UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

BRUNNO ALVES DE ABREU

Automated Design Space Exploration of Approximate VLSI Architectures for Low-Power Tree-Based Learning Models

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Microeletronics

Advisor: Prof. Dr. Sergio Bampi
Coadvisor: Prof. Dr. Mateus Grellert

Porto Alegre
December 2023

CIP — CATALOGING-IN-PUBLICATION

Abreu, Brunno Alves de

Automated Design Space Exploration of Approximate VLSI Architectures for Low-Power Tree-Based Learning Models / Brunno Alves de Abreu. – Porto Alegre: PGMICRO da UFRGS, 2023.

117 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica, Porto Alegre, BR–RS, 2023. Advisor: Sergio Bampi; Coadvisor: Mateus Grellert.

1. Low-Power. 2. Machine Learning. 3. Decision Trees. 4. Random Forests. 5. Approximate Computing. I. Bampi, Sergio. II. Grellert, Mateus. III. Título.

AGRADECIMENTOS

First of all, I would like to thank my parents, Fernando César Ávila de Abreu and Jaqueline Alves de Abreu, for fully supporting me in my choices and giving me advices when needed.

To all my professors, who have made their best to teach me all they could and helped me to find outside sources to learn even more.

To all my classmates, from my undergraduate and graduate studies, with whom I have shared the difficulties, stressful and happy moments we all have been through during our graduation years.

To my research colleagues Guilherme Paim, Leandro Rocha, Eduardo Costa, Gustavo Santana, Thomas Fontanari, Vitor Lima, who have helped me, directly or indirectly, in my maturation as a researcher, and also in the completion of this work. To the colleagues from my laboratory who, even though worked on other research topics, were great companies during coffee time.

Special thanks to my advisor Sergio Bampi, my co-advisor Mateus Grellert, and my supervisor during the period I spent in INESC-ID, prof. Paulo Flores, who have dedicated countless hours in helping me with doubts I had, to work with me on many other side projects, and to encourage me when I lacked the motivation.

ABSTRACT

The evolution in CMOS technology has led to an increased computational capacity of electronic devices, enabling complex applications to be processed in embedded platforms. An example of this is the growth of machine learning (ML) applications processed on-device. These techniques are efficient for pattern-recognition and prediction, but require huge amounts of data and operations to generate models that can learn efficiently. Hence, when considering devices with battery constraints, such as wearables, simpler models like tree-based ones may be more suitable, given their power/energy efficiency. ML algorithms allow for the insertion of errors without necessarily compromising the output, making approximate computing (AxC) techniques promising alternatives to further decrease the power/energy costs of these applications. The problem that arises from this approach is that the use of AxC combined with model selection substantially increases the amount of parameters that must be considered and optimized during design space exploration (DSE). This thesis proposes the use of automated frameworks to generate ML VLSI accelerators and perform automatic synthesis, for different degrees of approximation, greatly speeding up the DSE process. The proposed frameworks automatically map ML models to HDL, employing AxC techniques in different layers to achieve improved energy/area savings. The efficiency of the proposed frameworks is assessed by exploring approximate VLSI architectures for Decision Trees (DT) and Random Forests (RF). Different model/design parameters were tested, namely tree depth, number of trees, and quantization level, adding up to 1540 compared designs. The other proposed frameworks explore techniques for approximating comparators and performing gate-level pruning in DTs/RFs. The models generated from the initial framework present power reductions of $10\times$ or more for the same inference throughput reported in previous works. The remaining frameworks also obtained significant savings compared to the current state-of-the-art. The main contribution of this thesis is to enable an automated and comprehensive DSE of ML models, allowing designers to make a better-informed assessment of the trade-offs involved in this process.

Keywords: Low-Power. Machine Learning. Decision Trees. Random Forests. Approximate Computing.

Automação da Exploração do Espaço de Projeto de Arquiteturas VLSI Aproximadas para Modelos de Aprendizado Baseados em Árvores de Baixa Potência

RESUMO

A evolução da tecnologia CMOS tem levado a um aumento da capacidade computacional de dispositivos eletrônicos, permitindo o processamento de aplicações complexas em plataformas embarcadas. Um exemplo disso é o aumento de aplicações de aprendizado de máquina (ML) processadas no dispositivo. Essas técnicas são eficientes para reconhecimento de padrões e predição, mas requerem muitos dados e operações para gerar modelos eficientes. Assim, ao considerar dispositivos com limitações de bateria, como *wearables*, modelos simples como os baseados em árvores podem ser mais adequados, dada sua eficiência em potência/energia. Algoritmos de ML permitem a inserção de erros sem necessariamente comprometerem a saída, tornando técnicas de computação aproximada (AxC) alternativas promissoras para diminuir os custos energéticos dessas aplicações. O problema que surge é que o uso de AxC combinado com a seleção do modelo aumenta o número de parâmetros a serem considerados durante a exploração do espaço de projeto (DSE). Essa tese propõe o uso de *frameworks* para gerar aceleradores VLSI de ML e fazer a síntese automaticamente, para diferentes graus de aproximação, acelerando a DSE. Os *frameworks* propostos mapeiam modelos de ML para HDL, utilizando técnicas de AxC em diferentes camadas para atingir melhores economias de energia/área. A eficiência dos *frameworks* propostos é verificada explorando arquiteturas VLSI aproximadas para árvores de decisão (DT) e florestas randômicas (RF). Diferentes parâmetros foram testados, como profundidade da árvore, número de árvores e nível de quantização, somando 1540 *designs*. Os outros *frameworks* propostos exploram técnicas de aproximação de comparadores e *gate-level pruning* nas DTs/RFs. Os modelos gerados pelo *framework* inicial apresentam reduções de potência de 10$\times$ ou mais para a mesma vazão de inferência reportada em trabalhos anteriores. Os outros *frameworks* também obtiveram economias de potência significativas comparados ao estado-da-arte. A principal contribuição dessa tese é permitir uma DSE de modelos de ML automatizada, permitindo que projetistas façam uma verificação mais precisa dos *trade-offs* envolvidos no processo.

Palavras-chave: Baixa Potência, Aprendizado de Máquina, Árvores de Decisão, Florestas Randômicas, Computação Aproximada.

LIST OF ABBREVIATIONS AND ACRONYMS

ADL	Activities of Daily Living
AI	Artificial Intelligence
AIG	And-Inverter Graph
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AxC	Approximate Computing
C2PAx	Complexity-Aware Constant Parameter Approximation
CapTBL	Capacitance Table
CART	Classification and Regression Tree
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
CPD	Critical Path Delay
DC	Don't Care
DL	Deep Learning
DNN	Deep Neural Network
DSE	Design Space Exploration
DT	Decision Tree
EDA	Electronic Design Automation
FACT	First G-APD Cherenkov Telescope
FC	Fully-Connected
FDSOI	Fully Depleted Silicon On Insulator
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
FSM	Finite-State Machine

GPP	General-Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
HLS	High-Level Synthesis
IEEE	Institute of Electrical and Electronic Engineers
IoT	Internet of Things
IP	Intellectual Property
LEF	Library Exchange Format
LSB	Least-Significant Bit
LUT	Look-Up Table
MAC	Multiply-Accumulate
MAJ	Majority
MIG	Majority-Inverter Graph
MIN	Minority
ML	Machine Learning
NN	Neural Network
NPU	Neural Processing Unit
OOM	Out of Memory
PLE	Physically-aware Layout Estimation
PVT	Process/Voltage/Temperature
RAM	Random Access Memory
RF	Random Forest
RTL	Register-Transfer Level
SAIF	Switching Activity Interchange Format
SDF	Standard Delay Format
SK	Scikit-Learn

SoC	System-on-Chip
SVM	Support Vector Machine
TCF	Toggle Count Format
VCD	Value Change Dump
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large-Scale Integration

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The recent advances in semiconductor technology, more specifically in Complementary Metal-Oxide Semiconductor (CMOS) devices, have been leading to significant improvements in several types of applications. However, the CMOS evolution has also led to an increase in the power density in several devices (DENNARD, 2015). This issue has raised a problem denoted by designers as the Dark Silicon era (RAHMANI et al., 2017), which states that, given the limitations of cooling technology, it is not possible to have an entire device working simultaneously. In other words, some portions of the chip will have to be eventually turned off while other parts are operating (SHAFIQUE; HENKEL, 2015; KHDR et al., 2017).

These power and energy issues are even more serious when considering battery-powered devices, e.g., smartphones and wearables, which are strongly correlated to the Internet of Things (IoT). The miniaturization of such devices and the aforementioned advances in CMOS technology have allowed for the popularization of embedded systems operating in several environments and created a tendency for performing operations locally instead of sending them to remote servers (which also leads to increased security). Additionally, predictions from the GSM Association (ASSOCIATION, 2022) indicate that smartphone internet users will account for 60% of the world population by 2025. This popularization of embedded devices, allied with the increasing demands for more complex computational applications and the tendency of computing locally, is creating an urgency to take power dissipation and energy consumption into account. Therefore, it is of utmost importance to create solutions to mitigate this issue.

Wearables are among the embedded devices that have been receiving significant attention lately due to their capability of sensing and computing. Such devices are leading to a transformation in fields such as healthcare, shifting from a hospital-centered system to a person-centered one. Even though some smartphones can perform some sort of tracking regarding motion, wearable devices perform these activities (and many others) much more accurately.

According to Precedence Research (PRECEDENCE, 2022), the global wearable market size was estimated at USD 121.7 billion in 2021, and is expected to represent around USD 392.4 billion by 2030. The use of such wearable devices is allied to the concepts of IoT and edge computing, the latter which indicates a shift from a cloud-centered paradigm towards the extreme edge.

An important factor that contributes to achieving low-power when dealing with wearable and IoT applications is the fact that some of them do not require high frequencies to operate. As it will be seen in Chapter 3, power dissipation is proportional to the operating frequency, which is the reason why low frequencies are strongly related to decreased power dissipation, hence leading to low energy consumption. A thorough research was performed for different models of wearable devices. Bauer et al. (BAUER; WUTZKE; BAUERNHANSL, 2016) present frequencies of 20Hz and 200Hz for gyroscopes in Motorola and Invensense smartwatches, respectively. Beukenhorst et al. (BEUKENHORST et al., 2018) present a frequency value of 50Hz for the Huawei Watch 2 smartwatch. Mortazavi et al. (MORTAZAVI et al., 2015) perform experiments in smartwatches for posture tracking, using Samsung Gear smartwatch, considering sampling rates of 100Hz, 50Hz, and 10Hz. The Apple Developer documentation reports a limit of around 100Hz for accelerometers in iOs devices[1]. Other sources have presented values never higher than 1kHz.

Despite the low frequencies required for wearable applications, power dissipation is still a major issue. The limitations in battery life require the devices and applications to be executed efficiently. Processor manufacturers have realized that some applications are not efficiently executed using general-purpose processors (GPPs). This is due to instruction fetching and the additional hardware that would have to be present and might not be useful if a specific application is being targeted. Due to the popularization of some types of applications, such as machine learning (ML) and deep learning (DL), digital signal processing, video compression, etc., it has become viable to use dedicated hardware architectures for them, given requirements such as throughput, power, and energy.

Along with other techniques, such as clock and data gating, designing dedicated architectures is one of the main strategies to mitigate power, energy, and timing issues, as they are designed solely for specific applications. Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) are platforms for implementing such architectures. ASICs are designed for specific domains and represent the class of circuits that contrast the most from software solutions running on GPPs. The use of FPGAs may represent a good balance standing between the spectrum of GPPs and ASICs. However, when energy efficiency is the main design restriction to take into account, designing architectures in ASICs for dedicated execution of certain key compute-intensive algorithms is still the best approach (SCHMITZ; AL-HASHIMI; ELES, 2004).

[1] Available on: <https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_events>

ML and DL are types of applications that have been receiving significant attention lately, and this has also motivated the development of novel and more complex algorithms. GPPs are too power-hungry to execute these applications in an efficient manner. In addition, they cannot fully explore the degree of parallelism present in most of them. In order to mitigate these issues, such applications are executed in Graphics Processing Units (GPUs), FPGAs, or ASICs (ZHANG et al., 2015). An example of this is the Apple A12 System-on-Chip (SoC), which contains a Neural Processing Unit (NPU) to accelerate the inference of Neural Networks (NN) (FRUMUSANU, 2018).

The need for dedicated circuits for ML and DL (SZE et al., 2017) tends to increase in the following years, given that more cases of successful prediction models are being found. In particular, Deep Neural Networks (DNN) have been receiving the attention of researchers due to their versatility and excellent performance. Efficient solutions based on DNNs can be found in several applications, from the managing of web searches in data centers to mechanisms of object and character recognition in portable devices (LECUN; BENGIO; HINTON, 2015). However, recent works show that even efficient VLSI architectures present significant power dissipation. For example, the Eyeriss, an accelerator for a Convolutional Neural Network (CNN), dissipates 278mW of power (CHEN et al., 2017), which is unsuitable for many embedded platforms.

For simpler problems that do not require convolution or memory-based models, traditional ML techniques represent an important alternative that requires less computing resources. This is the case of tree-based models, such as Decision Trees (DT) and Random Forests (RF). These models are composed of a series of feature-based splits that can be implemented with comparators and basic logic functions. Consequently, tree-based models are simpler than NNs, as they do not require the use of multiply-accumulate (MAC) or other more complex operators and mostly use comparators. This characteristic makes DTs and RFs excellent candidates for applications with severe energy limitations.

In addition to employing simpler operators, other design techniques can be used to further increase energy savings of ML designs. Approximate computing has recently been employed to reduce the power consumption in applications that are resilient to a certain degree of error in their output (VENKATARAMANI et al., 2015), and it is present in many applications for battery-powered devices (CHIPPA et al., 2013). Due to their probabilistic nature, ML algorithms are well-suited for approximate computing approaches. However, the effect of such approximations in the output quality must also be taken into consideration. In that sense, circuit designers must choose the best model based on a

power-accuracy trade-off analysis. As shown in (HAN; ORSHANSKY, 2013), approximate arithmetic operators can be efficiently employed in operational blocks to achieve different qualitative levels, according to the observation of a set of variables in the system (HAN; ORSHANSKY, 2013), e.g., application being targeted, battery life available, etc.

Besides using approximate arithmetic operators, several other techniques of approximate computing can be performed in the design stack, each affecting performance and design parameters (e.g., area, power, delay) to a certain degree. Quantization techniques can be performed in the features to be analyzed, decreasing the size of the operators within the architectures and potentially decreasing the bandwidth required by a memory module fetching the data to the kernels. Other cross-layer approaches can also be performed by modifying the algorithm at the software level so that the hardware underlying it becomes simpler. Gate-level pruning is also a technique that has been receiving attention lately, which focuses on pruning specific nodes from the netlist representation of the RTL designs based on removal metrics (CASTRO-GODíNEZ et al., 2021). Applying sub- or near-threshold syntheses may lead to more power-efficient kernels as well, inserting errors in the modules. However, sub- and near-threshold analyses are out of the scope of this thesis.

The enormous design space created by combining several approximation techniques to ML models imposes a problem for designers. This is because verifying how different approximations interact with each other in a hardware accelerator cannot be easily inferred, requiring RTL descriptions and syntheses to be performed for each scenario. This leads to an unfeasible work for describing the VLSI accelerators, limiting the number of possibilities tested to find a model efficient enough to the needs of a designer. This issue is amplified for tree-based models, which do not have as many tools available, as they have not received as much attention as NNs.

1.1 Thesis Proposal and Goals

Given the considerations made previously, the claim presented in this thesis is that design frameworks implemented to automate the generation of approximate computing VLSI architectures can better explore the design space to achieve low-power in tree-based ML models while incurring small or negligible losses of accuracy. In order to accomplish this claim, the following goals were set:

- Present a literature review on the foundations and current state-of-the-art in low-power tree-based machine learning VLSI hardware architectures and approximate computing to determine the most suitable techniques to be employed;
- Propose frameworks for automatic generation of tree-based machine learning VLSI hardware architectures to ease the design space exploration and analyses of trade-offs;
- Evaluate and explore the most suitable approximate techniques to be included in the machine learning models generated in the frameworks;
- Provide sets of VLSI accelerators that can achieve efficient results in terms of power-, energy-, and area-accuracy trade-offs, considering ASIC and FPGA platforms and targeting applications related to embedded systems.

Several explorations have been made to support the thesis claim. The first focus was developing a baseline framework to design tree-based models, given that these are simple and popular ML methods. The framework has been developed to generate DT and RF VLSI architectures, allowing the variation of several design parameters. This framework enables the analysis of power-, energy- and area-accuracy trade-offs. This framework was extended to include approximation methods (i) to quantize the input features, (ii) to approximate the comparators of the tree (in both regular standard-cell and MAJ-based DTs/RFs), and (iii) to apply both indiscriminate and hierarchical gate-level pruning in the netlist representations. As a case study, data sets based on wearable devices have been analyzed, but the trade-off analysis to search for the best models can be easily adapted for any application by using the developed tools. Additional explorations have been performed during the Ph.D. thesis development period that also use the baseline framework, focusing on hardware security for error-tolerant circuits, but they are not presented here, given that they are out of the scope of the thesis claims. Even so, this shows the extensive usefulness of the developed framework even when power dissipation is not the focus.

The contributions brought by this research work include:

- a set of frameworks to generate VLSI accelerators of tree-based inference models in a fully parallel manner from raw data sets, including preprocessing, training, translation, and synthesis stages, employing (i) quantization, (ii) approximations in the comparators, and (iii) indiscriminate and hierarchical gate-level pruning;
- a dedicated and optimized translation method of the tree models to HDL descrip-

tion, which provides for significantly reduced power dissipation in the VLSI is significantly reduced;

- evaluation in terms of power-, energy-, and area-accuracy trade-offs of DT and RF models of different complexities, with varying tree depth, number of trees, input quantization, approximation of comparators, and gate-level pruning;
- a framework that performs approximations in tree-based accelerators applied to MAJ-based circuits, focusing on emerging technologies.

Possible future work can opt to analyze other types of tree-based models, such as XGBoost (CHEN; GUESTRIN, 2016). Alternatively, one can explore other ways of implementing these DT/RF architectures, e.g., using a fixed kernel sequentially calculating the inference through a fixed number of comparators.

1.2 Thesis Organization

This report is split into the DT/RF steps that involve power analyses and additional proposals related to tree-based models. This document is structured as follows: Chapter 2 presents a detailed background on the main concepts regarding ML; Chapter 3 presents a background regarding VLSI design, focusing on power dissipation throughout the abstraction layers, and approximate computing, detailing the main possibilities of hardware approximations along the design stack; Chapter 4 details a review of state-of-the-art works found in the literature that address similar topics regarding ML models (focusing on hardware architectures for DTs and RFs), or generic approximate computing techniques; Chapter 5 describes the proposals, the frameworks developed, the generation of the architectures, and the approximation techniques employed, for both regular standard-cell and MAJ-based circuits; Chapter 6 presents the power, energy, timing, area, and accuracy results of the architectures implemented; Chapter 7 concludes the thesis manuscript, by summarizing the contributions of the work, and highlighting the main future paths that can be addressed.

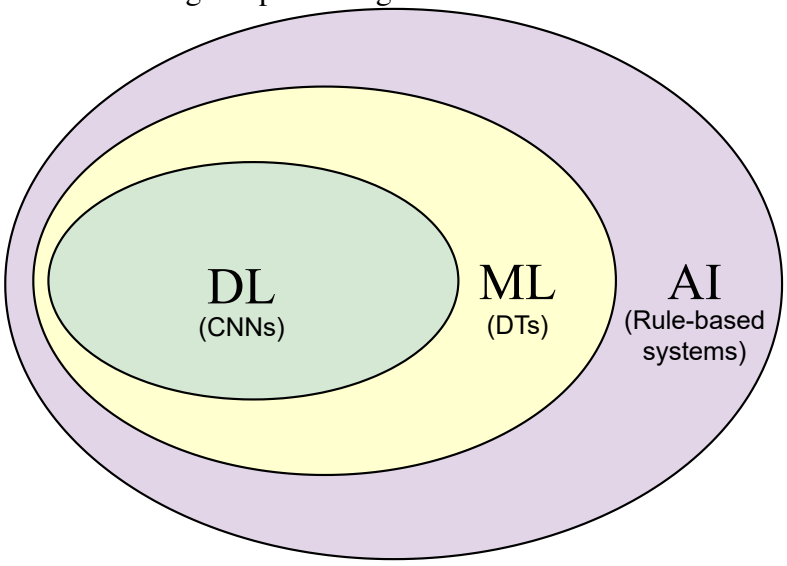
2 MACHINE LEARNING

Since the early days of computer science, researchers have wondered whether machines would be able to autonomously think (TURING, 1950), i.e., have intelligence and perform tasks that could mimic human behavior. This approach is based on software that could automate routine tasks, identify objects in images, understand speeches, perform medical diagnoses, etc. (GOODFELLOW; BENGIO; COURVILLE, 2016). This concept is denoted as Artificial Intelligence (AI). Even though the beginning of the field dates way back from the 1950s, employing AI has only become significantly popular nowadays, given that semiconductor technology has evolved to a point where some proposals are now feasible to be implemented to a certain degree.

Even though some applications that use AI are trivial to be executed by computers, the major challenge is to attempt to solve tasks that the human mind could intuitively solve but are not easily described as an algorithm (GOODFELLOW; BENGIO; COURVILLE, 2016). Therefore, achieving this intuition with AI would require additional research and evolution of the technology, which was not yet possible when the concept was conceived.

A particular field of AI called machine learning (ML) has developed particularly fast in the past few years. Figure 2.1 illustrates where ML stands within the AI coverage.

Figure 2.1: Venn Diagram presenting where ML stands w.r.t. the AI concept.



AI can be best characterized nowadays as an umbrella term that encompasses ML and many other approaches like heuristic search and genetic algorithms. In specific, rule-based systems are AI applications that are outside the ML scope. DTs are classical examples of ML systems and are one of the main focuses of this thesis within the broad

AI field. DL models are mostly represented by CNNs and other types of NNs.

It is convenient to define the term *learning* for clarification purposes. According to (MITCHELL, 1997), "A computer program is said to learn from experience E with respect to some class of tasks T and performance measures P, if its performance at tasks in T, as measured by P, improves with experience E".

The task T denotes the application that is being targeted, e.g., object detection. Many tasks are referred to as classification, as they aim to label inputs into specific categories (classes). A well-known classification problem is the Breast Cancer Wisconsin (STREET; WOLBERG; MANGASARIAN, 1993), which attempts to predict, based on ten different features, whether a patient has benign or malignant breast cancer. Regression tasks are another popular category in ML. Regression problems aim at finding a numerical output, such as when predicting prices of houses given parameters such as size, location, etc. Structured output, transcription, and clustering are also important tasks within the ML field.

