases for $SA'b' = 0.5\%$ are truncated 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, and this applies to all datasets under study. In fact, for some cases, the input noise reduces the accuracy loss, and this is due to the test dataset itself, and how it changes with the induced input noise. We finally note that the stuck-at-fault 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.



(a)



(b)

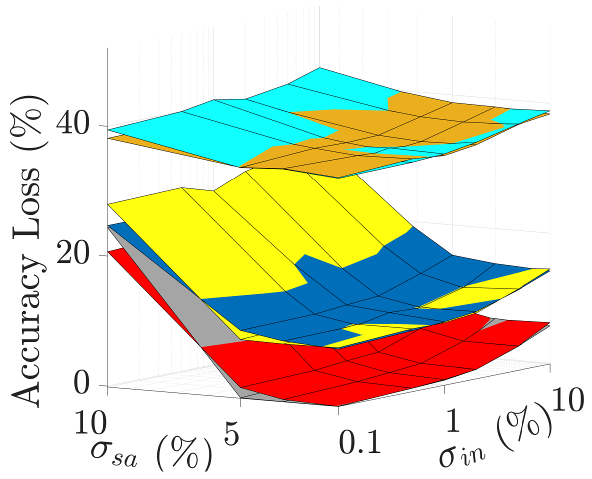


(c)

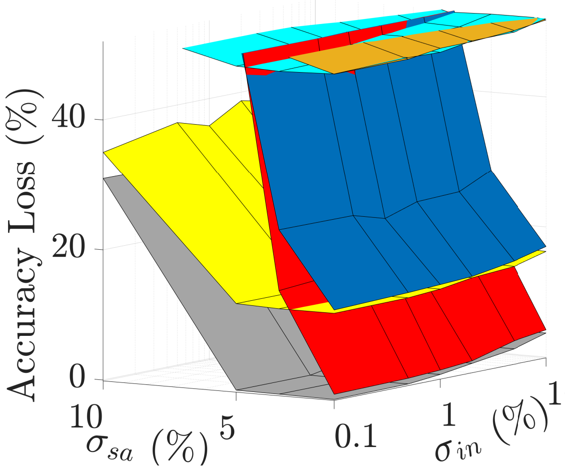
Figure 6.1: Per inference decision: (a) Energy vs throughput for the different datasets. The shape size determine the target size of the TCAM(s). From small to large shapes: 16×16, 32×32, 64×64, and 128×128. (b) Energy-Delay-Product, and (c) Reduction in EDP when SP is used compared to when it is not used.

6.3 Comparison with Other Hardware Accelerators

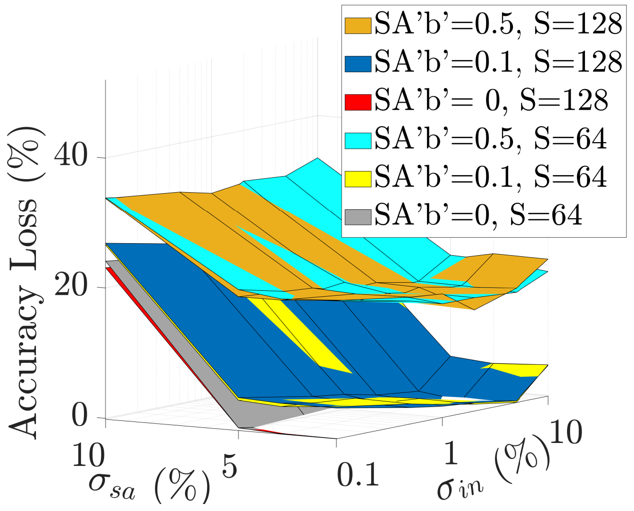
In Table 6.2, we summarize the per decision throughput and energy for our framework and other hardware accelerators for decision tree inference ([22, 80, 83, 98]). Fig. 6.3 shows the the per decision energy versus throughput for the IMC-based frameworks. For DT2CAM, we assume a 2000x2048 original TCAM size, divided into 128x128 ($S = 128$) tiles to mimic



(a)



(b)



(c)

Figure 6.2: Percent Accuracy loss due to different hardware non-idealities (input noise, sense amplifier manufacturing variability and stuck-at-fault problem) for five datasets: (a) Diabetes, (b) Covid, and (c) Cancer. $SA'b' = x$ is equivalent to $SA0 = SA1 = x\%$.

inference on the traffic dataset problem. In particular, we take into consideration the 2000 rows by 256 features reported for the traffic dataset in [22], 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 the ASIC accelerators ([80, 98]) and IMC based accelerators (ASIC IMC [83], ACAM [22]), our proposed DT2CAM achieves the highest throughput ($58.8E6\ Dec/s$) and consumes the lowest energy ($0.17nJ/Dec$). Furthermore, our proposed pipelined design has the same throughput while consuming $1.73x$ lower energy than the pipelined ACAM design [22].