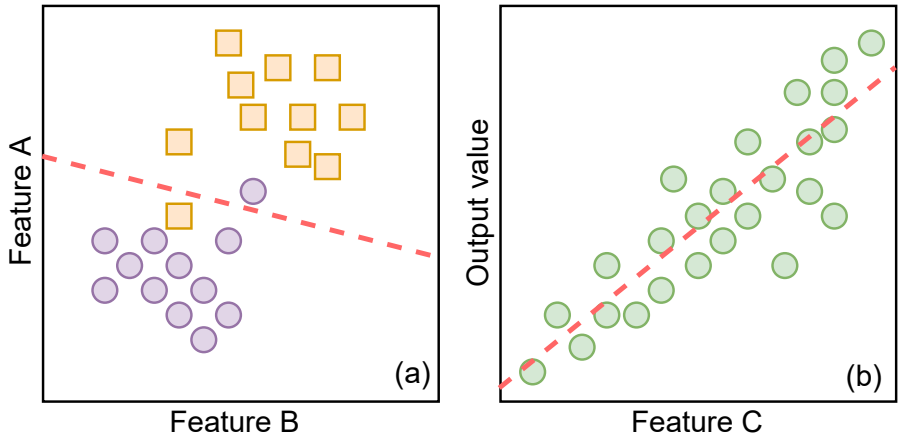
The experience E, with the learning process, can improve the performance P of the algorithm. Experience can be obtained through different methods and is represented by the data employed. Depending on the case, ML can be split into different categories, which are presented as follows:

- **Supervised Learning**: this is the most popular approach. It is based on learning through a set of features extracted from data, and each example of this feature set includes an associated output label (AGGARWAL, 2015). These features can be seen as characteristics of the input data that may be relevant to define the desired output, either for classification or regression problems. Usually, for these problems, part of the data set is used to train the model. With the training set, the ML model learns the patterns and similarities with the data being used for training, along with the knowledge of what class/value it is supposed to be. With that information, the ML model is adjusted to be able to learn how to find out the class – or value, for regression – when unseen sets of data come into play. Examples of supervised learning problems are shown in Figure 2.2, in which a binary classification and a regression problem are presented, considering specific features in each of them. In the classification problem of Figure 2.2-a, two features are considered, and the colors of each point represent their respective output label, provided by the data set, and the red dotted line represents the solution found by the ML model to separate the two classes. In the case of Figure 2.2-b, a regression problem with one feature

is presented, with its respective output value, which is a continuous value. In this example, the red dotted line represents the function that the ML model represents to infer the output values when different values for the feature are presented.

- **Unsupervised Learning**: this is based on attempting to learn similar patterns in data sets that have no associated output labels. This is mainly used for: (i) clustering purposes, to split data that can be similar to each other, allowing for categorization, or (ii) anomaly detection, which is based on detecting rare events from a set of inputs with little information about them. Unsupervised problems are beyond the scope of this work; therefore, a more detailed background is not presented in this report.
- **Reinforcement Learning**: this type of learning is based on concepts of trial and error. The ML algorithm using reinforcement learning usually performs operations and learns from its previous actions, based on scoring, e.g., rewardings and punishments. Popular examples of reinforcement learning are intelligent chess AI machines that interact with the chessboard and attempt to calculate the best actions to be performed based on a range of possibilities (SILVER et al., 2018). Reinforcement learning is also out of the scope of this thesis.

Figure 2.2: Example of (a) classification and (b) regression problems for supervised learning methods.



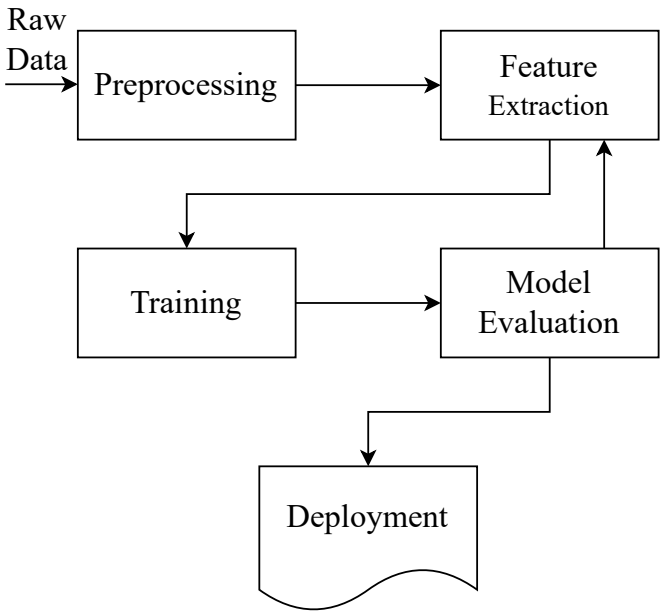
Lastly, the performance P is represented by a metric that can be used to evaluate the efficiency of the model. It is important to evaluate the ML model on unseen data (i.e., combinations of features that were not considered in the training process) to verify whether the model has generalized well enough. It is convenient to have a test set available whose data has not been used by the model prior to evaluation. The performance of the model is usually defined by the accuracy, indicating the proportion of correct outputs.

Several methods for measuring accuracy can be used, and their definitions highly depend on the type of task being addressed. For simple classification tasks, for instance, the accuracy is defined based on the number of correct classifications, determined by the equivalence of the model output and the label defined in the test set.

2.1 Machine Learning Flow

In general, the use of ML follows the flow presented in Figure 2.3. Each of the steps involved in this process is described below. This explanation focuses on supervised learning problems, specifically for classification ones, as they are the focus of this thesis.

Figure 2.3: Generic flow applied to ML problems.



Systems employing ML algorithms usually need to be fed with a considerable amount of input data. The data which feeds ML algorithms usually come in a raw format. Depending on the model employed, this data is not usually ready to be used as input for the algorithm. Such data can be generated from sensors, for instance, which read values from the environment in which they are inserted.

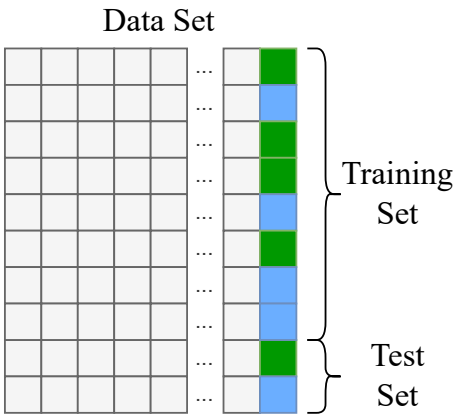
Thus, the raw data must be used in a preprocessing step to convert it to useful information for the context in which it is being used, depending on the ML model, specificities of the framework being employed, etc. In this step, data remapping or quantization can also be applied, depending on the limitations of the system.

The preprocessed data is utilized in a *feature extraction* process. This process can usually be performed by manually extracting the features based on the own intuition of

the designer of the data set or through features that are easily extractable. Features can also be combined to create others, which may lead to better results in the model evaluation step.

Before being used in the training process, the data can be split into training and test sets, as mentioned. This is known as the hold-out method (AGGARWAL, 2015). Usually, popular rules of thumb to be used when splitting data into train and test sets are 66.6%-33.3%, 80%-20%, and 90%-10%. It is up to the designer to be able to check which split works best for a specific problem. A simple example for a data set of a supervised problem with 10 samples is shown in Figure 2.4, where each line corresponds to one sample, each gray square in a line denotes a different feature, and the output classes are denoted in blue and green. In this case, the split chosen is 80%-20%.

Figure 2.4: Use of the hold-out method for a simple data set with 10 samples, and two output classes.



Alternatively, a cross-validation split can also be used. In this method, the data is split into k different groups. Then, $k-1$ of these split sets are used to train the model, and the remaining group is used as a test set. This process occurs until every group has been used as a test set.

The split of the data itself is also very important. Considering a classification problem, if we split the data into two sets and the training set only contains data whose output label indicates one specific class, the model will not learn when to output a different one. Therefore, performing a split that has adequate proportions of the classifications provided in the entire data set is important to guarantee greater robustness to the algorithm. This method is referred to as stratified sampling (AGGARWAL, 2015), and is also exemplified in Figure 2.4, given that both training and test sets contain half the samples for each output class.

ML algorithms receive this set of features to perform the training process. The

training stage only uses the training set and self-adjusts – by varying its own parameters, considering limitations imposed by the programmer – finding relations between the different features, leading to a final trained model. It is desired that this model predicts the correct outputs when sets of unseen data come into play.

The model is then evaluated using a performance metric, depending on the model being employed. This evaluation is performed on the test set, given that it contains data that was unseen for the model. For simple classification problems, for instance, we can employ the 0/1 loss function (RUSSELL; NORVIG, 2009), as in Equation 2.1, where y_i and $\hat{y_i}$ denote the predicted label and the correct one from the data set. Based on this function, the accuracy can be defined as in Equation 2.2, in which n is the number of samples of the test set.

$$L(\hat{y_i}, y_i) = \begin{cases} 1, & \text{if } \hat{y_i} = y_i \\ 0, & \hat{y_i} \neq y_i \end{cases} \tag{2.1}$$

$$Acc = \frac{1}{n} \cdot \sum_{i=1}^{n} L(\hat{y_i}, y_i) \tag{2.2}$$

Other important metrics for accuracy can be considered, such as precision and recall (TING, 2010). Precision is defined as the proportion of true positives (predictions that the ML model correctly classified) w.r.t. the total number of positives predicted by the model. On the other hand, recall denotes the proportion of true positives w.r.t. the number of instances that were supposed to be classified as positives. These two metrics are defined in Equations 2.3 and 2.4, where T_p, F_p and T_n denote the true positives, false positives, and true negatives, respectively.

$$Precision = \frac{T_p}{T_p + F_p} \tag{2.3}$$

$$Recall = \frac{T_p}{T_p + F_n} \tag{2.4}$$

Both precision and recall are out of the scope of this thesis, given that this would increase the number of parameters when searching for better configurations.

It can also be convenient to apply model evaluation using the training data to verify how the model performs on already seen data. Some problems arise when the algorithm presents high accuracy with the training set but performs poorly on new data. This is usually called overfitting. Alternatively, if the algorithm presents low accuracy on the

training data, we have a case of underfitting.

After evaluating the model to verify whether it is acceptable for the accuracy levels envisioned, the ML model can be deployed for further use on unseen data. Alternatively, the algorithm can opt to perform feature extraction and training once again.

2.2 Machine Learning Techniques

This section addresses well-known ML techniques, which are employed in this work to achieve the claims of the thesis. The focus is given to tree-based hardware architectures, since they are simple and hardware-friendly models.

2.2.1 Tree-based Methods

Decision Trees

A DT is a popular supervised learning method employed in both classification and regression problems. They are formed by using features of a training set to obtain a model that is made of comparisons, i.e., true or false decisions, between a feature and a constant (defined in the training process). These nodes are denoted as decision nodes. DTs classify the instances by sorting them from a root node to several leaf nodes (MITCHELL, 1997). The goal of a DT is to correctly predict the outputs, defined by the leaf nodes through the training process as well, when samples that may not have been tested yet are given as inputs to the DT.

Figure 2.5-a illustrates a DT for a classification problem with 4 classes. Here, the decision nodes are the $\leq$ (less than or equal to) comparisons, which always compare a feature (indicated by $feature_X$) with a constant (C_X), and the leaf nodes, indicated by the ellipses, represent the predicted class.

One of the main advantages of DTs is that they are more interpretable than other models, such as NNs. Therefore, the model can be easily validated by a designer.

The training step for generating a DT can be performed in many different ways. A popular algorithm used for generating DTs is the Classification and Regression Tree (CART) (BREIMAN et al., 1984). CART works by splitting the train data into different branches based on a measurement function. In order to determine the best class to be split, DT implementations usually consider the Gini Impurity metric, which defines the

probability of classifying a new random data incorrectly based on the existing distribution of data in the training set. The Gini impurity is calculated as in Equation 2.5, where N is the number of different classes and $p(i)$ refers to the probability of an element being from class i.

$$G = \sum_{i=1}^{N} p(i) \cdot (1 - p(i)) \tag{2.5}$$

After defining the best feature to split data, CART performs the split, and the Gini Impurity metric is applied once again for each of the features in the newly split data. The algorithm may halt – stop splitting – when all the data samples have the same target value or the samples in the current node are constant. Moreover, the split can be halted due to limitations on the tree depth or other additional heuristics considered.

In DTs, the training stage is performed offline. Therefore, for this thesis, a software-hardware approach is considered, in which the software is responsible for the training stage (performed in a GPP running the algorithm), and the inference stage is executed by the dedicated hardware accelerators generated in this thesis. This is employed given that there is no apparent need for the training stage to be performed in a power-efficient manner if we consider that it can be performed prior to the execution.

In some problems, it may be convenient to design architectures that are fully dedicated to a specific training process, with the comparisons explicitly programmed. This is due to the fact that some applications will be permanently embedded in a device. On the other hand, some other applications may require a DT model changing with time. In this case, we require an architecture model that can adapt to the new sets of comparisons being performed.

Hardware architectures for DTs can be employed using comparators, along with 'AND' and 'OR' chains (or multiplexers), as it will be shown in Section 5.1. This thesis employs DTs for classification problems only. More specifically, multi-classification target variables are considered.

Random Forests

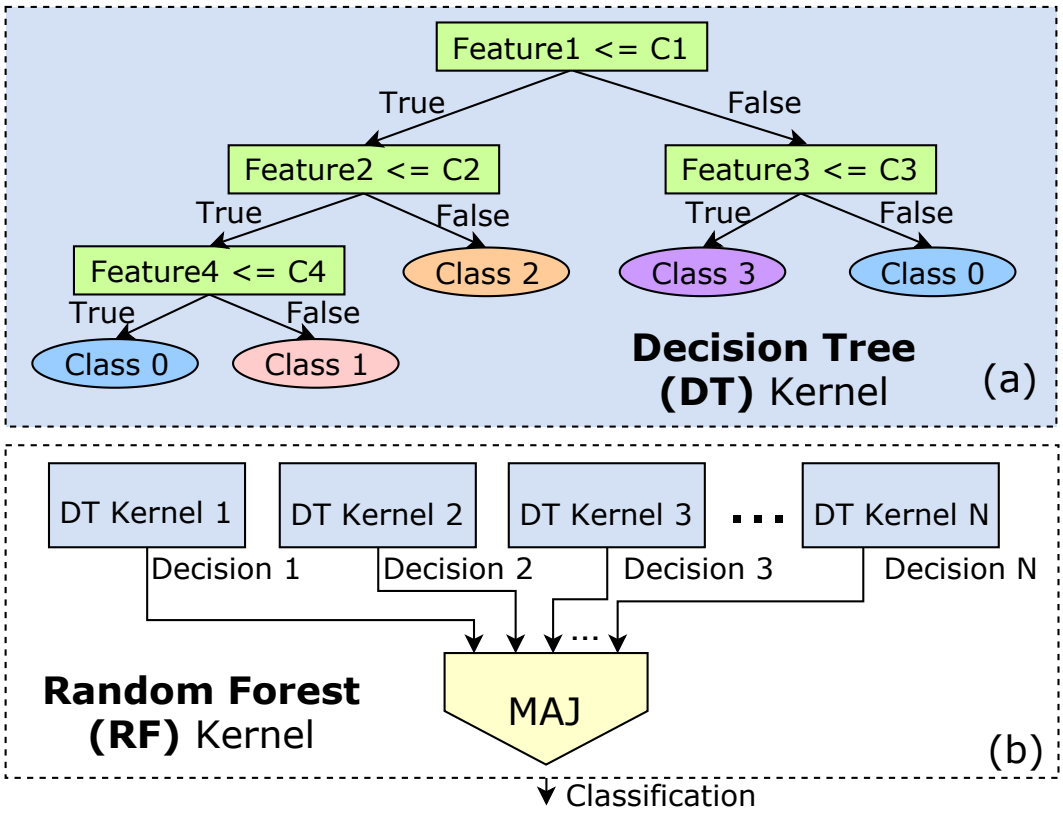
Random Forests (RF) are classifiers that consist of a large number of trees operating simultaneously (BREIMAN, 2001). RFs follow the assumption that a group of experts (or models) making decisions will lead to better results on average compared with a single tree. Given that each tree has its prediction, a voting method (e.g., majority voting) must

be employed to define the final RF decision.

The training stage in a RF is done by randomly sampling a subset of the input features used to train each tree. The number of features can be adjusted during training.

When designing circuits for RFs, in addition to all the DTs in the forest, the voter that makes the final decision also needs to be considered. One simple and efficient solution is to use a majority voter, as depicted in Figure 2.5-b, but more complex alternatives using weighted votes can also be considered (at the cost of more hardware resources).

Figure 2.5: Tree-based classifiers: (a) Decision Tree and (b) Random Forest.



2.3 Chapter Summary

This chapter presented a detailed background of the main concepts related to AI, focusing mostly on the ones that are related to this work the most. Therefore, concepts related to ML models such as DTs and RFs were presented.

Such concepts will be required for a better understanding of the proposals in Chapter 5.

3 VLSI DESIGN

3.1 Power Dissipation in CMOS Integrated Circuits

Given that this thesis focuses on obtaining combinations between model configurations and approximate techniques with the best power dissipation vs. accuracy trade-offs (besides area and energy reductions wrought by AxC), it is essential to provide a basic background on power dissipation in CMOS circuits.

There are two main sources for power dissipation in CMOS circuits:

- **Static Power** (P_{static}): Dissipation due to leakage currents, even when the circuit is in an idle state, with all circuit inputs at DC levels. Leakage currents in CMOS technology are due to three factors: MOSFET transistors subthreshold currents, reversely biased diodes off-state currents, and DC gate-to-channel or gate-to-drain tunneling currents through ultra-thin (less than 2 nm) dielectrics (RABAEY, 1996). Static power is given by Equation 3.1, where I_{static} is the total static current drawn from the circuit DC supply when adding all the DC currents mentioned above.

$$P_{static} = I_{static} \cdot V_{dd} \tag{3.1}$$

- **Dynamic Power** ($P_{dynamic}$): This dissipation is a consequence of the switching activity in a logic gate. Dynamic power in CMOS is composed of two portions: (i) $P_{switching}$, which is the power dissipated due to the capacitive loads at the output of the gate, i.e., charging the load when logic state transitions from '0' to '1' and discharging the load when a transition occurs from '1' to '0'; and (ii) P_{SC}, which occurs when a direct current flows through a path from the voltage supply to ground in a CMOS network. P_{SC} occurs when a static CMOS circuit is switched on by an input signal with rising and fall times different from zero, which leads to the PMOS and NMOS transistors of both networks being active simultaneously for a short period of time. This situation leads to the direct current flowing directly to ground. Hence, $P_{dynamic}$ is given by Equation 3.2.

$$P_{dynamic} = P_{switching} + P_{SC} \tag{3.2}$$

$P_{switching}$ is presented in Equation 3.3, considering a load capacitance C_L that will be charged or discharged based on a change in the gate output. A denotes the output node

activity, measured in events/second, and V_{dd} is the input source voltage. It may be more convenient to use a probability value α, which represents the node activity factor, along with the operating frequency f, given that most of the circuit nodes will not change at every clock cycle.

$$P_{switching} = C_L \cdot A \cdot V_{dd}^2 = \alpha \cdot f \cdot C_L \cdot V_{dd}^2 \tag{3.3}$$

Most tools for measuring power consider α as a fixed value. This is employed as a methodology for power extraction in many works, which does not lead to accurate results, given that the input variations depend on the application for which the developed architecture will be used.

Short-circuit power can be expressed by Equation 3.4, where I_{SC} is the short-circuit current, and β is a factor related to the input signal transition time to the cell peak short-circuit current.

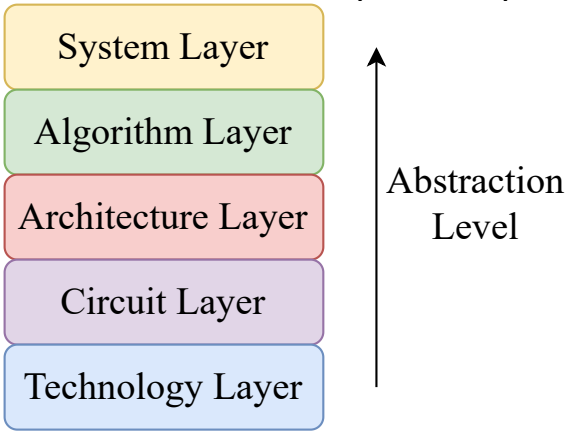
$$P_{SC} \approx V_{DD} \cdot I_{SC} \cdot \frac{\tau_{in}}{4} \cdot 2f \approx V_{DD}^2 \cdot f \cdot \frac{C_L}{10} \tag{3.4}$$

Taking the dynamic power $P_{dynamic}$ – which is the sum of $P_{switching}$ and P_{SC} – and static power P_{static} into account, we can define the total power as in Equation 3.5.

$$P_{total} = P_{dyn} + P_{static} = P_{switching} + P_{SC} + P_{static} \tag{3.5}$$

As presented by (RABAEY; PEDRAM, 1996), power dissipation can be mitigated by applying different design techniques at several abstraction levels, as shown in Figure 3.1.

Figure 3.1: Design abstraction levels at which power dissipation can be optimized



Source: Adapted from (RABAEY; PEDRAM, 1996)

These specific abstraction levels are described below.

- **System level**: power dissipation can be mitigated at this level through techniques such as partitioning and power states. Moreover, system-level approaches can benefit from techniques of frequency- and voltage-scaling, which will lead to a decrease in the power dissipation based on the equations presented above, and power gating, which tends to turn some gates off during the execution. In particular, power gating has been increasingly applied due to the increase in the power density, which led to the Dark Silicon era (as aforementioned in Chapter 1).
- **Algorithm level**: at this level, power dissipation can be decreased through optimizations in language, compiler (through optimization flags), concurrency, regularity, locality, etc, when a software is running on an embedded system, for example.
- **Architectural level**: this consists of reducing power by changing the architectural aspect, which can include: (i) paralellizing hardware functions to work at a decreased frequency; (ii) applying clock/power-gating to some units at a lower level w.r.t. the system one, so as to mitigate the dissipation occurring from switching in unused hardware modules; (iii) schemes for low-power data encoding.
- **Circuit level**: at this lower level, blocks can be optimized to focus on power, such as arithmetic operators. Approximate computing can also be applied here, by implementing efficient hardware, as long as the application is not compromised.
- **Technology level**: among the possibilities at this level, it denotes optimizations coming from the technology scaling, which leads to differences in the models for power dissipation, the technology employed itself, or from physical/device aspects that can be enhanced to change aspects of the transistors, such as the change in the threshold voltage through a change in doping concentration.