Furthermore, we report the average area of DT2CAM (based on Eqn. (5.11)), and we report the average area per bit (i.e. $A/\#TCAMCells$) in Table 6.2. Compared to the area reported for the analog CAM framework [22], we achieve about $3.8x$ and $17.5x$ reduction in area overhead and area/bit respectively.

We define a figure of merit, FOM, to better compare the accelerators' performances as follows.

$$FOM = EDP * A \tag{6.1}$$

Accordingly, the lower the FOM (i.e., smaller energy-delay product and area), the better the performance. Our sequential/parallel DT2CAM framework has 17.8x /6.3x better FOM compared to the ACAM realization.

Table 6.2: Comparison with SOTA hardware accelerators. P refers to pipelined accelerators.

Accelerator	Technology (nm)	f_{clk} (GHz)	Throughput (Dec/s)	Energy (nJ/dec)	Area (mm^2)	Area/bit (μm^2/bit)	FOM (J.sec.mm^2)
ASIC [80]	65	0.2	30	186.7E3	-	-	-
ASIC [98]	65	0.25	60	460E3	-	-	-
ASIC IMC [83]	65	1	364.4E3	19.4	-	-	-
ACAM [22]	16	1	20.8E6	0.17	0.266	0.299	2.17E-18
P-ACAM [22]	16	1	333E6	0.17	0.266	0.299	1.36E-19
DT2CAM_128	16	1	58.8E6	0.098	0.07	0.017	1.22E-19
P-DT2CAM_128	16	1	333E6	0.098	0.07	0.017	2.15E-20

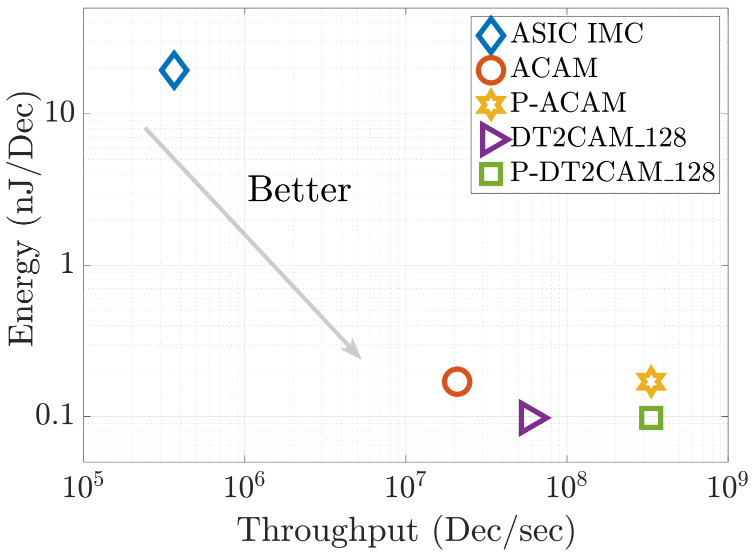


Figure 6.3: Energy vs. Throughput for our proposed DT2CAM and other SOTA hardware accelerators.

Chapter 7

Conclusion and Future Work

In conclusion, we first presented a comparative study between the capacitive and resistive sensing schemes of 2T-2R TCAMs relying on mathematical formulations. The resistive design showed up-to 260× FOM improvements over the capacitive design, while taking dynamic range constraints into consideration. We also presented an adder application that maintained our theoretical and SPICE results. After that, we proposed DT2CAM, a decision tree to the ReCAM framework which is capable of evaluating the energy, latency, and accuracy of performing decision tree inference using TCAMs (resistive in particular, with capacitive sensing schemes) with and without hardware non-idealities. The proposed framework comprises two main phases: the DT-HW compiler which maps a decision tree graph into a look-up table and the ReCAM functional synthesizer which maps the look-up table into ReCAM arrays (with capacitive sensing scheme) and performs simulations. Experiments on various datasets with varying the number of features and complexity 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).