The scaling with respect to the technology node affects power dissipation in different ways. Given that the capacitances decrease due to smaller transistor sizes, there is a consequent decrease in dynamic power dissipation. However, this reduction in the capacitance can be accompanied by an increase in the operating frequency, which can offset the capacitance reduction. Furthermore, static power dissipation tends to increase due to factors such as sub-threshold leakage, which becomes more significant when the node is smaller. Power density also tends to increase given that more transistors can fit in the same area w.r.t. larger technology nodes. Therefore, in general, power dissipation scales in a complex manner, involving trade-offs between several factors.

3.2 Approximate Computing

Approximate computing has emerged as a promising design alternative that trades exactness for reductions in the costs of either hardware or software (STANLEY-MARBELL et al., 2020). Regarding hardware costs, which are the focus of this thesis, approximate computing can bring significant reductions in VLSI circuit area, timing (reductions in the critical path delay), and power dissipation, hence reducing the energy consumption (HAN; ORSHANSKY, 2013; STANLEY-MARBELL et al., 2020). This is done by leveraging the intrinsic error resilience of several applications, such as image processing (PAIM et al., 2019; SOARES et al., 2019), video coding (PAIM et al., 2020), healthcare consumer electronics (SEIDEL et al., 2021), and ML (ZERVAKIS et al., 2021; TASOULAS et al., 2020).

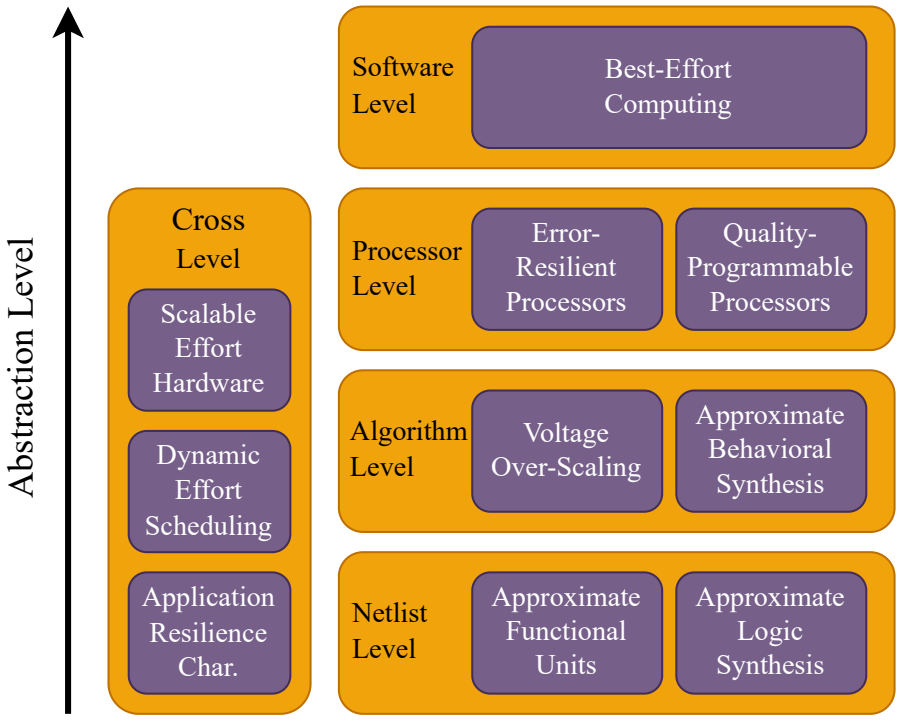
Approximate computing is very popular in image and video compression, given that the subjective characteristic of the human eye to the video content makes it intuitive to apply approximate computing techniques. In that specific scenario, it is mostly a requirement, given that a video without any kind of compression is usually unfeasible to be transmitted or stored. The use of approximate computing techniques in ML models is also really convenient due to their probabilistic nature.

The use of approximation is even more important when considering embedded devices, e.g., smartphones and wearables. Given that the evolution of ML and DL algorithms has led to more complex computations, and since they have been employed in these battery-powered devices even more due to the tendency of performing computing at the extreme edge, using approximate techniques is of utmost importance to alleviate the effects that this increase in computation would have in the battery life of the device. Therefore, designing approximate computing ML solutions has attracted significant research interest (ZERVAKIS et al., 2021), and it has been applied in ML for signal (BUSCHJÄGER; MORIK, 2018; PASHAEIFAR et al., 2019), image and video processing (GRELLERT et al., 2019), and healthcare (CHANG et al., 2019), to name a few.

Several techniques can be used to achieve approximation, and they can be applied in most levels of the design stack, from the software all the way to the netlist level. This classification is based on the taxonomy from (PAGLIARI; PONCINO; MACII, 2016), which is shown in Figure 3.2.

- **Netlist Level**: this level refers to transistor or gate-level approximations. Two main techniques are found in the literature: (i) the use of approximate functional units,

Figure 3.2: Approximate computing (AxC) techniques at different design abstraction levels.



Source: Adapted from (PAGLIARI; PONCINO; MACII, 2016)

such as approximate adders, multipliers, MAC operations, etc (GARG; PATEL; DUTT, 2020; KIM et al., 2019; ZENDEGANI et al., 2017), and (ii) approximate logic synthesis concepts, which is a more generic approximation regarding functional units, but now applied to any gate-level circuit (PAGLIARI; PONCINO; MACII, 2016). Within this context, gate-level pruning is a technique applied to the netlist descriptions, which is generated after the logic synthesis steps from the Register Transfer Level (RTL) model. The netlist contains the logic elements from a standard-cells library along with a file describing the switching activity of each gate (CASTRO-GODíNEZ et al., 2021). This technique allows for pruning the nodes based on some metrics regarding the switching activity. The use of pruning in the netlist level may lead to approximations that the other levels cannot achieve, given that the designer usually does not have any control of what the synthesis tool generated from the RTL descriptions. This technique is employed in this thesis. Alternatively, approximations regarding sub/near-threshold computing can also be applied at this level (SCHLACHTER; CAMUS; ENZ, 2015). This technique is related to transistors working at voltages closer to the threshold voltage V_{th}, in which the circuit is not fully turned on. The reduction in power and energy leads to a dete-

rioration of the robustness of the circuit against process/voltage/temperature (PVT) variations. However, sub/near-threshold computing is out of the scope of this work.

- **Algorithm Level**: according to (PAGLIARI; PONCINO; MACII, 2016), this level also includes architecture-level techniques, as there is not a well-defined distinction between this level and the netlist level approximations. Voltage Over-Scaling can be included at this level, as well as approximate behavioral synthesis. The latter can include approximations in the RTL description, such as truncation of LSBs, substitution of variable operators to constant ones, etc. This is also briefly employed in the thesis through the use of comparators with the constants explicitly described, leading to less costly hardwares w.r.t. generic comparators.
- **Processor Level**: this is mostly based on accepting errors due to temperature, process variability, radiation, etc, by not using reliability methods in the entire processor. This leaves some portions without any sort of protection, leading to decreased area and power.
- **Software Level**: this type of approximation refers to optimizations that can be exploited by an operating system by relaxing correctness requirements, whether by skipping computations due to limited budgets, or by relaxing synchronization between threads to improve performance.
- **Cross Level**: they usually rely on the effects of combining approximation techniques at different abstraction levels to obtain power and area savings. Dynamic scheduling can also be applied, to change the approximation levels in different layers at runtime, based on budget requirements. Alternatively, some of the techniques in this thesis are performed as cross level. For example, as it will be seen, remapping and quantization were performed at a higher level before generating the RTL descriptions, as an attempt to decrease hardware resources. The same will be seen from the contribution regarding constant parameter approximation, as they were adjusted expecting to achieve reductions in the underlying hardware in which they will be implemented.

3.3 Chapter Summary

This chapter aimed at providing an overview of concepts related to power dissipation in CMOS circuits, presenting possibilities of mitigating power in several abstraction

levels. It also presented a background on approximate computing, which is important to understand the proposals and results obtained in this thesis. This background used the taxonomy from (PAGLIARI; PONCINO; MACII, 2016), and detailed the most well-known approximate computing techniques throughout the design stack. More details are given to some specific levels since they presented techniques related to the proposals of this thesis.

4 RELATED WORK

For this thesis, extensive research has been performed to find related proposals in the literature. This chapter will present and describe works regarding implemented VLSI accelerators and analyses of ML models. The focus is given to tree-based methods, given that they are simple, hardware-friendly, and popular models. More attention is given to works that provide area, power dissipation, or energy consumption results, which are the main focuses of this thesis.

4.1 Tree-based Models

The ML research in this thesis is mainly represented here by DTs, RFs, or other tree-based methods. To compare the work developed in this thesis with other implementations of DTs and RFs, research on the literature had to be made. The comparisons made in part of Chapter 6 are performed using the most similar works found in this section.

Most related works that were found in the literature that evaluate power dissipation and energy consumption focus on NNs, due to their higher popularity and good accuracy results for more complex problems such as face recognition. Despite that, there are works that also make explorations on other tools such as DTs or RFs. Considering the reduced circuit requirements of DTs/RFs, these methods are usually more suitable for low-power environments, and it is important to have works performing analyses on these.

Yang et al. (YANG; BOLING; MASON, 2014) propose a DT-based spike classification system for neural recording implants. The proposed architecture dissipates 32nW power per channel, using a clock frequency of 50kHz, considering eight channels. However, apparently, the data set used in their paper is no longer available for download and, therefore, the work is not reproducible.

Buschjäger et al. (BUSCHJÄGER; MORIK, 2018) investigate implementations of DT/RF considering models for Von-Neumann architectures and FPGAs. Power results are presented for both an FPGA – which are compared to the work of this thesis in Section 6.3 – and an ARM processor for several tree models. Although the work performs comparisons between different models of DTs/RFs and presents power results, no data for VLSI ASIC is presented, the input data switching activity for their data set is not considered, and no model complexity variation (tree depth or number of trees) or approximation techniques are applied. Additionally, even though the work performs comparisons between

different models of DTs/RFs and presents power results, High-Level Synthesis (HLS) was used to obtain the HDL architectures directly from the algorithm description, which is not suitable to fulfill power-aware design requirements. Even so, the work was reproduced as much as possible and is compared to the DTs/RFs from the proposed framework.

Kang et al. (KANG et al., 2018) implement an integrated circuit of a RF classifier focused on traffic sign recognition and face recognition data sets. The chip presents power results of 7.1mW. Despite the fact that this work only implements single models for the RF classifiers (no analysis on the trade-offs for several parameters is performed), the work was reproduced and is compared to the proposal in this thesis.

Tong et al. (TONG; QU; PRASANNA, 2017) present algorithms and architectures using DTs for online traffic classification. However, this work is not focused on power-efficient hardware architectures. Mitsunari et al. (MITSUNARI et al., 2018) propose an architecture for a decision tree ensemble focused on object detection. Also, a task scheduling algorithm is proposed in order to control the memory accesses to improve the performance of the architecture. Even though the work considers the decision and leaf nodes of the DTs, no approximations are performed. Wang et al. (WANG et al., 2019) propose the use of deep neural DTs. These models take advantage of the characteristic of automatic feature extraction of NNs and employ the obtained features in DT modules. The works from Tong et al. (TONG; QU; PRASANNA, 2017), Mitsunari et al. (MITSUNARI et al., 2018), and Wang et al. (WANG et al., 2019), do not provide any power reports.

Chang et al. (CHANG et al., 2019) presents a processor focused on automatic sleep staging, using a NN-based DT for classification. The chip was prototyped using a 180nm CMOS process and dissipates 4.96μW of power. This work presents an interesting co-design approach to optimize the algorithm and architecture of the model.

Shen et al. (SHEN et al., 2012) present an architecture that combines depth- and breadth-first searches in a DT to reach the shortest path with reduced space. The architecture was synthesized to a 65nm CMOS standard cells library at 333MHz and achieved a power consumption of 7.26mW. However, the architecture includes modules other than the DT; hence, it is unfeasible to estimate the power dissipation of the DT to make a fair comparison.

Ignatov et al. (IGNATOV; IGNATOV, 2017) proposes the use of decision streams, which are more efficient manners of assembling a DT with better generalization. This is given the fact that DTs have similar distributions in lower layers closer to the leaf nodes. Hence, the tree becomes an acyclic graph, merging nodes that are statistically

indistinguishable.

Zhao et al. (ZHAO; SUN; CHEN, 2018) proposes a discretization method for floating-point values on DTs. The proposal is based on the fact that the memory requirements for floating-point values are high; therefore, discretization, in this context, is essential.

Kuehn and Manoli (KUEHN; MANOLI, 2018) propose an integration between FPGA and ASIC to implement a DT model, so that ASIC can implement the parallel architecture, whereas the FPGA can keep updating the model whenever it needs, generating a reconfigurable architecture. However, no results for architectures are presented.

Baumgartner et al. (BAUMGARTNER; HUEMER; LUNGLMAYR, 2022) propose an efficient majority voter and applies it to a RF as a case study, for the MNIST data set. The proposal achieves significantly better results in terms of classifications per second when compared to other state-of-the-art accelerators, despite presenting decrease accuracy given that the related work used SVMs and NNs. However, the results considering the entire RF module using the proposed majority employs pipeline techniques, and the main focus here is on obtaining increased throughput, and not on finding good trade-offs between power and accuracy.

Bartels et al. (BARTELS et al., 2021) present a setup with hardware and software for training and inference for classifying cow behavior using a DT. An 8-bit quantization is applied in the features, and a power of 216μW was obtained. Besides the DT module, a feature extraction stage is implemented in hardware as well, making it difficult to estimate the power of the DT. Moreover, the work does not perform any exploration on different models, no information is provided on the exact algorithm used for training, and on the tree depth considered. Also, the exact data set could not be obtained.

Ajirlou et al. (AJIRLOU; PARTIN-VAISBAND, 2022) propose the use of a RF classifier to dynamically adjust the clock frequency of a MIPS processor, based on instruction information. The proposal consists on employing the accelerator as a pipeline stage of the MIPS implementation. The work presents some variations in tree depth and number of trees, but no approximate computing techniques are mentioned. Moreover, the power estimations and overheads considered the entire MIPS processor and all its stages (fetching, decoding, execution, memory, and write back), so estimating the cost of the analyzed RF accelerator is unfeasible.

Silva et al. (SILVA; GRELLERT; MEINHARDT, 2022) propose the use of approximate comparators in DT models. Proposals of comparators fully dedicated or based

on subtractors were analyzed and, for both models, approximate versions were proposed. Quantization was also performed in the data sets, and the training algorithm applied was the C5.0. However, it was not clear how the DT models were designed in hardware, or if the power/energy results are only a brief estimation based on the number of comparators. Also, no automatic generation of the models was presented, which makes it difficult to perform a broader evaluation. Even so, reductions in energy without much impact in the accuracy were obtained with some of the approximate models.

Shih et al. (SHIH; CHIU; WU, 2023) implement a DT with online training and inference in the same ASIC chip. The inference portion performs a technique of double-root DT, which leads to a speedup when compared to the approach denoted as Single Module Per Level (SMpL). The approach uses pipeline schemes to accelerate the process, taking the online training into consideration. The ASIC implementation considers a TSMC 40nm technology with multi-Vt. The power reported is 73.7mW, but this implementation considers both the online training and inference stages simultaneously operating.

Daghero et al. (DAGHERO et al., 2021) present an adaptive RF model, for a RISC-V microcontroller. The approach is based on dynamically deciding to early-stop the inference of the weaker trees that compose the RF when a high-enough classification confidence is obtained. In this case, the proposal can stop either based on thresholds of maximum energy or minimum accuracy. The work manages to achieve energy reductions of up to 90% while incurring in accuracy drops of about 0.5%, for the data sets analyzed.

Some works that focus on DTs or other similar tree-based models have been proposed focusing on wearable domains, such as (SONG et al., 2019; GAURA; KEMP; BRUSEY, 2013; KIM et al., 2020; MIAO et al., 2019; WANG et al., 2017). However, these works also do not present explicit power results.

Tree-based models have been employed in several other scenarios, which proves their usability even though NNs have been receiving increased attention. Some examples are the application of DTs for identifying performance bottlenecks in GPUs (ZHENG; HU; JIN, 2018), energy consumption forecasting (AL-GUNAID et al., 2016), color filtering (TUSOR et al., 2015), packet classification (KITAMURA et al., 2015), LTE and Wifi systems (CAI et al., 2016), etc.

Even though the DT/RF proposals of this thesis involve the use of generic frameworks that can operate in any data set, the tools were tested using case studies focused on wearable devices or similar embedded-related environments. Most of the analysis presented in this report involves the use of four data sets: activities, accelAdl, wearable,

and wireless. The papers focused on these four data sets are shown in Section 6.1, in which the characteristics of each of these data sets are described and detailed. Hence, this explanation of related works based on these data sets is not presented in this section.

Table 4.1 summarizes the characteristics of the related papers in the current literature that focus on DTs, RFs, or other tree-based methods, in the context of this thesis. None of the works listed propose any method of automatically generating tree-based hardware descriptions from software structures, which is one of the contributions of this thesis. Therefore, design space explorations (with the variation of approximate techniques, tree depth, number of trees, etc) cannot be easily made with the related works. Additionally, every work presenting power results does not mention the use of an accurate hardware switching activity estimated from the actual data set considered. This is important when we are focusing on power-aware accelerators, given that the switching activity of the data being fetched to the architecture significantly influences its power dissipation. Some works also employ training inputs not directly available and, although some works can be reproduced performing feature extraction, others cannot be reproduced due to lack of details in their papers. This is specifically detailed in the "Features Obtainable?" column. The feature extraction was only attempted for the works that presented power results, in order to perform hardware comparisons.

Table 4.1: Main characteristics of prior related works regarding DTs, RFs and other tree-based methods.

Work	Decision Tool	Dedicated Hardware	Automatic Generation	Approx. Techniques	ASIC	Power Results	Features Obtainable?
(SHEN et al., 2012)	DT	✓			✓	✓	
(YANG; BOLING; MASON, 2014)	DT	✓		✓	✓	✓	
(TONG; QU; PRASANNA, 2017)	DT	✓		✓			✓
(IGNATOV; IGNATOV, 2017)	Decision Stream/DT						
(SHARMA; SHEWANDAGN; BHUKYA, 2017)	DT	✓					N/A
(BUSCHJÄGER; MORIK, 2018)	DT/RF	✓	≈			✓	✓
(KANG et al., 2018)	RF	✓		✓	✓	✓	✓
(ZHAO; SUN; CHEN, 2018)	DT	✓		✓			✓
(KUEHN; MANOLI, 2018)	DT	✓			✓		N/A
(MITSUNARI et al., 2018)	DT	✓					N/A
(WANG et al., 2019)	Deep Neural DT						✓
(CHANG et al., 2019)	NN-based DT	✓			✓	✓	
(BARTELS et al., 2021)	DT	✓		✓		≈	N/A
(KANANI; VAIDYA; AGARWAL, 2021)	Gradient Boosting	✓	≈	≈		✓	✓
(DAGHERO et al., 2021)	RF			✓		✓	✓
(BAUMGARTNER; HUEMER; LUNGLMAYR, 2022)	RF	✓				≈	✓
(AJIRLOU; PARTIN-VAISBAND, 2022)	RF	✓			✓	≈	N/A
(SILVA; GRELLERT; MEINHARDT, 2022)	DT	✓		✓		✓	✓
(KARN; ELFADEL, 2022)	DT	✓	✓			✓	✓
(CHOUDHURY; AHAMED; GUHA, 2023)	DT (learning)	✓				≈	✓
(SHIH; CHIU; WU, 2023)	DT	✓		✓	✓	≈	✓
(SLIMANI et al., 2023)	RF			≈			✓
This Thesis	DT/RF	✓	✓	✓	✓	✓	✓

4.2 Chapter Summary

This chapter presented the most relevant works found during the literature research performed for this thesis, based on the goals that were set. This research was performed to verify where the state-of-the-art is, in several aspects, regarding VLSI design for ML (focusing on tree-based models) models. Even though several works found are focused on the models used in this thesis, i.e. DTs and RFs, not all of them can be replicated to perform fair comparisons, given that some data sets are unavailable or some details regarding the hyperparameters are not presented. Some works that include approximate computing solutions for tree-based models (mainly DTs and RFs) were also researched. Mostly, the approximations found consist in applying quantization to the data, which is the first technique employed in the proposals, as it will be seen in Chapter 5.

5 PROPOSED DESIGN TECHNIQUES FOR AUTOMATED EXPLORATIONS ON TREE-BASED VLSI ARCHITECTURES

5.1 A Framework for Designing Tree-based RTL Models

The first part of this thesis aims at implementing hardware architectures for DTs and RFs by developing a framework. Given the potential of these simpler solutions on battery-powered systems and the lack of detailed works in the literature regarding the analysis of these ML methods in terms of power and energy efficiency, it was decided that working on solutions for DTs and RFs would be the focus. The framework implemented in this thesis can be easily adapted to other tree-based methods. This proposal, considering only DTs, was published in (ABREU; GRELLERT; BAMPI, 2020), and its extension including RFs and additional analyses was published in (ABREU; GRELLERT; BAMPI, 2022). This framework was also released as an open-source contribution[1].

In order to justify the research for a low-power context, the focus was on data sets that are related to scenarios that could be found in embedded devices. Therefore, the primary goal was to analyze the power and energy efficiency of DTs and RFs in data sets related to wearable devices.

The Scikit-learn (SK) (PEDREGOSA et al., 2011) library from Python contains methods for modeling DTs and RFs, namely *DecisionTreeClassifier* and *RandomForestClassifier*. These are well-known solutions in data science: along with methods that enable the manipulation of the data sets, such as the *train_test_split* function, a designer can easily train these ML models with their own set.

The *DecisionTreeClassifier* method allows for the variation of the maximum tree depth of the DT. As stated in Section 2.2.1, this limitation early-terminates the splitting process that was being performed through the calculation of the Gini index (or entropy). This is useful when such a deep tree is not required, given that it may lead to increased execution time and complexity for the training and inference processes. The *RandomForestClassifier* method, besides also supporting the limitation of the tree depth of every DT, allows the designer to choose the number of estimators (trees) required for the model. This leads to more trees being calculated in parallel, which may result in increased accuracy, given that the number of voters deciding for the final outcome is higher. However, this also results in more time to execute the inference, as well as increased complexity.

[1] Available on: <https://github.com/Brunno815/Framework_DTRF>

Additionally, limiting both maximum tree depth and number of trees may be beneficial, given that it may decrease the overfitting aspect of the models. For example, a DT with an unlimited depth may split the data based on excessive specificities, which may lead to incorrect predictions when the test set is being analyzed. The same analogy may be applied to the number of trees in a RF: given that many weaker voters are instantiated, they can be responsible for making a wrong prediction that a RF with fewer voters would not make.

Despite the high support for the variation of parameters, SK does not support an analysis on any kind of hardware architectures for DTs or RFs, given that the library is intended for software purposes. Therefore, no precise power analysis can be performed using only SK. At the same time, if we desire a detailed trade-off analysis including parameters such as power, area, and accuracy, for each data set, manually describing hardware architectures for each different scenario would be unfeasible. For such an extensive analysis to be conceived, creating a dedicated design automation framework that generates the hardware architectures for the DT and RF models, based on the SK structures, is required. Even though solutions such as HLS are becoming popular as well when it comes to automatically generating RTL descriptions based on a software code, these tools are still not useful enough when power-efficient scenarios come into play. Also, HLS only focuses on generating one single RTL model for each high-level description. As it will be seen, the framework developed here can generate several models at once and perform a broad design space exploration; hence, this work does not directly compare with HLS.

The goal of this tool is to output RTL descriptions, in HDL format, of the tree-based models (equivalent to their software descriptions), along with testbench files that read data from the respective data set for which the hardware was described. These testbenches are used in the simulation process to obtain the switching activity, so that the synthesis can provide accurate power results. The power estimation methodology will be detailed in Section 6.2. Additionally, a debugging environment was created in order to assess the correctness of the described architectures.

For this specific framework, the architectures were developed as dedicated as possible (i.e., the constants of the models were all hard coded) and described specifically for each data set analyzed. Therefore, the tool generates an architecture that is the unrolled version of the DT or RF. As an example for a DT, it may be convenient to visualize its software structure as a function with a chain of *if-else* operations. This is presented in

Algorithm 1, for a structure with a tree depth of 2, with three output classes. The RF structures are simply composed of several of these DTs with a voter.

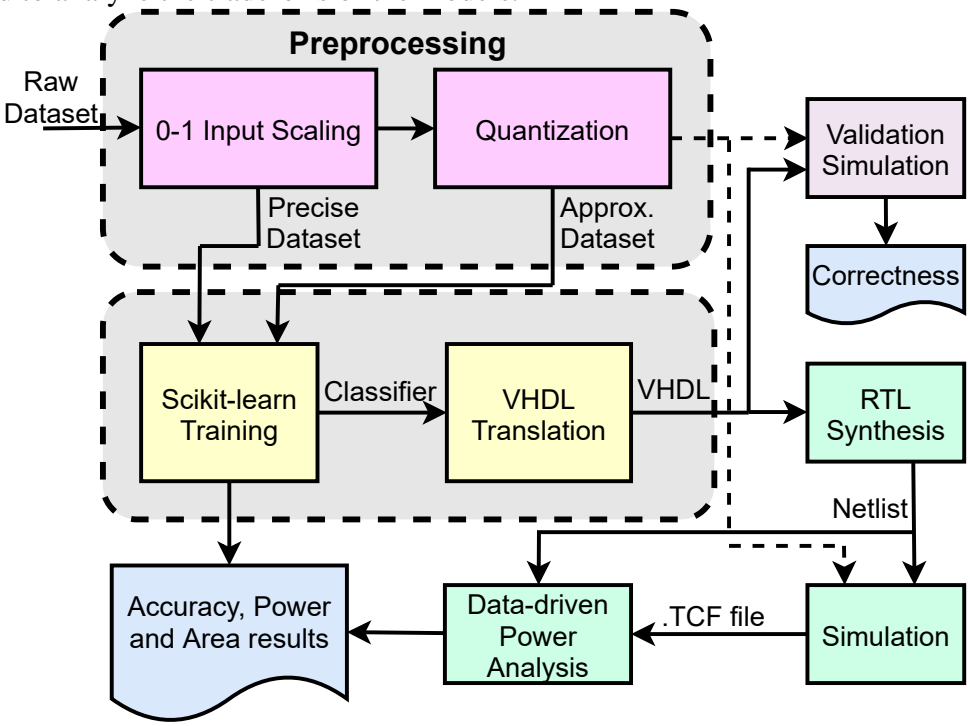
```
Function dt (feat_0, feat_1, feat_2):
    if feat_0 ≤ cte_0 then
        if feat_1 ≤ cte_1 then
            return 0;
        else
            return 1;
        end
    else
        if feat_2 ≤ cte_2 then
            return 2;
        else
            return 1;
        end
    end
return
```

Algorithm 1: Example code for a DT.

The HDL architectures are generated from these structures. It may be intuitive to associate each *if-else* comparison with hardware comparators, which are easily described in hardware language. The paths leading to each result can be described by 'AND' and 'OR' gates. Even though the paths could be described by a chain of multiplexers, preliminary analyses did not present significant differences in the power results. The voting operation in the RF can be described by some variant of a majority gate, or a chain of comparisons between the final sums of votes for each class. These considerations will be detailed later. Additionally, quantization techniques are employed in the features, so that the width of the comparators can be decreased when describing the hardware accelerators.

This work can be better visualized through a design flow that was created. The process flow employed in this work is summarized in Figure 5.1. Each part is explained in the following sub-sections. This flow was also used as a baseline for the other two proposals performed in this thesis, presented in Sections 5.2 and 5.3.

Figure 5.1: Processing flow of the framework that generates tree-based accelerators, employed to analyze the trade-offs of the models.



5.1.1 Preprocessing

In order to ensure a generalist approach, features are expected to be in the floating-point precision format, given that this is the case in many practical scenarios. Therefore, even though some data sets may contain features with integer values only, the most generic type for them are floating-point. These features can be described using the IEEE Standard for Floating-Point Arithmetic (IEEE, 2008), which defines arithmetic formats, rules, operations, etc., so that the operators can be portable between application domains. However, this work is focused on power-aware approaches, and using IEEE 754-compliant Floating Point Units (FPU) would require hardware resources too power-hungry for the goals of this thesis. For this reason, fixed-point units are employed in the proposed designs.

The first step of the design flow is to scale the features to 0-1 ranges. This is done to standardize the features coming from the data sets. This remapping is performed as shown in Equation 5.1, where X denotes the value of the feature, X_{min} and X_{max} refer to the maximum and minimum value of the same feature in the data set.

$$X_t = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5.1}$$

Then, the values of each feature f are quantized so that they correspond to the

maximum input width of the inference kernel (Max_{width}), as shown in Equation 5.2. The value for Max_{width} defines the quantization level.

Given that some features present very few different values between them, we would waste an unnecessary amount of bits if we defined the same Max_{width} value for these features as well. For this reason, it was defined that features with less than 20 unique values (defined by $Nvalues$) were deemed categorical and thus limited to the required amount of bits for them. This was applied by analyzing the features from the datasets and observing that features with more than 20 different values mostly had several values, whether from being integer values or floating-point ones. In some cases, features that presented few values are related to the gender of the user, or some other categorical context. Therefore, less bits were applied to these features. As an example, if a feature presents only 14 different values throughout the whole data set, we would only require 4 bits, according to Equation 5.2 (first condition).

$$Nbits_f = \begin{cases} \lceil \log_2 Nvalues_f \rceil, & \text{if } Nvalues_f \leq 20 \\ Max_{width}, & otherwise, \end{cases} \tag{5.2}$$

It is important to notice that this value of 20, as presented in Equation 5.2, can also be generalized. However, including this variable to the work would scale the number of required analyses and syntheses even more. Therefore, changing the threshold below which features are deemed categorical is out of the scope of the thesis.

Then, the features are simply scaled back based on the value of $Nvalues$. This is performed by multiplying the scaled feature (with the 0-1 range) to the power of 2 of the value of Max_{width}, as in Equation 5.3.

$$X_{new} = \left\lfloor X_t \cdot 2^{(Max_{width})} \right\rfloor \tag{5.3}$$

With the X_{new} values for the features, FPUs are not required. Given that the accuracy results can be obtained with SK independently from the hardware architectures, it is possible to assess the impact of the quantization step by calculating the accuracies with the precise (unquantized) data sets and with the quantized ones.

In order to estimate the accuracy results, this framework considers the hold-out method, i.e., the data set is split into training and test sets. For this work, the split used was 66% for the training set and 33% for the test set. It is important to notice that a validation set is not required in this case, given that fine-tuning the weights/parameters

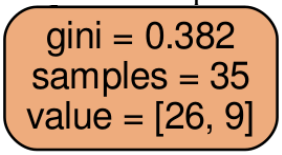
does not apply for a DT/RF, as the algorithms employed fix the constants with the iterative splits of the training set. This split was done with the *train_test_split* method from the SK library, which yields the training and test set separately. This method also stratifies the data set, maintaining the same proportions of outputs in each training and test set in order to keep a balance on the predictions.

5.1.2 Training and Accuracy Evaluation

With the training and test sets obtained, the *DecisionTreeClassifier* method is applied to generate the classifier structures. For the DT, the splitting metric considered was the Gini impurity metric. Even though the entropy metric was also available, both yielded very similar results. Preliminary tests were performed with both metrics, and they presented negligible differences between each other. For that reason, the default parameter (Gini) was maintained. The only parameter that was varied was the *max_depth*. To ensure reproducibility, the seed was maintained to a value of 0.

The leaf nodes of each DT in the SK library define the number of instances of each output class that remained after the last split. Figure 5.2 shows an example of a leaf node of a generic DT, with two classifications. In this case, the result of the output is the class with the most representation in that split. This is trivial when we denote the use of a single DT, as we can ignore these weights when designing the accelerator and only set the final output as the highest one. However, when it comes to the RFs, when using *RandomForestClassifier*, these weights are taken into account, given that the final decision – after the voter – considers the sum of the weights of each class in each DT operating simultaneously. Therefore, a weighted sum has to be taken into account.

Figure 5.2: Leaf node example in a DT from SK.



Given that it is not desirable to use a large number of adders as the focus is on low-power accelerators, it was decided to implement a dedicated RF method using several *DecisionTreeClassifier* structures, so there could be full control on how to implement the voter. For simplicity purposes, it was decided to implement it by solely considering the final decision of each DT, and sum these preliminary decisions for each output class –

note that this sum is much cheaper in terms of hardware resources, as it does not take into account the proportions of each class like they were presented in Figure 5.2. In case of a draw, the lowest index is chosen as the final decision.

In order to generate different DTs to assemble the RF, a subset of the features is considered, which is equal to the difference between the total number of features and the square root (truncated) of the total number. With this set of features, the *DecisionTreeClassifier* structure is applied. This step is performed for each DT required in the RF.

The process to execute the inference stage of the DTs and RFs uses the *predict* method of the SK library. Three different approaches are calculated for the analysis with this framework: (i) training and testing with precise data sets, (ii) training with precise data set and testing with quantized data set, and (iii) training and testing with quantized data sets. The first one is useful to consider how much the use of quantization loses in terms of accuracy when compared to an implementation using FPUs, and the third one was effectively employed in the architectures. The second one was theoretically calculated to estimate whether training with different but similar data domains could lead to good accuracy results as well.

The accuracies are calculated using the *accuracy_score* method, which simply takes the labels from the test set and the predicted values – obtained from the *predict* method – and compares them, yielding an accuracy.

5.1.3 Tree-to-VHDL translation

In order to automate the process and allow for a wide variety of DT and RF kernels, a tree-to-VHDL algorithm was also proposed and implemented in this stage. This algorithm receives the Python structure from the SK library – either a unique *DecisionTreeClassifier* or an ensemble of them, for the RF – and translates it into an RTL description in VHDL language. The goal of this stage is to output a module that receives the set of features as an input and yields the decision in the output.

As previously mentioned, DTs are composed of comparators that perform the $\leq$ (less than or equal to) operation between a feature and the constant defined by the model. This is performed as in Algorithm 2, with the use of processes.

The comparators are the main building blocks of the generated architectures. The translator runs through every node in the structure to find every comparison being per-

```
process(feat_0, cte_0):
    if (feat_0 <= cte_0) then
        cmp <=' 1';
    else
        cmp <=' 0';
    end
return
```

Algorithm 2: Process that performs the comparison in HDL.

formed. Note that it may be the case that the structure presents the same comparison in two parts of a DT. More likely, the different DTs of a RF structure may present redundant comparisons as well. Therefore, it is important to obtain the unique comparisons of the structure so that no redundant hardware is instantiated. The recursion performed to obtain the comparisons of a single DT is shown in Algorithm 3.

```
Function getComparisons(tree, featName):
    tree_ = tree.tree_;
    comparisons = [ ];
    Function recurse(node):
        if tree_.feature[node] ≠ _tree.LEAF then
            name = featName[node];
            thr = tree_.threshold[node];
            if [name, thr] ∉ comparisons then
                comparisons.append([name, thr]);
            end
            recurse(tree_.children_left[node]);
            recurse(tree_.children_right[node]);
        end
    return
    recurse(0);
    return comparisons;
return
```

Algorithm 3: Function that obtains the comparisons.

This process is performed once when the structure is only a DT, and for each DT when it is a RF. After this, the list is obtained by appending every list of comparisons, and the repeated values are removed.

The second step is responsible for creating boolean expressions that lead to each classification. The literals of each expression are within the n unique comparisons, which are named $c_0, c_1, ..., c_{n-1}$. To do that, the algorithm recursively runs through the DT (or ensemble of DTs) once again, for each classification. Then, it recursively considers every comparison all the way down to the leaf nodes, creating AND expressions for each

path from the root to the leaf node. The paths that lead to the same classification are merged through an OR operation. Given that the conditions are exclusive, there can be separate expressions for each of the classifications, as they will never lead to a $'1'$ logic value simultaneously. For example, for a multi-classification problem with 4 different classes, the algorithm runs through the structure for each class to obtain these expressions. Algorithm 4 shows an example of obtaining the expression for a single classification within a DT.

Function `getExpr(`*tree, featName, outIdx*`)`**:**
  $tree\_ = tree.tree\_$;
  $ors = [\,]$;
  Function `recurse(`*node, expr*`)`**:**
    if $tree\_.feature[node] \neq \_tree.LEAF$ **then**
      $name = featName[node]$;
      $thr = tree\_.threshold[node]$;
      $exprLeft = expr + [[name, thr, TRUE]]$;
      $exprRight = expr + [[name, thr, FALSE]]$;
      $recurse(tree\_.children\_left[node], exprLeft)$;
      $recurse(tree\_.children\_right[node], exprRight)$;
    else
      if $argmax(tree\_.value[node]) == outIdx$ **then**
        $ors.append(expr)$;
      end
    end
  $recurse(0, [\,])$;
  return ors;

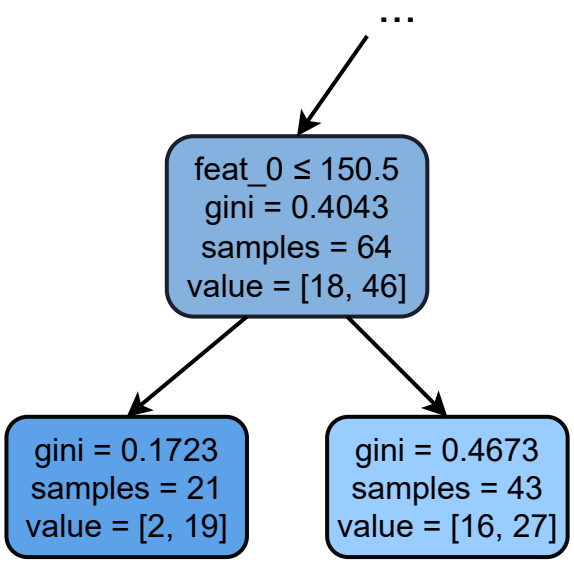
Algorithm 4: Function that obtains the expression for a single class in a DT.

For the specific scenario that the problem is a binary classification one, obtaining the expressions may be performed differently. In this case, only the expression for the cases that lead to one of the classes have to be obtained. Otherwise, the expression will be a zero, in which case the correct class is the remaining one. We can consider one output for this binary classification problem, and it becomes either a one or a zero. If this pattern was maintained, encoding all the outputs in a binary way would be required when the problem was a multi-classification one, because this solution of one single expression does not work for more than two classes. Therefore, it was decided that the number of outputs for a multi-classification problem would be the same as the number of classes. Hence, there is one explicit expression per class obtained in the recursion of the DTs.

After this process, the framework obtains one expression per class, in the more generic scenario. This expression can contain redundancies and can likely be simplified

or optimized. This may happen in any case where the two children in a comparison node lead to the same classification. In this case, the recursion will go through the whole expression for the $true$ side and for the $false$ one. This can be exemplified with the portion of the DT shown in Figure 5.3.

Figure 5.3: Portion of a DT whose children of a decision node lead to the same classification, indicating a possibility for simplification.



In Figure 5.3, the comparison is performed from $feat\_0$ and the value 150.5. In this case, when the training algorithm is applied to split the data set and define the comparisons (using the gini metric), 64 samples are left in this portion of the tree. The parameter $value$ indicates how many samples from each different class are left in this node. In this case, a binary problem is presented. When splitting the tree even further, two nodes are defined, which are leaf nodes. In the leaf nodes, the class with the most samples is defined as the final decision. In this situation, no matter the result of the comparison between *feat_0* and the constant 150.5, the result will always lead to the second class, given that it has the most instances in both leaf nodes.

To simplify the expressions, the Python Boolean library (KRAMER, 2009) was employed. This library contains the *simplify* method that performs the simplification of the expression, and it was applied for this case. There are specific scenarios where a feature appearing in a tree may not even be required in the architecture if it is only used as a comparison where both children lead to the same class. Therefore, simplifying the expressions may even lead to reduced required bandwidth.

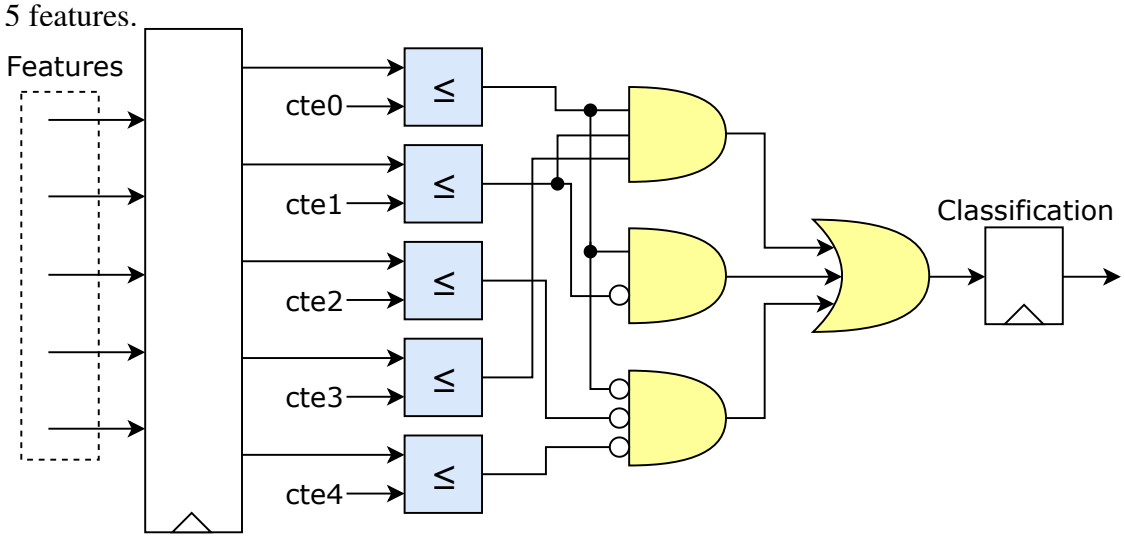
After the expressions are obtained, the last stage performs the mapping between comparisons and paths to the VHDL description. The comparisons were mapped to the

aforementioned process, in Algorithm 2. The mapping of the expressions uses AND/OR gates, as previously detailed. Additionally, the features are all registered in the inputs, so these specificities must be taken into account as well.

In the specific scenario of a RF, the voter also needs to be mapped. This has to be done with the use of adders that will keep track of the sum of the preliminary decisions of each class, considering all DTs. This is performed with a tree of small adders for each class. It is left for the optimization process to map this to the elements of the standard-cell library in an efficient manner.

Figure 5.4 illustrates part of a generic DT architecture generated by this translator. Only a single classification output is shown, but there will be several AND/OR combinations similar to the one shown, for each output label. The constants were defined in the training stage, and the comparisons and paths were obtained in the stages of the translation.

Figure 5.4: Illustration of an architecture generated by the tree-to-VHDL translator, with 5 features.