For the future work, we have been planning to extend our research on the following topics:

1. Extending the DT2CAM framework: We will extend our DT2CAM framework, including selective precharge, to accommodate other ReRAM cell typologies, including ACAM [99], potentially resulting in better performance.

2. Exploring DT2CAM with resistive sensing scheme: Motivated by the performance enhancements noted for the resistive sensing scheme compared to the capacitive sensing scheme, we will explore the DT2CAM framework with TCAM designs that rely on the resistive sensing scheme.

3. Enhancing DT2CAM's compiler: Moreover, we plan to include software optimizations (including pruning and quantization) to the compiler of DT2CAM, whereby the decision tree can be optimized for better overall performance when mapped to the ReCAM arrays.

Bibliography

[1] D. R. B. Ly *et al.*, “In-depth characterization of resistive memory-based ternary content addressable memories,” in *2018 IEEE IEDM*, 2018.

[2] S. Dua and X. Du, *Data mining and machine learning in cybersecurity.* CRC press, 2016.

[3] R. C. Deo, “Machine learning in medicine,” *Circulation*, vol. 132, no. 20, pp. 1920–1930, 2015.

[4] A. L. Tarca, V. J. Carey, X.-w. Chen, R. Romero, and S. Drăghici, “Machine learning and its applications to biology,” *PLoS computational biology*, vol. 3, no. 6, p. e116, 2007.

[5] N. Artrith, K. T. Butler, F.-X. Coudert, S. Han, O. Isayev, A. Jain, and A. Walsh, “Best practices in machine learning for chemistry,” *Nature chemistry*, vol. 13, no. 6, pp. 505–508, 2021.

[6] N. M. Ball and R. J. Brunner, “Data mining and machine learning in astronomy,” *International Journal of Modern Physics D*, vol. 19, no. 07, pp. 1049–1106, 2010.

[7] J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[8] C. Kingsford and S. L. Salzberg, “What are decision trees?” *Nature biotechnology*, vol. 26, no. 9, pp. 1011–1013, 2008.

[9] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[10] R. S. Williams, “What’s next?[the end of moore’s law],” *Computing in Science & Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[11] D. Ielmini and H.-S. P. Wong, “In-memory computing with resistive switching devices,” *Nature electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[12] C. C. Foster, *Content addressable parallel processors.* John Wiley & Sons, Inc., 1976.

[13] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive tcam accelerator for data-intensive computing," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 339–350.

[14] C. E. Graves, C. Li, X. Sheng, D. Miller, J. Ignowski, L. Kiyama, and J. P. Strachan, "In-memory computing with memristor content addressable memories for pattern matching," *Advanced Materials*, vol. 32, no. 37, p. 2003437, 2020.

[15] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "Ac-dimm: associative computing with stt-mram," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 189–200.

[16] I. Arsovski, T. Chandler, and A. Sheikholeslami, "A ternary content-addressable memory (tcam) based on 4t static storage and including a current-race sensing scheme," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 1, pp. 155–158, 2003.

[17] 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.

[18] H. E. Yantir, *Efficient acceleration of computation using associative in-memory processing*. University of California, Irvine, 2018.

[19] S. K. Maurya and L. T. Clark, "A dynamic longest prefix matching content addressable memory for ip routing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 6, pp. 963–972, 2010.

[20] S. A. Schuster, H. Nguyen, E. A. Ozkarahan, and K. C. Smith, "Rap.2 - an associative processor for databases and its applications," *IEEE Trans. Computers*, vol. 28, no. 6, pp. 446–458, 1979.

[21] O. Castañeda, M. Bobbett, A. Gallyas-Sanhueza, and C. Studer, "Ppac: A versatile in-memory accelerator for matrix-vector-product-like operations," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 149–156.

[22] 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.

[23] A. Biswas and A. P. Chandrakasan, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 488–490.

[24] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, "8t sram cell as a multibit dot-product engine for beyond von neumann computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2556–2567, 2019.

[25] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[26] H.-S. P. Wong, H.-Y. Lee, S. Yu, Y.-S. Chen, Y. Wu, P.-S. Chen, B. Lee, F. T. Chen, and M.-J. Tsai, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, no. 6, pp. 1951–1970, 2012.