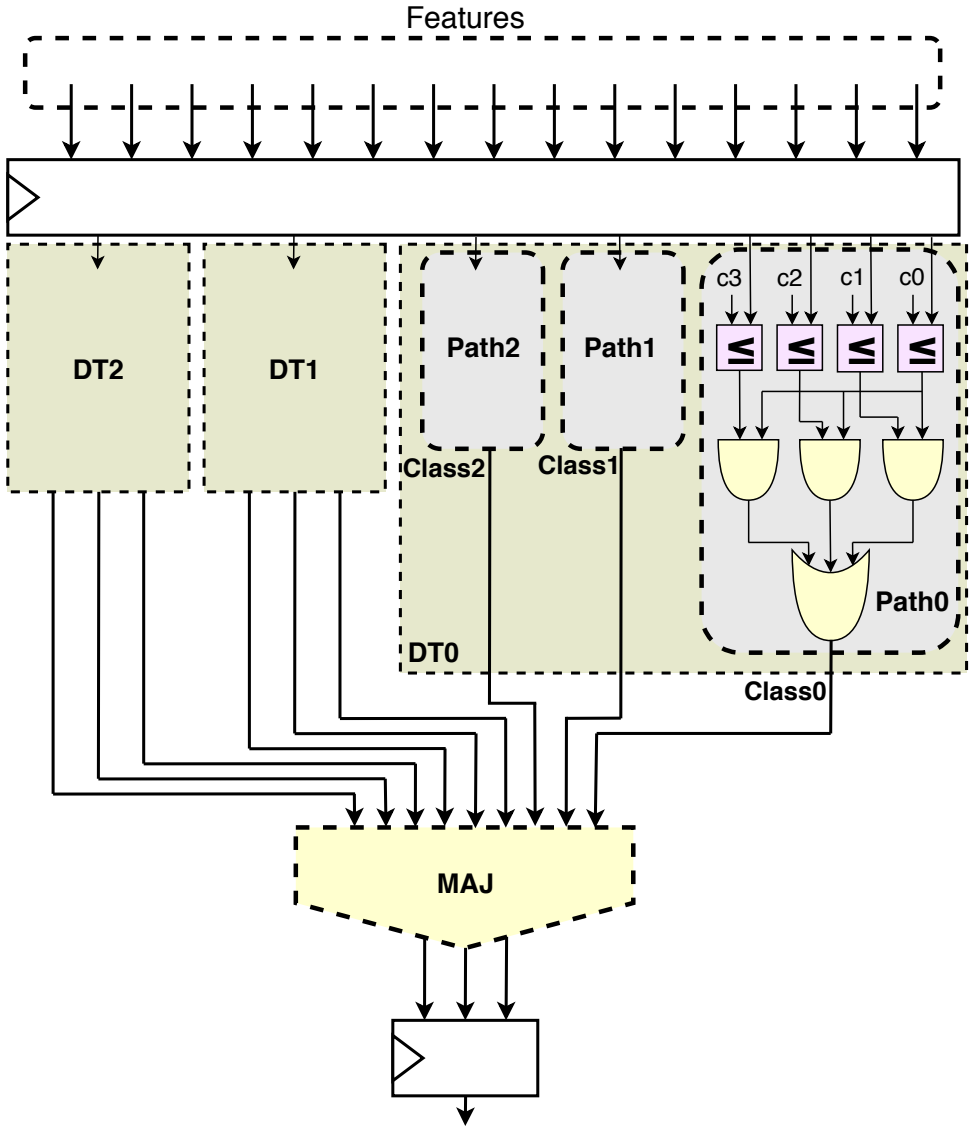


The architectures only consider input and output registers; therefore, no pipeline is applied. This design decision is warranted by the fact that the focus is on power-aware scenarios, not targeting high frequency. Inserting many pipeline registers would lead to higher power dissipation. Additionally, even though using pipeline could allow us to synthesize the architectures to higher frequencies, this is not necessarily needed when considering simpler AI tasks such as the ones in wearable devices (as will be presented in Section 6.1). Therefore, exploring various levels of pipeline is not aligned with the scope of this current work.

Figure 5.5 shows a RF accelerator generated by the translator, considering a sce-

nario with three labels. The constants (represented by $c0$ to $c3$ in Figure 5.5) were defined by the training stage, with Scikit-learn, and the comparisons and paths were obtained in the stages of the translation.

Figure 5.5: Block-diagram of a RF architecture generated by the VHDL translator.



The case studies for this proposal are wearable devices. However, the translation methods can be performed for any kind of data set that is explicitly separated in features.

5.1.3.1 Testbench Generation

The testbenches are required to perform the simulation that will obtain the switching activity of the circuits. These testbenches were also automatically generated to account for the different data sets evaluated and for the different sets of inputs depending on the features that are required by the architecture. These files simply read the input files

from the entire data set.

It is important to notice that, even though the test set is used to obtain the accuracy results (so that the performance of the model can be verified considering unseen data), the testbenches read the entire data set. This is due to the fact that it is desirable to have as much data as possible to infer the switching activity, and some data sets would not be large enough to reliably provide that if only the test set was used. This is acceptable because the model was not built as a function of the power dissipation or the switching activity itself, so estimating the activity through these data would not bias any result.

5.1.3.2 Validation Environment

Once the translation is performed, all the DTs and RFs considered in this work are tested in a validation environment, which runs the hardware descriptions and the Scikit-learn $predict$ method to determine the correctness of each one, for a sub-set of lines of each data set employed. After the correctness of each configuration is verified, the synthesis and execution steps are ready to be executed. Also, given that the solutions were validated in this step, the accuracy results can be obtained from the Scikit-learn methods, as they are equivalent to the result from the RTL description. Additionally, the accuracy results for the precise models were obtained as well, to perform a comparison and verify the impacts of applying quantization to the architectures.

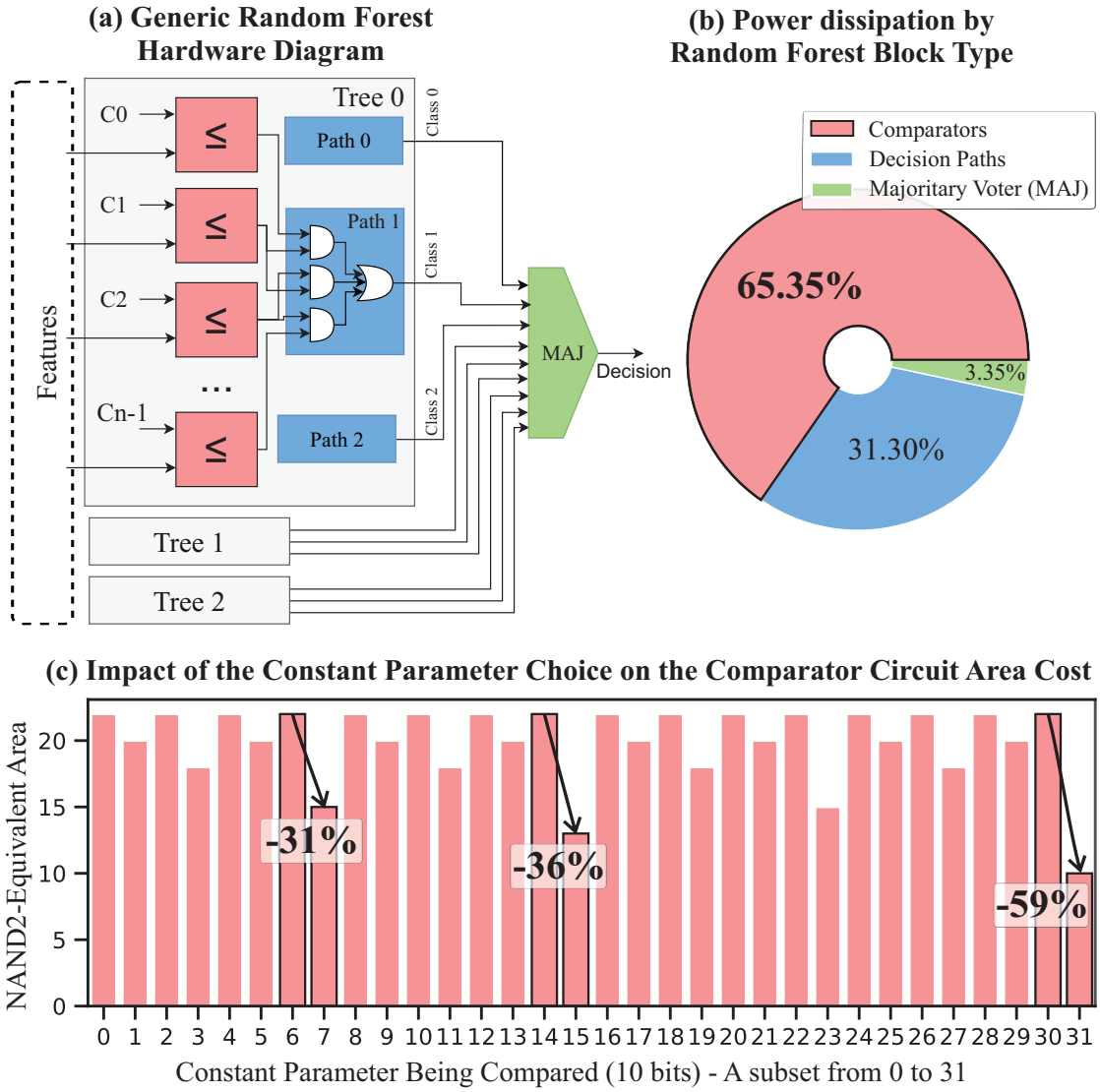
5.1.3.3 Synthesis and Simulation

The last part of the flow synthesizes the RTLs generated by the translation step. This process generates a netlist with specific gates from the chosen standard-cells library. This netlist is then used in the simulation process, to generate Toggle Count Format (TCF) files, which are useful to obtain information on the switching activity for the data-driven power analysis. Then, the desired results are obtained. This process is presented with more details in Section 6.2.

5.2 Complexity-aware Constant Parameter Approximation

This proposal analyzes a custom method developed in this thesis to approximate the comparators in tree-based models, denoted as Complexity-aware Constant Parameter Approximation (C2PAx). Figure 5.6 presents a general motivation for the proposal. All

Figure 5.6: (a) RF block diagram with 3 trees, each containing comparators, decision paths, and a voter; (b) Power dissipation per block type in a RF accelerator; (c) normalized area of comparators for different constants.



the analyses in this proposal only consider the use of RFs instead of DTs, given that RFs are composed of several trees and can be seen as a more generic scenario. This work was accepted and is already published at IEEE Transactions on Circuits and Systems - I (ABREU et al., 2022).

As considered in the previous proposal from Section 5.1, one can opt to implement RF accelerators as dedicated as possible, describing the comparators with the constants explicitly defined. In this case, the logic synthesis tool can optimize its comparator operator to work only for that specific constant.

In Figure 5.6-a, a generic RF accelerator is presented. RFs are composed of three elements: (i) comparisons by constants (in pink), whose values are computed in the training stage, (ii) decision branches, implemented with multiplexers or AND/OR chains, and

(iii) majority (MAJ) gates. As it can be seen, RF accelerators are hardware-friendly models, without complex arithmetic operations. Even so, there is significant room for optimization in tree-based accelerators regarding energy efficiency.

The comparator units are the most power-consuming components of tree-based accelerators, accounting for more than 65% of the total power dissipation. This is shown in Figure 5.6-b, which depicts the average power dissipation of accelerators for RFs trained on four data sets (which will be detailed in Section 6.1). Therefore, optimizing the comparisons might lead to significant reductions in the energy and area of such circuits.

In Figure 5.6-c, the area of the comparators for different constants (for 10-bit operators) is analyzed, through standalone synthesis (i.e., out of the context of the RF architecture), using Cadence Genus (CADENCE, 2019). Based on the plot, it can be seen that the area of these comparators varies drastically. Only the first 32 values are presented for clarification purposes, but this varied pattern is similar for all the 1024 possible values. As it can be observed, reductions of up to 59% could be obtained by changing a constant from 30 to 31, for example.

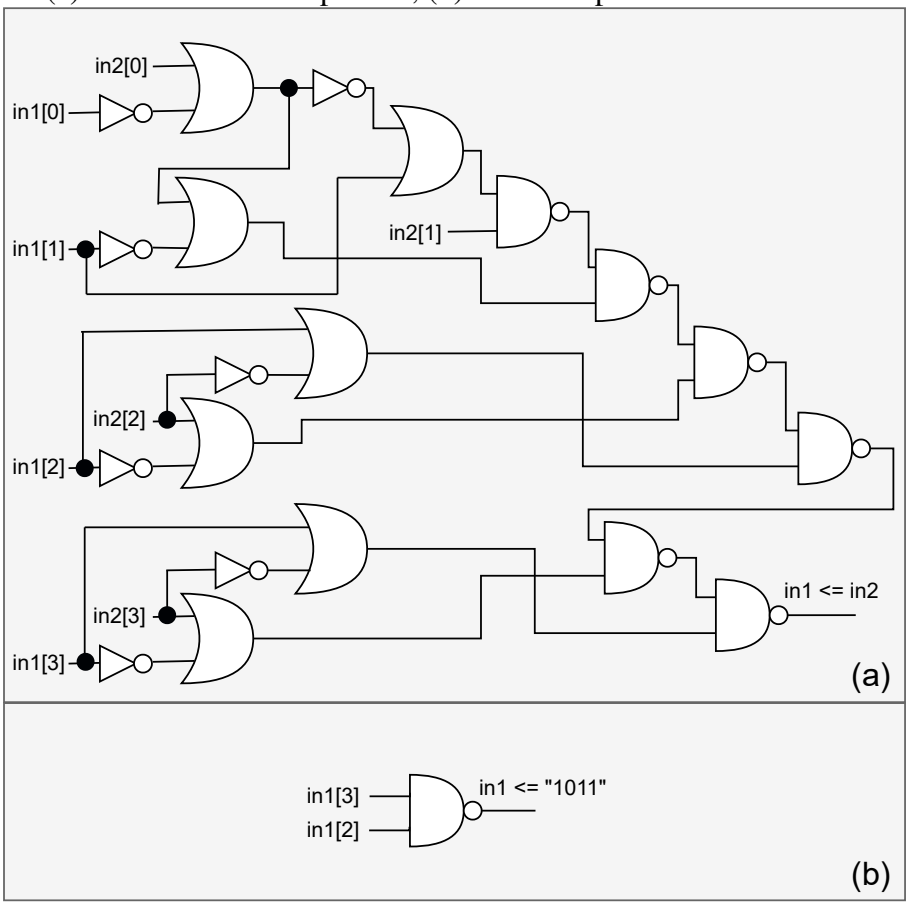
As an example in terms of gate-level circuit, Figure 5.7 presents both a 4-bit generic comparator that performs the less than or equal operation (Figure 5.7-a) and a 4-bit comparator that works only for the specific constant of 1011 (Figure 5.7-b). The circuits were obtained from an unmapped synthesis using Cadence Genus. As it can be seen, employing comparators exactly as required by the ML model trained, as opposed to fully generic comparators, can lead to significant area savings.

The flow that describes the framework used in this work to implement the C2PAx proposal is presented in Figure 5.8. C2PAx has also been released as an open-source contribution[2].

This proposal is basically an extension of the framework presented in Section 5.1. The previous framework only considers the approximations in terms of quantization, enabling the variation in tree depth and number of trees. This extension allows for the approximation in the constants of the comparators. Just as in the previous proposal, this one is agnostic to the data sets being employed, as long as the comparisons are described using explicit constants. More generically, this proposal can work for any other tree-based models, such as XGBoost (CHEN; GUESTRIN, 2016), or other architectures that allow the insertion of errors (not only tree-based models) and use comparators with fixed constants.

[2] Available on: <https://github.com/Brunno815/C2PAx>

Figure 5.7: (a) Generic 4-bit comparator; (b) 4-bit comparator for the constant 1011 (11).

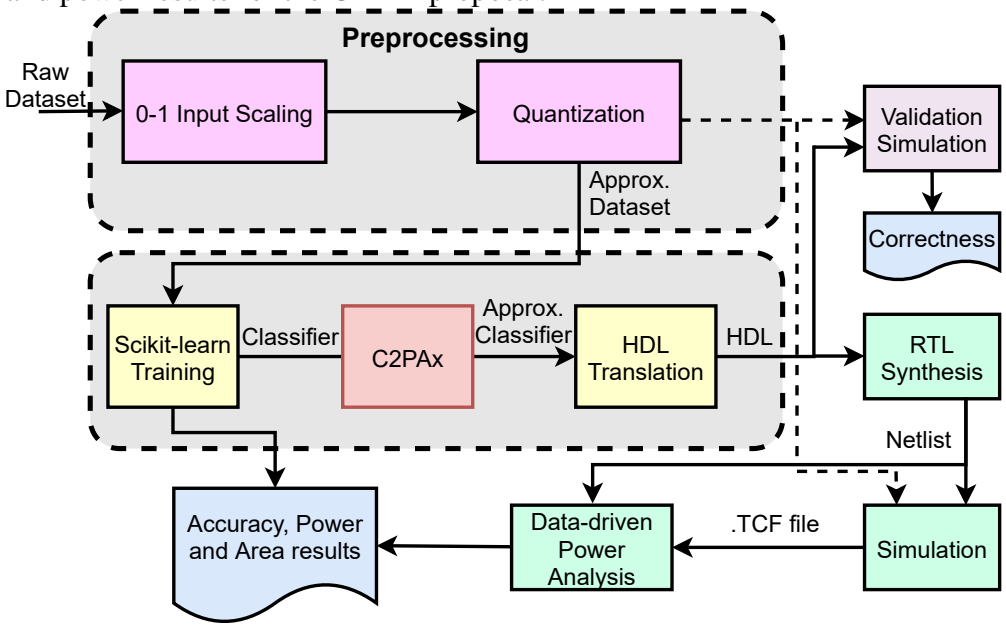


Before training, every data is converted to positive numbers and then quantized to natural values, so that it does not need to apply any floating-point units, ensuring a simpler hardware. Therefore, every constant in the RF is composed of natural numbers, as in Section 5.1. For the analyses regarding this proposal, based on preliminary results of the baseline framework, the features are quantized to a 10-bit integer representation.

For the training stage of the RF, the *RandomForestClassifier* structure was employed. Some simplifications were performed so that there was no need to consider the weighted sums of partial votes in each tree. Even though this was also considered in the first proposal from Section 5.1, this updated version of the framework does not consider several instances of *DecisionTreeClassifier* structures. Instead, the exact same initial structure from *RandomForestClassifier* was used, but with a modified voter. For the evaluations in Section 6, the number of estimations is set to 5 for the C2PAx proposal.

The adjustment of constants may be performed in several different ways. A baseline approximation method is defined, denoted as Don't Care (DC). This technique for approximating the constants works by simply defining the constant and applying DC signals (SALAMAT et al., 2017) to the LSBs of the constants being compared. For example, if a feature with the constant 0101100111 is being compared, it can be approximated to

Figure 5.8: Flow describing the framework employed to obtain the accuracy, VLSI circuit area and power results for the C2PAx proposal.

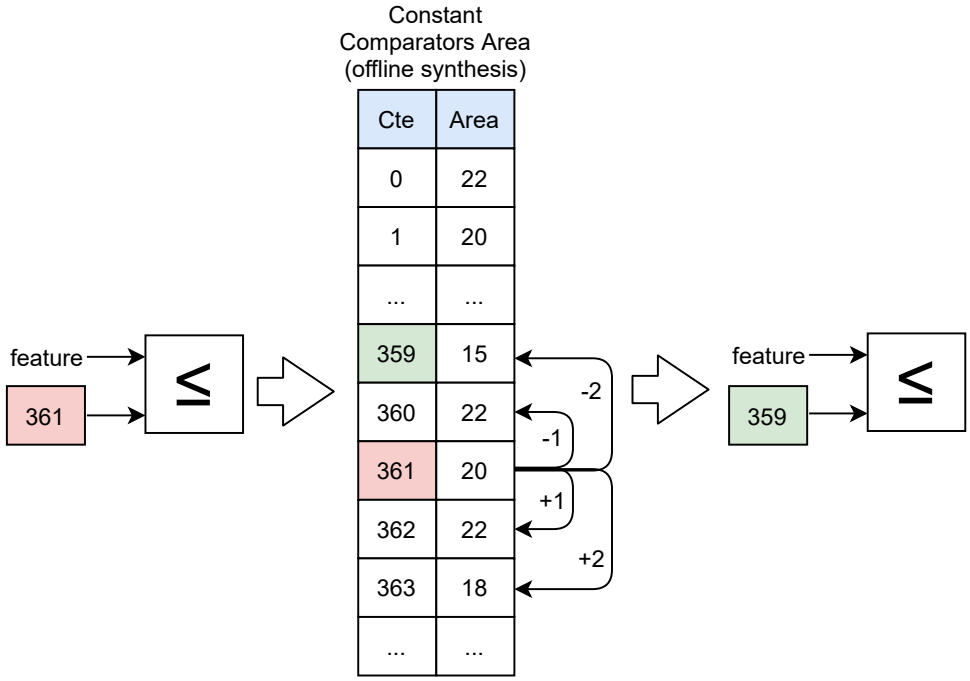


$010110011x$. This allows the synthesis tool to find the most suitable operator without considering a strict value in the LSB. Given that this technique seems promising for this specific application, this is implemented to be compared with the C2PAx proposal. For this technique, more than one bit can be assigned a DC signal. In the analyses in Chapter 6, the number of DCs is varied from 0 (precise case) to $n-1$, where n is the total number of bits in the constant.

The main idea behind the C2PAx proposal is to slightly adapt the constants of the comparisons so that the new comparisons are less costly in terms of area and energy, reducing the resources when considering the entire RF architecture being described. This may also change the accuracy of the model, depending on the feature being compared. Even though the aforementioned DC technique also attempts to adapt the constants in one way, it does not allow for a fine-grained variation. Additionally, it leaves the optimization of the constant solely to the synthesis tool.

The C2PAx proposal tests, from the constant of each comparator defined in the training stage, the constants close in a range of m, in both directions. For example, for a constant x, the constants searched will range from $x-m$ to $x+m$. The algorithm will ignore the analysis if the current value of the constant being searched is smaller than 0 or higher than the maximum value of that specific precision. This technique allows for a more fine-grained analysis, as the number of possible values for the range is much higher than the DC technique. This idea is illustrated in Figure 5.9, for a step of $m=2$ and a constant $x=361$. The list of areas presented in the figure is obtained through previous

Figure 5.9: Proposed C2PAx technique, considering an initial constant of 361 with a step of 2.



synthesis, with the standalone comparators synthesized with different constants (not in the context of any RF accelerator). For a step of 2, the technique decides to change the constant of the comparator from 361 to 359, since this is the one with the smallest area among the ones in a distance up to 2 from the initial constant value. This process is applied separately for every comparator in the RF accelerator.

Other ideas may be considered to the adjustment of constants. For example, some comparators may be more relevant than others to define the inference value, whether from being in a specific height of the tree. Therefore, applying different step values for different comparators in a DT/RF may bring beneficial results. It is also possible to adjust the constant to an opposite side from where a constant in the same path of a tree was adjusted. However, these analyses are out of the scope of this thesis.

The metric to define the best constants to be adjusted is also important. For this work, it was opted to synthesize every comparison, as presented in Figure 5.6-c (a subset of the 1024 possible values), and gather the area results, normalized by *NAND2* gates. Even though this could easily scale to a huge amount of syntheses required, some preliminary analyses for the framework proposed in Section 5.1 found that, for the data sets employed in this thesis, the use of precisions larger than 10 bits do not lead to better accuracy results. Therefore, only 1024 comparisons needed to be synthesized – if precisions smaller than 10 bits were required, only 1023 more comparisons would be needed. Addi-

tionally, this process only needs to be done once; hence, even if precisions larger than 10 bits were required, e.g. 16 bits, this process could be performed offline, to be used later for every other data set. This process does not apply for the DC technique, given that the resulting comparator after applying the DC will fully rely on the synthesis tool.

It is important to notice that the *NAND2* metric was only employed given that this is a normalized and fair metric. One could argue that directly using the power dissipation could lead to more accurate results; however, individually synthesizing the comparisons prevents us from finding the exact switching activity that each comparison will have under the full RF architecture. If that level of exactness was desired, we would have to synthesize every feature of the data set with every possible constant, which would not scale well. For these reasons, the decision was to use the *NAND2*-equivalent area as the metric. Using other metrics to choose the best constant is out of the scope of this work.

For a more generalist approach, one can synthesize the comparators without any sort of mapping to a standard-cells library. This means that the netlist generated by the synthesis tool will have generic gates, and whose area is also estimated by the synthesis tool. Even though this does not depend on any standard cells – which is positive given that it can be used to estimate the best comparisons when synthesizing for any technology –, this method could lead to worse results, given that a different netlist would be generated. The differences between these approaches were also analyzed and are discussed in Chapter 6.

5.3 DTs/RFs with Gate-Level Pruning

5.3.1 Indiscriminate Pruning

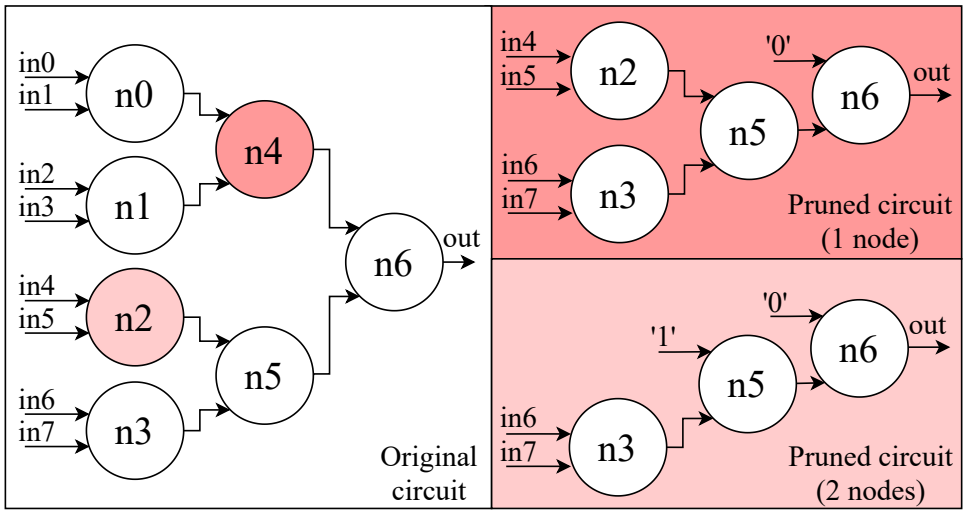
This proposal explores the use of indiscriminate gate-level pruning in DT and RF accelerators, to analyze the energy- and area-accuracy trade-offs. This work was accepted for publication (ABREU et al., 2022).

As it was seen, several approximate techniques can be employed in tree-based models, at many levels of the design stack. In higher levels, approximations can be introduced using quantization in the input features, by limiting the maximum tree depth and/or the number of estimators, or by approximating the constants of the comparators (C2PAx). In lower levels, one can perform gate-level pruning based on the description of the accelerator after logic synthesis steps (i.e., netlist).

This technique can be used synergically with the other approximation techniques employed in this thesis, such as quantization and C2PAx. Therefore, a designer working on a system with fewer resources can evaluate the best possible model to its energy constraint, while losing minimal accuracy. To the best of the author's knowledge, this is the first work to propose such technique for tree-based architectures.

A tool for performing indiscriminate gate-level pruning in the netlists was recently proposed in (CASTRO-GODíNEZ et al., 2021), denoted as AxLS (available online). This tool receives a netlist containing logic elements from a standard-cells library, along with a file describing the switching activity of the gates. AxLS allows for two kinds of pruning techniques: *inOuts* and *probprun* (LINGAMNENI et al., 2011). The first technique eliminates inputs based on the user's choice, hence eliminating some gates that rely on these inputs. The *probprun* technique (employed in this thesis) prunes nodes based on the ones with the least switching activities, based on input parameters. The use of pruning in the netlist level may lead to approximations that the other levels cannot achieve, given that the designer usually does not have any control of what the synthesis tool generates from the RTL descriptions. An example of the *probprun* is presented in Figure 5.10, for a generic circuit. In this example, $n4$ presents the smallest switching activity of all nodes, followed by $n2$. For a pruning with only one node, $n4$ and all nodes whose outputs are unused (in this case, $n0$ and $n1$) are pruned. When pruning for two nodes, $n2$ is excluded as well.

Figure 5.10: Pruning technique employed by the AxLS tool (CASTRO-GODíNEZ et al., 2021).

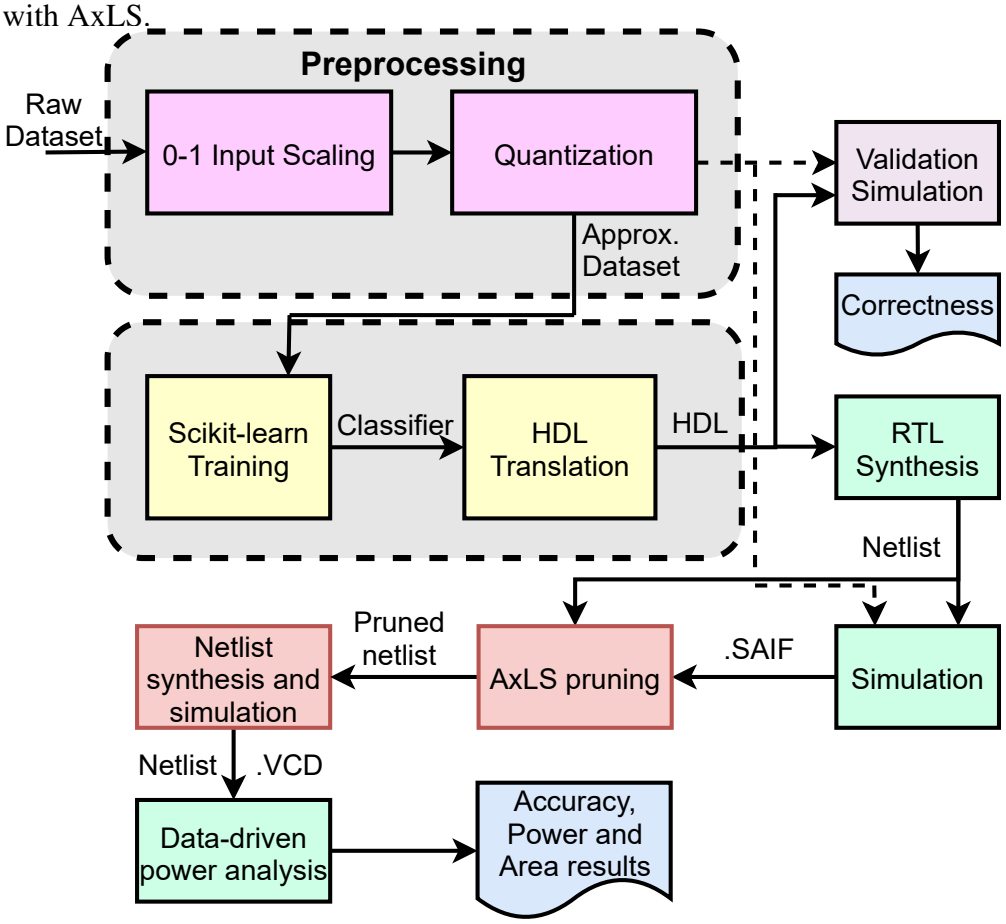


The term *indiscriminate* denotes the fact that, when generating a netlist to be pruned, the hierarchical setting of the modules of the RTL description is lost. Therefore, pruning will occur on the gates with the least switching activity, regardless of the

module they were part of before the synthesis.

This technique is also an extension of the framework developed in Section 5.1. Given that this approximate technique can be employed generically for any other proposal, it is important to evaluate its effect on several data sets to verify whether it is feasible to be applied. The flow used in this proposal is presented in Figure 5.11.

Figure 5.11: Flow used in this work to apply gate-level pruning using the DT/RF generation with AxLS.



Similar to the previous proposals, this flow considers data sets related to wearable devices. Also, the same *DecisionTreeClassifier* and the handcrafted *RandomForestClassifier* structures were employed. The number of estimators was fixed to 5 as well.

The translation process is similar to the one performed in the first proposal in Section 5.1, but for Verilog language. This was performed given that this language is less verbose than VHDL and is an enhancement of the baseline framework.

The RTL descriptions are used in a logic synthesis flow with Cadence Genus. Besides the netlist, the AxLS tool requires Switching Activity Interchange Format (SAIF) files. These files, as the name suggests, contain the switching activity of the circuit gates, based on the inputs. To generate them, simulation of the descriptions is required, with specific inputs. For this case, we use the inputs regarding each respective data set for

which the architecture was designed.

Then, the netlist and the SAIF file are used as input to the AxLS tool. The SAIF file is used to apply the *probprun* technique. For this thesis, the use of the *inOuts* technique was not considered, given that the quantization process (described in Section 5.1, to avoid the use of FPUs) already performs indirect eliminations to the inputs of the model. Therefore, the use of *inOuts* would be redundant and would depend on the designer's specific choices of elimination (or through random pruning), whereas the *probprun* technique only relies on the switching activities of each data set.

The AxLS tool generates a new cropped netlist, in which the n nodes with the smallest switching activities are eliminated from the original netlist. This new file is then resynthesized using Cadence Genus, simulated, and a data-driven power estimation process is performed using the input-vectors of the data set, to obtain accurate switching activity of the circuit. Then, the area, power/energy, timing, and accuracy results are gathered.

A proposal regarding iterative pruning is also possible. In this case, instead of pruning a number of nodes in a single batch, one may decide to prune less nodes, synthesize the circuit, and prune nodes once again, or in even more iterations. For example, when deciding to prune 12 nodes, instead of a single pruning, a designer can opt to prune 4 nodes, synthesize the circuit, and do this process for a total of three times. This will lead to a different netlist each time, different from the regular pruning that ranks the initial netlist and prunes the 12 nodes with the least switching activity.

5.3.2 Hierarchical Pruning

In most VLSI circuits, there will be modules that are more tolerant to approximation than others. For example, one may wish to prune part of a datapath that contains arithmetic operators, and decide not to prune a finite-state machine (FSM) of a control portion, given that it can fully break how the circuit operates.

In this proposal, instead of synthesizing the flattened netlist, i.e., indiscriminately from the modules described in the respective RTL, we keep the information of every module. Therefore, the netlist will contain the gates inside each of the modules in which they are supposed to be. Hence, the designer aiming to approximate the accelerator can have better control of which modules they want to prune and how much, based on prior knowledge of the circuit.

It is also important to notice that the possibilities to approximate the circuits depend on the way that the modules in the RTL were designed. Therefore, the circuit has to be properly described in a way that it can take advantage of the hierarchical pruning.

Furthermore, this will allow for more combinations of pruning w.r.t. flattened netlists, given that the proposal allows for pruning from 0% to 100% in each of the modules. In a example, for a circuit with two modules with 100 gates each (and its flatten version with 200 gates, for simplification purposes), in the first case we would have 100^2 possible pruning combinations, whereas the flattened version would only allow for 200.

This proposal was also implemented in the form of a framework, to generalize and ease the design space exploration. The framework was implemented as an extension of the AxLS framework. Most of the modifications required were regarding the parsing of how the netlist is generated when using the hierarchical option. In this case, since Cadence Genus is implemented, the command used in the synthesis process to allow for hierarchical pruning is to set the $auto\_ungroup$ attribute to $none$. It is important to notice that generalizing the implementation to several synthesis tools would require the parsing to account for the various syntactic differences between the netlists generated from each tool. Therefore, the work developed only focuses on Genus.

Even though the proposal can be applied to any netlist, this thesis will present its application on RFs. As it was already seen in the previous proposals, RFs are composed of a module with comparators, one with the 'AND/OR' paths and a final module for the voter. Each of these modules can be pruned separately in the hierarchical pruning. In Chapter 6, the differences between indiscriminate and hierarchical pruning will be shown, to verify whether using hierarchical pruning can lead to a better Pareto curve.

In the case of indiscriminate pruning, the circuit is synthesized in a regular manner, pruned, and then synthesized again to account for possible optimizations that the pruning may have caused. For the hierarchical approach, the difference is that the first synthesis is an ungrouped one. Then, the circuit is pruned according to the directives set by the designer, who decides which modules to prune. Lastly, the pruned circuit is synthesized in a normal way, generating a netlist containing only a top-level module, to also account for possible optimizations that the hierarchical pruning may have generated.

5.4 MAJ-based Logic Synthesis for Tree-based Models

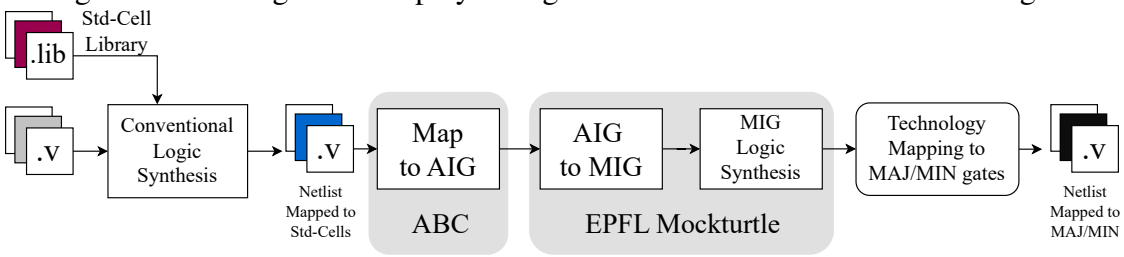
These explorations were performed based on a previous work published by the

author. The paper can be found in (ABREU et al., 2023) and proposes compact majority (MAJ) and minority (MIN) gates employing the body biasing feature present in fully depleted silicon on insulator (FDSOI) technology nodes. The work in (ABREU et al., 2023) manages to reduce the number of transistors by 2.3× and 2× for the MAJ and MIJ, respectively, when compared to traditional CMOS implementations.

Despite being related to the aforementioned study, these works were based on the fact that several emerging technologies operate mostly based on MAJ logic gates (MISHRA et al., 2021). This is the case of quantum-dot cellular automata (QCA) technology, for instance. This has led the MAJ-based logic synthesis paradigm to arise to overcome the conventional logic synthesis (AMARU; GAILLARDON; MICHELI, 2015) when using these technologies, given that any logic function can be described with only MAJ and MIN gates. Given that some emerging technologies can implement MAJ gates more easily, converting accelerators to use only MAJ/MIN gates and applying optimizations to their MAJ-based versions is an important topic to address.

The work in (ABREU et al., 2023) applied the Mockturtle tool (SOEKEN; AL., 2018) to perform MAJ-based logic synthesis. The flow employed to generate netlists based on MAJ/MIN gates is shown in Figure 5.12. This flow has also been employed in the proposals that will be presented in the sub-sections below.

Figure 5.12: Design flow employed to generate netlists based on MAJ/MIN gates.



The flow starts from a conventional logic synthesis process, using a regular standard-cells library. The ABC tool (SYNTHESIS; GROUP, 2019) is responsible for converting the Verilog netlist to the AIG format, which is supported by Mockturtle. Then, Mockturtle receives the regular netlist, in AIG format, converts it to an alternative format denoted as MIG, and performs MAJ-based logic synthesis along with some optimizations. Most of the times, Mockturtle cannot convert every logic from a regular netlist to one using only MAJ (as some portions cannot be efficiently converted). Therefore, it leaves the output netlist with some AND, OR and INV operations. The conversion of both AND and OR to MAJ gates is done through a parser, based on the fact that an AND logic is obtained

when a MAJ gate has one of its inputs as '0', and an OR is equivalent to a MAJ gate with one of its inputs as '1'. Then, a netlist composed of only MAJ and INV gates is obtained. Then, an additional script is applied to convert any MAJ gates that are connected only to one INV gate to a MIN gate. A MAJ gate that is connected to more than one INV or another MAJ gate cannot be converted to an MIN gate. Therefore, these INV gates are not converted to MIN gates. Hence, the final netlist is composed of only MAJ, MIN, and INV gates.

MAJ-based logic synthesis was employed in DT and RF circuits in some proposals of this thesis. The two analyses performed for MAJ-based DTs and RFs are presented in the following subsections.

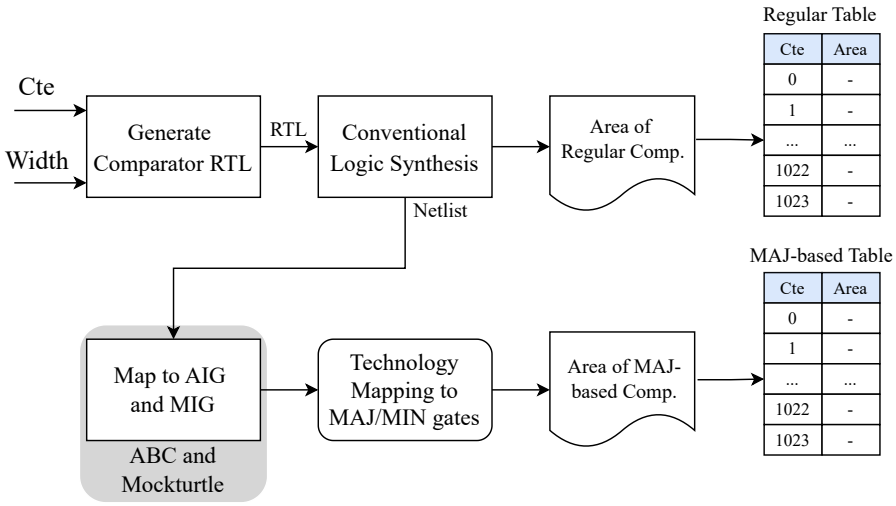
5.4.1 MAJ-Based C2PAx

This first exploration attempted to verify whether applying the C2PAx technique in tree-based models that use MAJ gates is a promising approach that may lead to interesting results. As seen in Section 5.2, the C2PAx technique requires a table with the area results of the comparators for every possible constant, for a specific width (see Figure 5.9). This table is obtained by synthesizing all comparators using regular logic gates (mapped to a standard-cells library or not).

When MAJ-based DTs/RFs (i.e., DTs/RFs whose RTL descriptions are processed in the flow from Figure 5.12) are considered, it is interesting to verify whether using this standard table of area is the most suitable approach. Alternatively, one can opt to synthesize every comparator and convert the structures of the comparators to MAJ/MIN gates using the flow from Figure 5.12, and obtaining a new table of areas considering the costs of the structures with MAJ/MIN gates, which may lead to different relations between the different constants. Figure 5.13 presents the idea more clearly, with the two tables being highlighted here.

The process in Figure 5.13 considers the tables using comparators of 10 bits, which leads to each table containing only 1024 lines. The process starts by generating an RTL description of a comparator, using a Python script. Then, conventional logic synthesis using a standard-cells library is performed, to generate a netlist. In order to obtain the area for the regular table, no other process is required, given that this is the final netlist for this case. For the MAJ-based table, a conversion of the netlist of the comparator to a MAJ-based one is required. This process is the same as in Figure 5.12, but shrunk for

Figure 5.13: Flow employed to obtained the regular and MAJ-based tables, required for applying C2PAx.



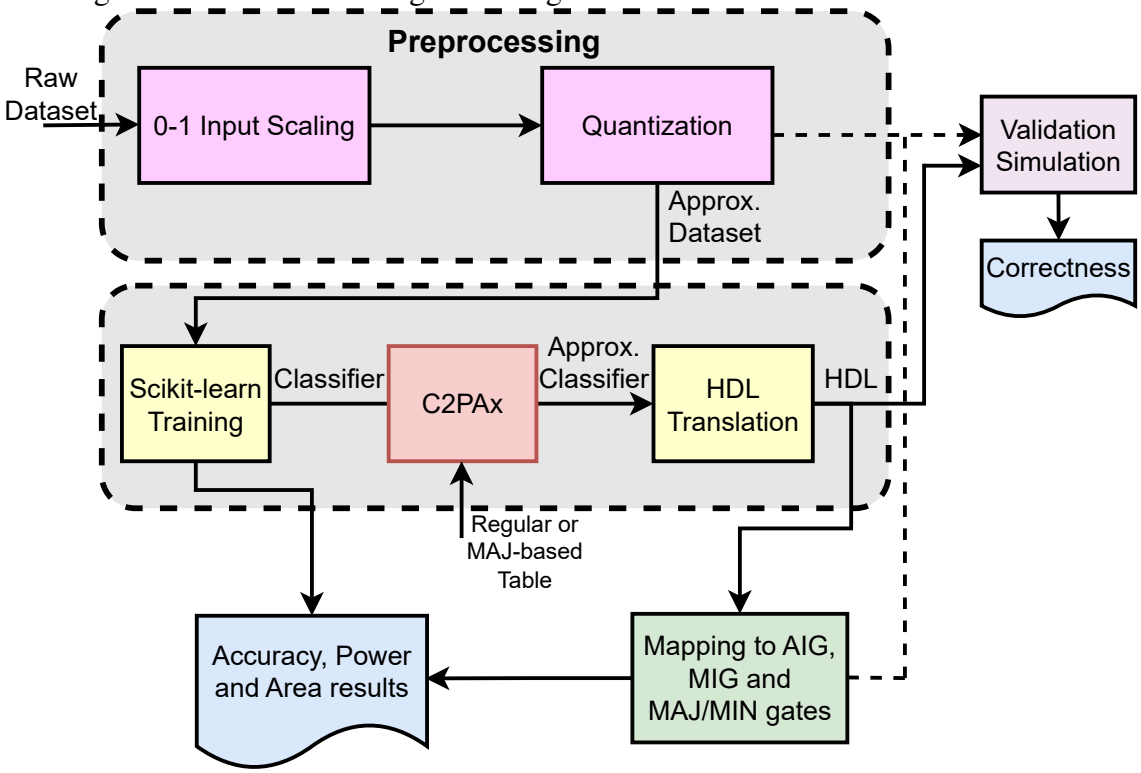
illustrative purposes.

The area of the MAJ-based circuits is obtained in terms of number of transistors. Even though MAJ-based circuits are reported to be more easily implemented in emerging technologies, a simulator for such systems was not found. Therefore, the number of transistors considered for the MAJ and MIN gates were the ones obtained from the author's previous work (ABREU et al., 2023), where the compact CMOS-compatible MAJ/MIN gates were proposed. In this previous work, the number of transistors for the MAJ and MIN gates are 6 and 8, respectively. This value, however, will not particularly matter for the table, given that what matters the most here is the relative area results between the different constants; hence, even if different number of transistors were considered, the relative results would not change significantly.