[27] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[28] X. Fong, Y. Kim, K. Yogendra, D. Fan, A. Sengupta, A. Raghunathan, and K. Roy, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 1–22, 2015.

[29] S.-Y. Wang, C.-W. Huang, D.-Y. Lee, T.-Y. Tseng, and T.-C. Chang, "Multilevel resistive switching in ti/cu x o/pt memory devices," *Journal of Applied Physics*, vol. 108, no. 11, p. 114110, 2010.

[30] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature electronics*, vol. 1, no. 1, pp. 52–59, 2018.

[31] D. Niu, Y. Xiao, and Y. Xie, "Low power memristor-based reram design with error correcting code," in *17th Asia and South Pacific Design Automation Conference*. IEEE, 2012, pp. 79–84.

[32] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[33] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (cam) circuits and architectures: A tutorial and survey," *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712–727, 2006.

[34] M. A. Bahloul *et al.*, "Design and analysis of 2t-2m ternary content addressable memories," in *IEEE MWSCAS*, Aug 2017, pp. 1430–1433.

[35] K. Eshraghian *et al.*, "Memristor mos content addressable memory (mcam): Hybrid architecture for future high performance search engines," *IEEE TVLSI*, vol. 19, no. 8, p. 1407–1417, 2011.

[36] Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[37] M. A. Bahloul, M. E. Fouda, R. Naous, M. A. Zidan, A. M. Eltawil, F. Kurdahi, and K. N. Salama, "Design and analysis of 2t-2m ternary content addressable memories," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2017, pp. 1430–1433.

[38] M. Imani *et al.*, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 373–378.

[39] J. Li *et al.*, "1 mb 0.41 µm^2 2t-2r cell nonvolatile tcam with two-bit encoding and clocked self-referenced sensing," *IEEE JSSC*, 2014.

[40] M. Imani *et al.*, "Remam: Low energy resistive multi-stage associative memory for energy efficient computing," in *IEEE ISQED*, 2016.

[41] T. V. Mahendra *et al.*, "Energy-efficient precharge-free ternary content addressable memory (tcam) for high search rate applications," *IEEE TCAS I: Regular Papers*, pp. 1–13, 2020.

[42] K. J. Thurber and L. D. Wald, "Associative and parallel processors," *ACM Computing Surveys (CSUR)*, vol. 7, no. 4, pp. 215–255, 1975.

[43] M. Hout, M. E. Fouda, R. Kanj, and A. M. Eltawil, "In-memory multi-valued associative processor," *arXiv preprint arXiv:2110.09643*, 2021.

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

[45] C. Seifert, A. Aamir, A. Balagopalan, D. Jain, A. Sharma, S. Grottel, and S. Gumhold, "Visualizations of deep neural networks in computer vision: A survey," in *Transparent data mining for big and small data*. Springer, 2017, pp. 123–144.

[46] J. Hernavs, M. Ficko, L. Berus, R. Rudolf, and S. Klančnik, "Deep learning in industry 4.0–brief overview," *Novi Sad*, vol. 21, no. 2, p. 1, 2018.

[47] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[48] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[49] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020.

[50] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine learning for predictive maintenance: A multiple classifier approach," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 812–820, 2014.

[51] 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.

[52] F. Song, Y. Diao, J. Read, A. Stiegler, and A. Bifet, "Exad: A system for explainable anomaly detection on big data traces," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2018, pp. 1435–1440.

[53] P. R. Magesh, R. D. Myloth, and R. J. Tom, "An explainable machine learning model for early detection of parkinson's disease using lime on datscan imagery," *Computers in Biology and Medicine*, vol. 126, p. 104041, 2020.

[54] 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.

[55] T. Thomas, A. P Vijayaraghavan, and S. Emmanuel, "Applications of decision trees," in *Machine learning approaches in cyber security analytics*. Springer, 2020, pp. 157–184.

[56] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[57] A. Priyam, G. Abhijeeta, A. Rathee, and S. Srivastava, "Comparative analysis of decision tree classification algorithms," *International Journal of current engineering and technology*, vol. 3, no. 2, pp. 334–337, 2013.

[58] A. Rathee and R. P. Mathur, "Survey on decision tree classification algorithms for the evaluation of student performance," *International Journal of Computers & Technology*, vol. 4, no. 2a1, pp. 244–247, 2013.

[59] "1.10. decision trees," https://scikit-learn.org/stable/modules/tree.html, accessed: 2022-05-19.

[60] S. A. Kumar *et al.*, "Efficiency of decision trees in predicting student's academic performance," 2011.

[61] T. Blank, "A survey of hardware accelerators used in computer-aided design," *IEEE Design & Test of Computers*, vol. 1, no. 3, pp. 21–39, 1984.

[62] T. N. Theis and H.-S. P. Wong, "The end of moore's law: A new beginning for information technology," *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017.

[63] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[64] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–12.

[65] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[66] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[67] G. Roos, "Fpga acceleration card delivers on bandwidth, speed, and flexibility," 2019.

[68] T. P. Morgan, "Drilling into microsoft's brainwave soft deep learning chip," *Retrieved from¡¿, Aug*, vol. 24, p. 8, 2017.

[69] D. Chiou, "The microsoft catapult project," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE Computer Society, 2017, pp. 124–124.

[70] "Nvidia tesla v100 tensor core gpu," https://www.nvidia.com/en-us/data-center/tesla-v100/, accessed: 2022-05-19.

[71] R. Krashinsky, O. Giroux, S. Jones, N. Stam, and S. Ramaswamy, "Nvidia ampere architecture in-depth," *NVIDIA Developer Blog*, 2020.

[72] R. Smith, "16gb nvidia tesla v100 gets reprieve; remains in production," 2018.

[73] "Taking a deeper look at amd radeon instinct gpus for deep learning," https://blog.exxactcorp.com/taking-deeper-look-amd-radeon-instinct-gpus-deep-learning/, accessed: 2022-05-19.

[74] "Amd announces radeon instinct mi60 mi50 accelerators powered by 7nm vega," https://www.anandtech.com/show/13562/amd-announces-radeon-instinct-mi60-mi50-accelerators-powered-by-7nm-vega, accessed: 2022-05-19.

[75] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, W.-m. W. Hwu, J. P. Strachan, K. Roy *et al.*, "Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 715–731.

[76] "The linley group microprocessor report highlights brainchip's akida spiking-neural-network processor," https://brainchip.com/the-linley-group-microprocessor-report-highlights-brainchips-akida-spiking-neural-\ network-processor-brainchip-311019-01/, accessed: 2022-05-19.

[77] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, "Pipelined decision tree classification accelerator implementation in fpga (dt-caif)," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 280–285, 2013.

[78] A. Zoulkatni, C. Kachris, and D. Soudris, "Hardware acceleration of decision tree learning algorithm," in *2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2020, pp. 1–6.

[79] R. Struharik, "Decision tree ensemble hardware accelerators for embedded applications," in *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*. IEEE, 2015, pp. 101–106.

[80] 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.

[81] 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.

[82] X. Yin, F. Müller, A. F. Laguna, C. Li, W. Ye, Q. Huang, Q. Zhang, Z. Shi, M. Lederer, N. Laleni *et al.*, "Deep random forest with ferroelectric analog content addressable memory," *arXiv preprint arXiv:2110.02495*, 2021.

[83] 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.

[84] H. Abunahla and B. Mohammad, *Memristor Device Overview*. Cham: Springer International Publishing, 2018, pp. 1–29.

[85] A. Grossi *et al.*, "Experimental investigation of 4-kb rram arrays programming conditions suitable for tcam," *IEEE TVLSI Systems*, 2018.

[86] H. E. Yantır *et al.*, "A two-dimensional associative processor," *IEEE TVLSI*, vol. 26, no. 9, pp. 1659–1670, 2018.

[87] 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.

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

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

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] "Give me some credit." [Online]. Available: https://www.kaggle.com/c/GiveMeSomeCredit/data?select=cs-training.csv

[95] U. M. Learning, "Pima indians diabetes database," Oct 2016. [Online]. Available: https://www.kaggle.com/uciml/pima-indians-diabetes-database

[96] "A titanic probability." [Online]. Available: https://web.stanford.edu/class/archive/cs/cs109/cs109.1166/problem12.html

[97] 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.

[98] 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.

[99] J. Bazzi, J. Sweidan, M. E. Fouda, R. Kanj, and A. M. Eltawil, "Efficient analog cam design," *arXiv preprint arXiv:2203.02500*, 2022.