The power results are estimated based on SPICE simulations of the MAJ/MIN gates proposed in (ABREU et al., 2023), for different values of fanout. A script calculates the fanout of each MAJ/MIN gate, and applies the estimated value from the fanout table obtained from the SPICE simulations.

MAJ-based C2PAx is then applied to the DT/RF models, as in the flow from Figure 5.14. As it can be seen, the synthesis process is embedded within the generation of the AIG. This conversion of the regular DT/RF VLSI accelerator to a MAJ-based DT/RF is applied in the green square, and is also illustrated as a simplification of the flow from Figure 5.12. In this case, both the regular and MAJ-based tables were employed to generate the approximate RTL for the DT/RF model, to verify whether applying regular C2PAx or MAJ-based C2PAx is better when performing the conversion to a MAJ-based DT/RF later in the flow.

Figure 5.14: Modified design flow to generate results for MAJ-based DTs/RFs.



Hence, this proposal leads to two different contributions: (i) the analysis of the reductions in MAJ-based DTs/RFs using the C2PAx technique, and (ii) the analysis of the use of different area tables for applying the C2PAx technique in MAJ-based DTs/RFs.

5.4.2 MAJ-Based Pruning

The second proposal is the use of pruning techniques in MAJ-based circuits. So far, there is a scarcity in the literature regarding approximate computing involving MAJ-based accelerators. Hence, a flow that integrates the tools to generate such circuits, along with one that performs the gate-level pruning, is of utmost importance to pave the way for approximate computing for emerging technologies. Besides that, it is important to verify how pruning behaves under MAJ-based circuits, given that they are more democratic circuits, since the output will only change when most of the inputs decide for it.

The flow of this proposal is presented in Figure 5.15. The flow is similar to the one with regular pruning, with the difference that mapping to AIG, MIG, and technology mapping to MAJ/MIN gates is applied in the RTL description of the DT/RF model. Just as in the MAJ-based C2PAx proposal, this mapping is performed in the same way as in

the flow from Figure 5.12. The netlist is then simulated, to obtain the SAIF file, which is a requirement to apply the *probprun* technique in AxLS. In this case, there is no particular .lib file for the MAJ/MIN gates containing the power and area values. As mentioned previously, power is obtained by counting the fanouts of each gate and fetching that value from a table which was obtained via SPICE simulation, from the work in (ABREU et al., 2023). Area is obtained by counting the number of MAJ/MIN gates and multiplying it by the number of transistors of each one, based on that same work. After obtaining the SAIF file, pruning is applied for an amount of gates tested by the designer.

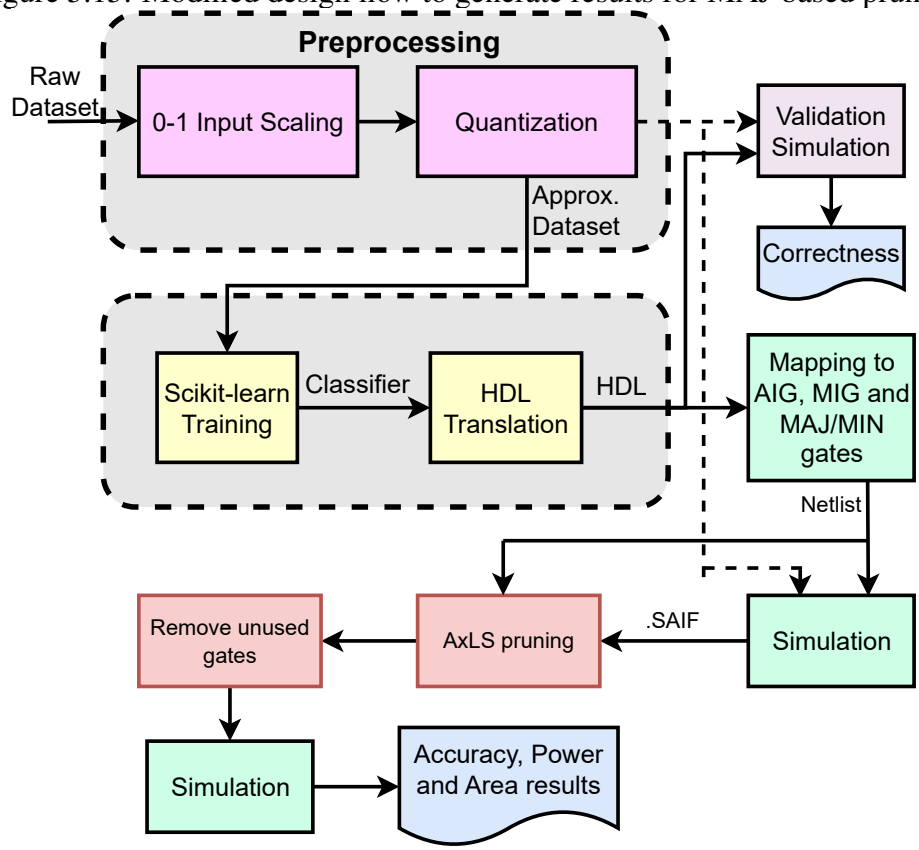
Due to the absence of a .lib file, synthesis is not performed after the process to optimize the circuit. In this case, a Python script is responsible for eliminating the gates that became useless after the pruning process. This is applied by verifying gates whose inputs were connected to MAJ/MIN gates that were eliminated. Some of these gates can be removed, in case at least two of their inputs become either '1' or '0'. This process is applied repetitively, since the elimination of some gates can always lead to other gates also being eliminated, until no gates can be removed. Moreover, the inputs of pruned gates are also verified to check for unused gates as well.

After this process, with an optimized pruned netlist, simulation is performed once again, to verify the accuracy results, along with power and area results through gate counting.

5.5 Chapter Summary

This chapter presented the proposals developed so far to achieve the goals of the thesis. A baseline framework for tree-based ML models is presented, which enables the automatic generation of tree-based VLSI accelerators. This framework is easily adapted to include approximate techniques, and three of them have been included: (i) input quantization, (ii) constant approximation, and (iii) indiscriminate and selective gate-level pruning. Additionally, approximations were performed for MAJ-based DTs/RFs. First, a proposal joining the C2PAx technique along with MAJ-based is presented. Then, a second proposal that integrates the generation of MAJ-based circuits with pruning techniques is proposed.

Figure 5.15: Modified design flow to generate results for MAJ-based pruning.



6 EXPERIMENTS, RESULTS AND DISCUSSIONS

This chapter presents details on the specificities of the experiments performed for the frameworks developed. First, the data sets employed for the majority of the analyses are detailed, as well as the power estimation methodology employed. Then, the results and discussions for every proposal are presented regarding the trade-off analyses performed.

6.1 Data sets Employed

Four data sets are used in this thesis: *accelAdl, activities, wearable*, and *wireless*. These data sets were selected based on the fact they are, to some extent, related to wearable applications with low power restrictions. The data sets originally have features in integer or floating-point representation, except for some (such as gender) that are represented with categories. The features were all converted to the same representation, as presented in Section 5.1.1, through the processes of remapping and quantization. Their characteristics are summarized in Table 6.1.

AccelAdl

This is a data set related to Activities of Daily Living (ADL) Recognition with a wrist-worn accelerometer. This is a collection of data recordings from accelerometers for simple ADL. This data set is composed of 14 activities, such as: brushing teeth, climbing stairs, combing hair, descending stairs, drinking glass, eating meat, etc. These activities are performed by 16 different volunteers and are obtained by a single tri-axial accelerometer attached to the right wrist of each volunteer. Four features are considered in this data set: the gender of the volunteer, and the x-, y- and z-axes values of the accelerometer.

Two works (BRUNO et al., 2012; BRUNO et al., 2013) consider this data set, which is available at (DUA; GRAFF, 2017). However, none of these works develop any kind of hardware architecture.

Activities

This data set is named Classification of Body Postures and Movements (PUC-Rio). It is composed of 5 classes (sitting down, standing up, standing, walking, and sitting) collected from 4 subjects.

This data set is available at (UGULINO et al., 2012), which also explains in more detail how the data was obtained.

Wearable

This data set corresponds to weight lifting exercises monitored with inertial measurement units, and it contains data from six young subjects performing a variation of biceps exercises.

This data set is detailed in (VELLOSO et al., 2013), but no dedicated hardware is presented in the work to perform inference based on this set. (MORTAZAVI et al., 2014) also uses this data set and mentions the proposal of a RF model. However, no architectures are presented and, therefore, no power comparisons can be performed.

Wireless

This is a data set focused on the classification of body postures and movements. This data set contains data from 14 older aged people between 66 and 86 years old. The set contains 4 classes: sitting on a bed, sitting on a chair, lying on a bed, and ambulating. The features include gender, age, height, weight, body mass index, and the x- and y-axes of all the four accelerometers used in each volunteer. Torres et al. (TORRES et al., 2013) have assembled this data set for public use; however, no architectures are proposed for the inference stage and no power results are presented.

Table 6.1: Characteristics of each data set.

Data set	# Instances	# Features	# Classes
AccelAdl	446471	4	14
Activities	165632	18	5
Wearable	39242	54	5
Wireless	75128	9	4

6.2 Power Estimation Methodology for ASIC Design Flow

The architectures already implemented for this thesis use a methodology based on works such as (SILVEIRA et al., 2017). This method uses a design flow that considers real-input vectors from the specific data set, which will be used to estimate the dynamic power dissipation more precisely.

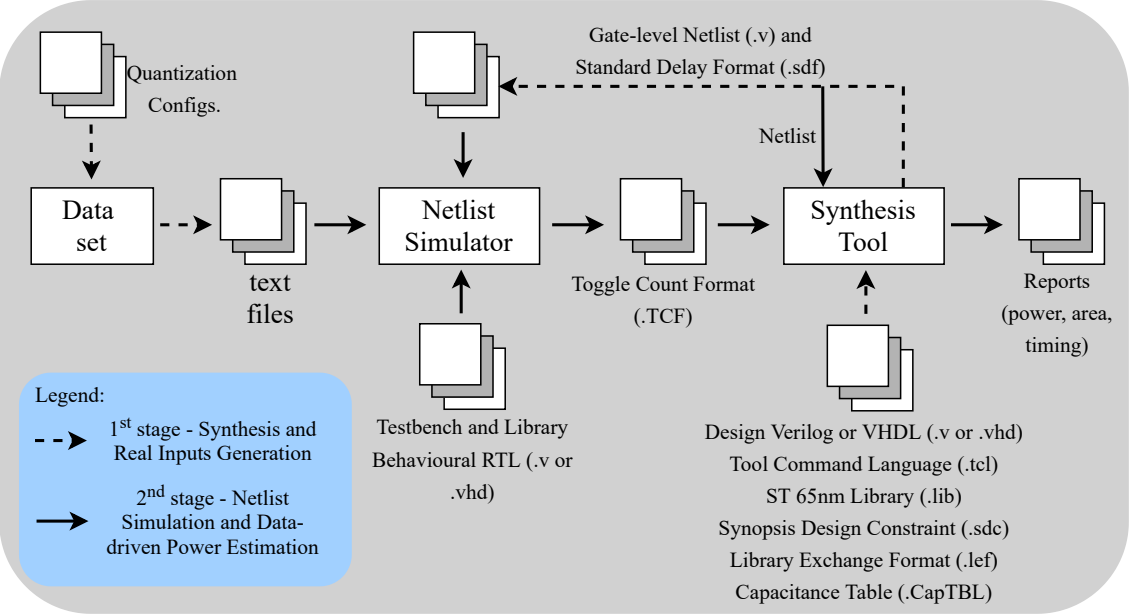
The dynamic power dissipation depends on the signal transitions that can occur in a specific digital circuit. Commercial synthesis tools, e.g., Cadence Genus Synthesis Solution (CADENCE, 2019), perform an estimation on the power dissipation based on standard switching activity – probabilistic default input values – when the designer does not provide any details on the inputs feeding the circuit. However, these estimations are pessimistic and rarely represent the real behavior of the circuit when a specific application is targeted.

Given that power dissipation is a major concern on embedded devices and obtaining power results is a primary goal of this thesis, an accurate method to obtain the power dissipation is required. Thus, designers will be able to represent, more precisely, the real behavior, allowing them to access more accurate results, which will make the optimization of power-hungry modules better.

Figure 6.1 shows the methodology used in the proposals of this thesis to obtain the power, area, and timing results. The flow begins with the description of the architectures in a hardware description language (HDL). In this work, the description is generated through automated tools described in Python language to ease the process, as presented in Chapter 5. In parallel, the data sets being used as case studies are parsed (based on quantization configurations) to extract the real-input vectors and use them along with the register-transfer level (RTL) description of the architectures, which will be synthesized. Every synthesis in this thesis, so far, uses Genus, which generates a netlist of the circuit, in Verilog hardware language. The resulting netlist contains elements from the standard cells library employed in the process. The architectures synthesized in this thesis use a 65 nm commercial standard-cell library from STMicroelectronics (STMICROELECTRONICS, 2021), at a 1.0V voltage supply. All the optimization effort flags were set as *high*. A Standard Delay Format (SDF) file is generated, which contains delay information for the gates and nets, along with area, power, and timing reports. After this first synthesis, a simulation tool, such as Irun, is used to simulate the generated netlist with a testbench file – which was also generated through the automated tool. This testbench reads the input

data from the data sets being analyzed in this thesis. This process generates a dump file, which can be in Value Change Dump (VCD) or Toggle Count Format (TCF) file formats (the works in this thesis use TCF files).

Figure 6.1: Methodology used to obtain accurate power results for the data sets.



After the simulation of the architecture and with the VCD or TCF file, a second process (data-driven power estimation) is performed with the netlist already generated. This second process generates the power, area, and timing results taking the real-input vectors of the data sets into account.

Additionally, Genus has a mode denoted as physically-aware layout estimation (PLE), which takes the gate interconnection penalties – in terms of area, power and timing – into account. This estimation is performed by estimating the lengths of the circuit nets and considering the capacitance effects when estimating power dissipation. However, the PLE process considers a relatively pessimistic layout routing estimation. Overall, routing is not considered in the syntheses performed in this thesis, so the estimation from the PLE process is the only information related to routing that was obtained.

The PLE analysis requires the use of Library Exchange Format (LEF) files. These files contain the physical layout information of the standard-cells library, which is essential to take the interconnections into account. The LEF macro includes the capacitance of the inner library cell, and the tech LEF comprises the process metal capacitance for the estimation of the interconnection capacitance. A capacitance table file (CapTBL) can also be used with the library, and it contains descriptions of the technology capacitances

in a fine-grained way, by considering process variations.

6.3 Framework: Tree-based RTL Models

This section presents the results of the architectures generated by the baseline framework, detailed in Section 5.1. The experiments were conducted with the default hyperparameters (parameters that control training algorithms) for generating the classifier except for the tree depth and, for RF models only, the number of trees in the forest. The values tested for tree depth were 1, 2, 4, 6, 8, 10, 12, and the ones tested for the number of trees were 3, 6, 9, and 12. The quantization (input width) values tested were 1, 2, 3, 4, 6, 8, 10, 12, 14, 16, and 32 bits. Every configuration was simulated for the four aforementioned data sets. Therefore, considering DTs and RFs, a total of 1540 VLSI accelerators were generated. Such a wide design space exploration was only possible due to the highly automated design flow proposed in this thesis.

First, some considerations regarding the operating frequency are presented. Then, the results regarding (i) classifier performance under different levels of quantization, (ii) approximations in the train and test sets, and (iii) analyses on power dissipation are presented. Finally, a comparison with related works is presented to verify the efficiency of the generated architectures w.r.t. the state-of-the-art.

6.3.1 Operating Frequencies

To synthesize the architectures, specific frequencies had to be chosen. If the architectures were run in their maximum frequencies, it would not be possible to fairly compare them in terms of power, given that each one of them has different maximum frequency values. In a preliminary work (ABREU; GRELLERT; BAMPI, 2020), the operating frequency was considered to be the maximum frequency of the worst-case scenario, which is the one for the maximum depth, and every other architecture was synthesized for the same frequency. This approach, however, does not consider the target application (wearable devices).

As mentioned in Chapter 1, the research performed indicates values never higher than 1kHz for the applications mentioned. Considering these reports, and taking into account that this work deals with accelerometers/gyroscopes or similar applications from

wearable devices, the required hardware frequency is not nearly as high as the maximum frequency achieved by the worst-case scenario. However, to maintain a wide margin above the limits of the devices found, it was decided to run the architectures at 10kHz. Even though no wearable devices have been found working at this frequency, it can be guaranteed that, even in this case, all of the architectures can properly work.

Moreover, even though frequencies higher than the required are being considered, the focus of the thesis is to analyze the trade-offs involving power. In other words, the goal is to achieve power reductions w.r.t. a given baseline while minimizing the reductions compared to a baseline accuracy, and not an absolute small power value without any reference.

6.3.2 Classifier Performance with Input Quantization

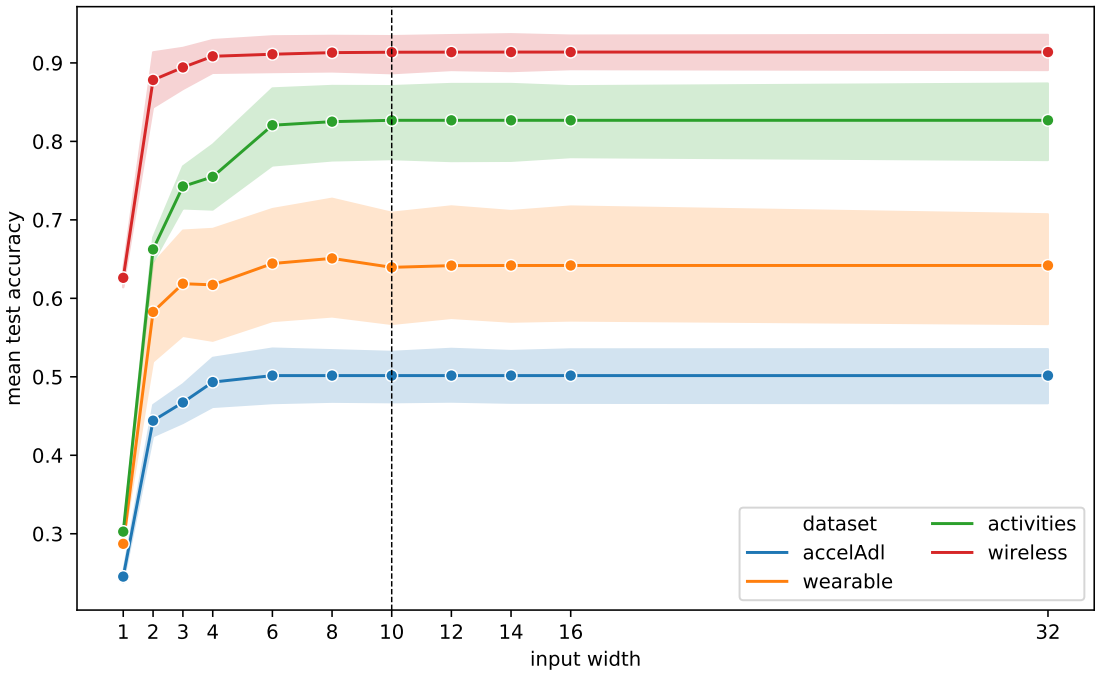
The first analysis of this work aimed at assessing how input quantization affects model performance. The results are presented in Figure 6.2, and each series represents the average and confidence intervals across the 1540 experiments. It is possible to observe that the test accuracy for all data sets converged with an input width of 10 bits, so larger values can be considered a waste of resources. Therefore, all the subsequent results of this section were limited to a 1-10 input width range. We can also observe in Figure 6.2 that the best accuracy was obtained with less than 10 bits for the wearable data set. This shows that input quantization is working as a regularization form, which is in line with other results in the literature (ZHOU et al., 2017; XU et al., 2018).

Although the goal is not to obtain a single best configuration in this thesis, it can still be observed that using bit widths larger than 10 is not advisable, as the increase in accuracy is negligible. In contrast, using less than 10 bits will likely incur in prediction loss, but this might still be interesting in applications with tight power constraints. This will be better discussed in the following analyses.

6.3.3 Train/test set Approximation Analysis

Since training is usually performed offline, one can still use precise (floating-point) data to build models, leaving the approximations (fixed-point, quantized) in the inference step only. However, using approximate data in both steps might improve per-

Figure 6.2: Input width (in bits) vs accuracy for all data sets (average across 1540 experiments).



formance since we are keeping similar operating conditions.

The following analysis aimed at figuring out whether the approximate (fixed point and quantized) feature values should be used in both training and inference stages, or if the precise (floating point) data should still be used for training. The results displayed in Table 6.2 show three configurations: (i) training and inference stages with precise data, (ii) training with precise data and inference with approximate data, and (iii) training and inference stages with approximate data.

Table 6.2: Average accuracy results for each data set, based on different training and inference approaches.

	Accuracy (%)		
Training Stage / **Inference Stage**	**Precise** / **Precise**	**Precise** / **Approx.**	**Approx.** / **Approx.**
AccelAdl	50.15	40.53	46.92
Activities	82.69	63.34	74.93
Wearable	64.19	53.55	60.07
Wireless	91.38	84.93	88.18
Average	**72.10**	**60.6 (-11.5)**	**67.5 (-4.6)**

As shown in the table, precise data for both training and inference leads to better results on average, although approximate models can actually perform better if quantization is not too aggressive, as will be discussed later. When approximate data is used in the

inference step only, the accuracy drops significantly by 11.5% on average, indicating that this approach may be harmful in practical applications. Finally, when approximate data is used on both stages, the performance drop is quite smaller (4.6%), and it may even lead to performance gains if features are not quantized too aggressively, as discussed in the following paragraphs. Therefore, the remaining discussions will only show results using approximate values on both stages.

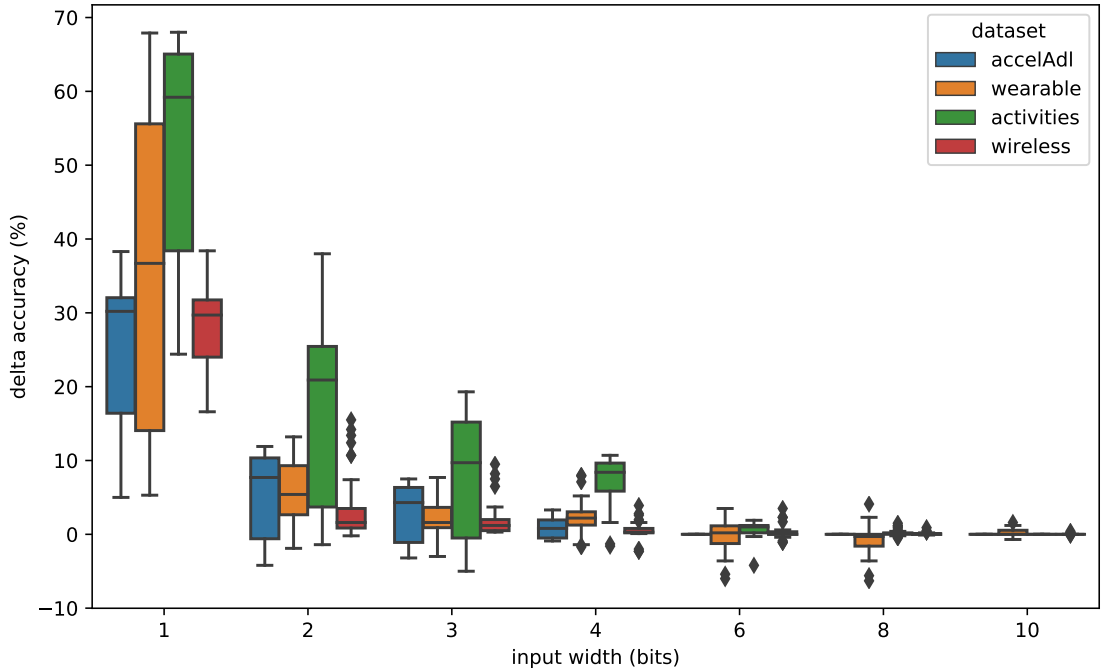
The average values from Table 6.2 are underestimated because they represent an average across several precision values, number of trees, and tree depths, and consider both DTs and RFs. All data sets can achieve higher accuracy results than this. It is also important to highlight that the accelAdl data set has 14 classes, so results above 1/14 (7.1%) can already be considered better than a random guess (provided the classes are balanced). Even so, the analyses presented in this thesis do not necessarily focus on obtaining high accuracy values, but on the possibility of finding power-accuracy trade-offs based on given baselines of power and accuracy.

The accuracy loss was also evaluated for different levels of input quantization, which is depicted in Figure 6.3. The loss was computed using as baseline the test accuracy obtained with the precise train/precise inference configuration and was averaged across all experiments for each data set.

As shown in the figure, using a single bit for each feature should be avoided, as it may lead to significant losses in accuracy, but this loss becomes smaller with each extra bit to the point where negative losses are observed. Negative values indicate that the models trained with approximate data performed better than the ones with precise data. As discussed previously, this happens because quantization acts as a regularization mechanism, reducing overfitting (ZHOU et al., 2017; XU et al., 2018). For all data sets, accuracy gains are observed starting with an input width of 6, and accuracy gains of up to 6.3% were obtained with 8-bit features. However, it is important to emphasize that the most suited quantization level depends on each application and must be carefully tuned. Therefore, we can conclude that using quantized, fixed-point units is ideal for tree-based circuits. The fixed point units greatly reduce resource consumption (area and power), whereas quantization works for both reducing resources and also for better accuracy.

With the feasibility of applying approximate computing in both training and inference, an analysis using confusion matrices was also performed. These results analyze the accuracy per class of each data set. The confusion matrix for the accelAdl data set is presented in Figure 6.4, for a RF with a tree depth of 10, a bit width of 10, and with

Figure 6.3: Variation in the accuracy when comparing the approximate model w.r.t. the precise model (in both training and inference stages).



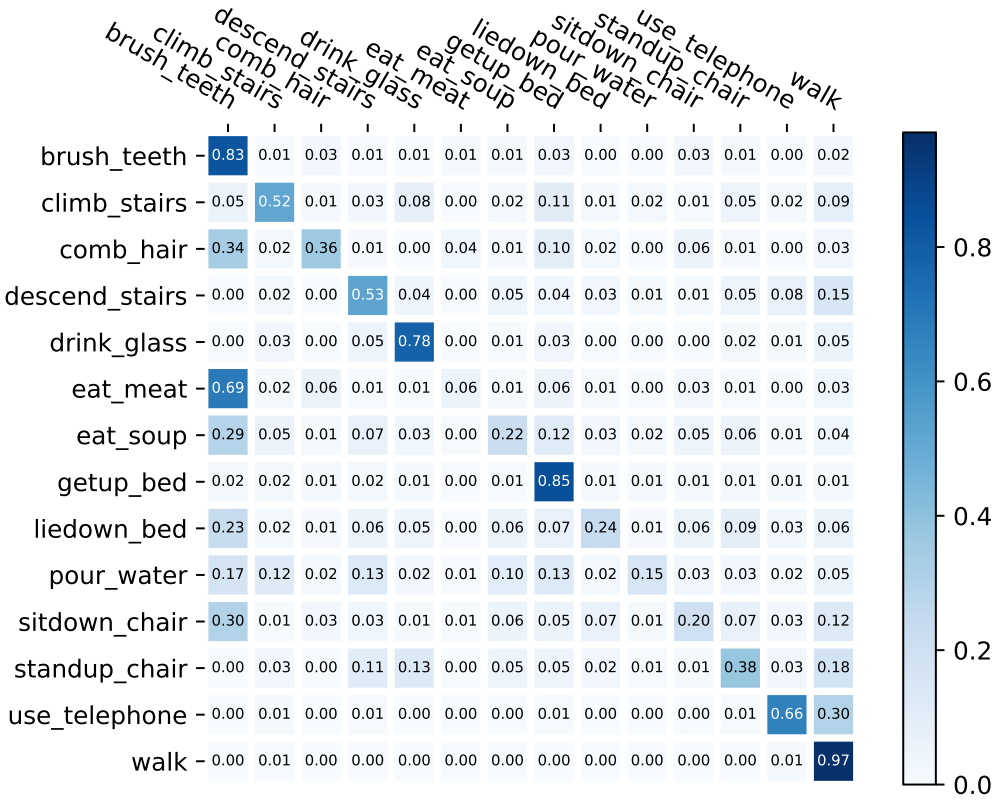
6 trees. This is the only data set analyzed here, for simplification purposes, and given that this was the most critical data set in terms of accuracy, and the one that contains the highest number of classes. The values in the diagonal of the matrix indicate the amount of correct predictions for each class.

As it can be seen from Figure 6.4, some of the classes presented very low accuracy values. For instance, the eat_meat class certainly was responsible for harming the overall accuracy (only 6% of the predictions were correct), since the model confused this action with the $brush\_teeth$ one (69%). Other actions performed by the subjects, such as $pour\_water$, $sitdown\_chair$, eat_soup, and lie_down, also presented accuracy values under 25%. These values explain the lower accuracy presented in Table 6.2 for the accelAdl data set. Even though the wearable data set also obtained a low average accuracy of around 60%, maximum accuracy results of around 97% were obtained.

6.3.4 Power Dissipation Analysis

As different parameters were tested (tree depth, number of trees, quantization level) and they all contribute to the final circuit consumption, the next step was to assess which ones affect circuit power and accuracy the most. Therefore, the results from

Figure 6.4: Confusion matrix for the accelAdl data set.



all 1540 syntheses were parsed and the correlation between each parameter with power and accuracy was computed. The power dissipation values were computed with the actual switching activity for each data set. The obtained correlation matrix is shown in Figure 6.5.

This analysis shows that tree depth and input width are highly correlated with power, but only the first one correlates with accuracy. The low correlation between accuracy and input width can be explained by the results previously presented. Therefore, if a designer has to choose between these two parameters to vary in order to reduce power without severely impacting on the accuracy, it is advisable to start with the input width.

Because the DTs and RFs can be imbalanced, we added the average (avg) tree depth as an extra parameter for analysis. As expected, this variable correlates better with the power dissipation and accuracy than the maximum depth. In addition, it can also be observed in Figure 6.5 that the number of trees (nb trees) has the smallest correlation with power and accuracy (17% and 4%, respectively) compared with the other variables. This is likely because larger RFs could not be synthesized due to limitations in the synthesis tool. When RFs with more than 12 trees were attempted, the synthesis would take an

unfeasible amount of time or crash due to Out of Memory (OOM) errors. Thus, the remaining results focus on the variation of tree depth and input width only.

Figure 6.5: Correlation between the parameters analyzed in this work.

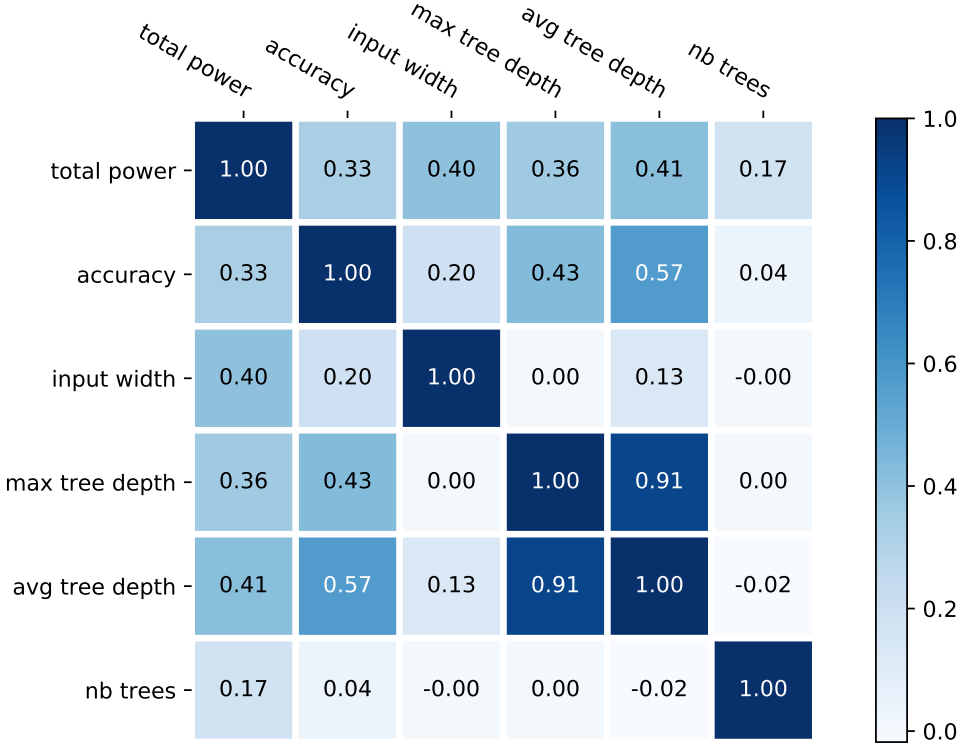
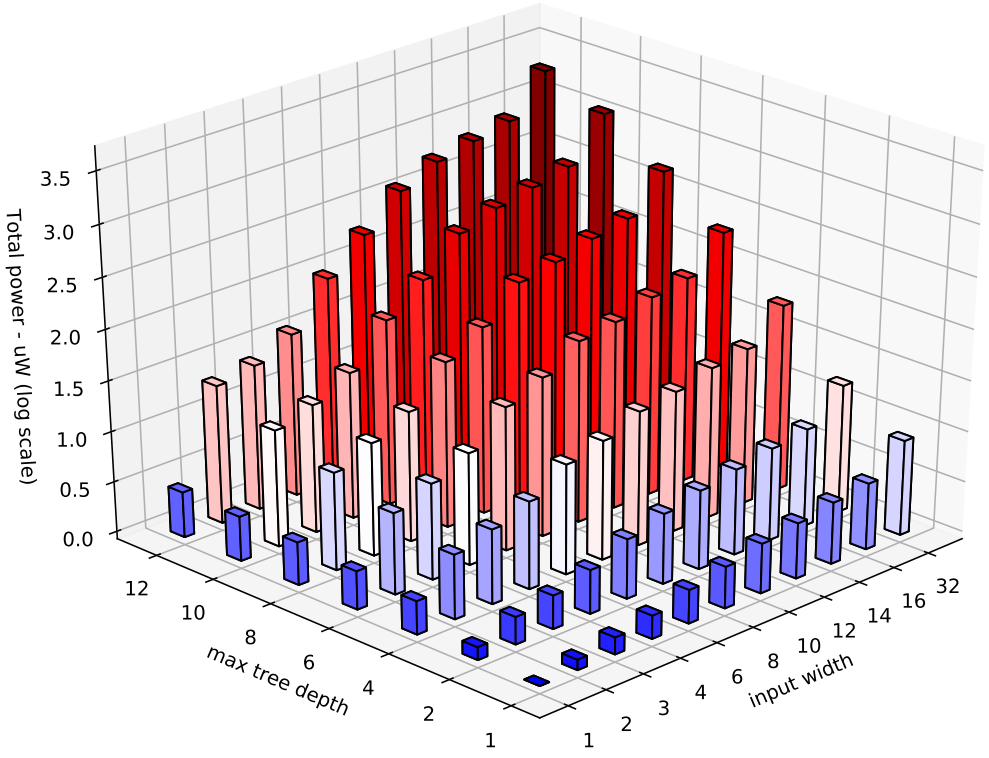


Figure 6.6 shows the impact of model complexity in terms of tree depth and input width on power dissipation. In this case, the power results presented are the averages between the DT model and the remaining number of trees (for the RFs). The analysis is presented in a log_{10} scale for better visualization, with the maximum tree depths of 1, 2, 4, 6, 8, 10, and 12, and the input widths of 1, 2, 3, 4, 6, 8, 10, 12, 14, 16, and 32. For example, a total power of 0 in this log scale indicates a power of $10^1 \mu$W, and a power of 3 indicates a value of $10^3 \mu$W. The results were also averaged across the four data sets considered in this work, for simplification purposes.

As Figure 6.6 shows, the power dissipation increases as both maximum tree depth and input width are increased. This is an expected scenario, given that, with the increase in tree depth, there are more comparisons between features and constants, as well as higher number of 'AND/OR' chains between the root and the output node. The input width is expected to increase power since each comparison will have more bits, even though the 'AND/OR' chains will remain the same. Also, the optimization performed to deal with categorical features (features with less than 20 unique values being set to a variable number of bits instead of the value set by Max_{width}) also influences the variation in input width. In the additional case of the number of trees, which was averaged here,

Figure 6.6: Power scaling analysis (in log scale) for different levels of input width and maximum tree depth. The data sets and number of trees were averaged.

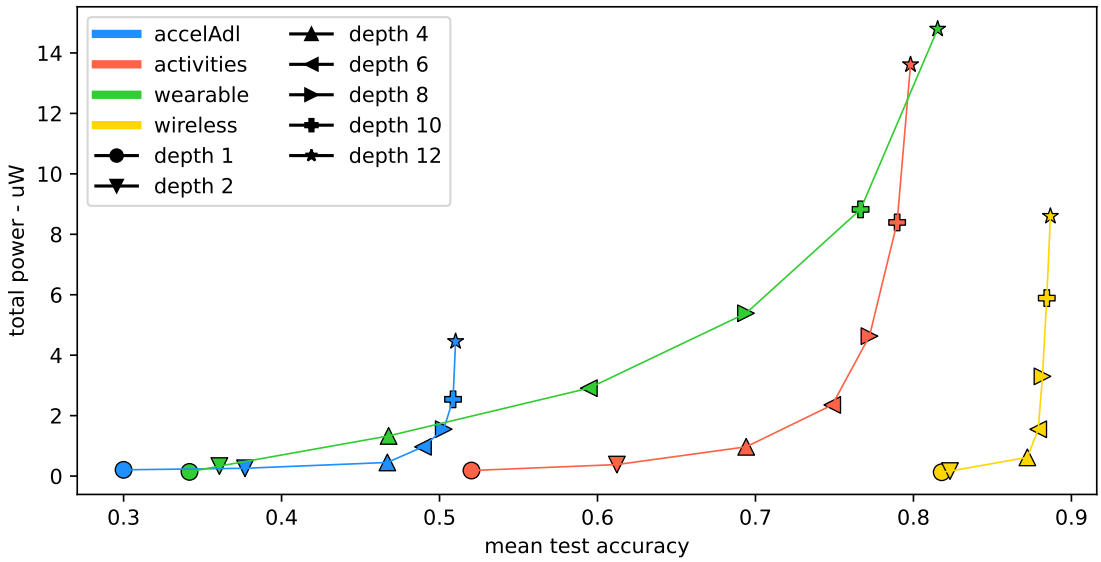


power will increase due to the additional trees and 'AND/OR' chains for each of them. Moreover, the increased number of trees will also increase the voter near the output of each architecture, given that more outputs need to be compared. The impact of the number of trees in power is softened by the fact that the redundant comparisons are optimized in this implementation due to redundancy removals from the synthesis tool and from the methods that remove the redundancies, described in Section 5.1. This is one of the reasons why the correlation between the number of trees and power dissipation from Figure 6.5 was smaller than the remaining tested parameters, besides the limitation of not using higher values for the number of trees.

The impacts of the maximum tree depth and the input width in power dissipation can also be verified from their increase ratio, by fixing one of them. When we compare the versions with a fixed input width of 32, the variation in tree depth from 1 to 12 increases power by about 238×. On the other hand, by fixing the tree depth to 12 and varying the input width from 1 to 32, we obtain an increase of 741×.

Figure 6.7 presents a power-accuracy trade-off analysis, made possible through the use of the proposed framework. The varying depths are represented by different sym-

Figure 6.7: Power-accuracy trade-off analysis of all data sets.



bols in the plot. The lines are presented for clarification purposes for each data set. The results were averaged across the different combinations of number of trees and input width (limited to 10). As the figure shows, power scales almost linearly with accuracy until a saturation point is reached. After this point, power continues to grow while accuracy remains practically unaltered. The main conclusion from this analysis is that it is important to include power estimations during model training. The training algorithms will usually build more complex models whenever possible, even if it leads to marginal accuracy gains at the cost of a significant increase in power consumption. Most cases reach a stable accuracy with relatively shallow trees, except for the wearable data set, showing that the proper values must be adjusted for each application.

Although an optimal configuration cannot be defined, as there are well-defined trade-offs presented in Figure 6.7, we can see that increasing the maximum depth does not tend to increase the accuracy significantly after a certain point, but power continues to grow in the same proportion. Therefore, one might still choose smaller depths to achieve less power consumption at the cost a small loss in prediction performance. This reaffirms the importance of a solution that can bridge the gap between the two developmental phases of training and synthesizing a model.

6.3.5 Comparison with related work

As discussed in Section 4.1, works that propose tree-based circuits that present power results are scarce, and the test conditions are usually difficult to reproduce (closed data sets, poor documentation, etc.), making it impossible to perform an ideal related work comparison. After an extensive search, DT architectures with characteristics similar to those of (BUSCHJÄGER; MORIK, 2018) and (KANG et al., 2018) were generated. The comparisons among these and the architectures from this thesis are shown in Table 6.3 considering the same target throughput in inferences/sec.

Buschjäger et al. (BUSCHJÄGER; MORIK, 2018) implement different models for DTs and RFs as well. However, the power results are not provided for the RFs when implementing it for an FPGA. Therefore, only comparisons with the DT models are performed. After contacting the authors, the data set used in their work (gamma events using First G-APD Cherenkov Telescope – FACT (BOCKERMANN et al., 2015)) was obtained. Their paper mentions the use of 12-bit precision, but the maximum tree depth is not documented, so a maximum depth was estimated based on the logarithm of the number of nodes (reported in the paper). In Table 6.3, the results for two DT models tested in (BUSCHJÄGER; MORIK, 2018) are presented, namely DNF (1) and Native (2). The DNF solution is one that performs all comparisons in parallel, and the Native one computes the inference in a more sequential way, through a repetition statement. Therefore, the DTs were trained with 12 bits of precision, and a tree depth of 11, using the same data set they employed. The paper also mentions the use of Zedboard containing an Artix-7 Z-7020 FPGA, but the actual FPGA device is not stated. The architecture was synthesized considering a frequency that would equal the number of inferences per second of prior works. In this case, the constraint was set to 1.1MHz for the logic synthesis. In Table 6.3, the top-level power consumption is presented. The power obtained for all the FPGA board was 82 mW. It is noteworthy that the operating frequency chosen in the synthesis of the generated architecture was not the maximum frequency possible in a 65nm CMOS technology. The low frequency 1.1MHz was just set to equalize the throughput of this work with the other architectures' results for comparison purposes, as throughput is directly linked to the application. However, even at any higher frequency for the VLSI architecture from this work, the energy/inference results would be the same, as the number of inferences/sec would then be orders of magnitude higher than the value in (BUSCHJÄGER; MORIK, 2018).

Kang et al. (KANG et al., 2018) design a RF accelerator for two data sets: Traffic sign recognition, using KUL Belgium traffic sign data set (PRISACARIU et al., 2010) and face detection using MIT CBCL data set (POGGIO, 2007). The maximum tree depth employed was 6 and the number of trees was 64, so we trained RFs with the same parameters. The work mentions the use of 8 classes from the traffic sign data set, and 2 classes from the face detection set. However, both data sets, when obtained, presented a higher number of classes than reported in the paper. For this reason, it was decided to pick the classes with the most train and test images. The images were resized for 16×16 pixels in gray-scale, similar to how they performed it, and a sub-sampling of 4:1 was also applied, and formed an additional data set from these pixel values (each pixel is a feature). The frequency was set to 364.4kHz, which is their throughput in terms of inference/second, for comparison purposes only, even though the architecture could achieve a higher frequency of operation. Considering their work implements a single chip for both data sets and the proposal in this thesis implements an architecture per data set, the power and area results presented from this thesis are from the worst-case scenario between both data sets. However, the results in Table 6.3 referring to (KANG et al., 2018) do not mention how much of the power dissipation is due to the RF kernel. It can be briefly inferred from the share that the RF kernel occupies in the proposed chip, which is about 10%. Therefore, a better power result of 0.71mW can be estimated instead of the 7.1mW reported in the paper, but this is also a rough estimation since in-memory computing was applied in that paper as well.

As mentioned in the previous paragraphs, the exact conditions of the previous works could not be perfectly reproduced. For that reason, there is no guarantee that the accuracy in the comparisons will be the same. Therefore, we had to resort to a fair approximation based on model complexity (i.e., same tree depths, input quantization, and number of trees). In doing so, we can safely assume that the power and energy results in this proposal are similar to those obtained with an ideal comparison, i.e. same accuracy, using the actual models presented by the related works.

Based on the results from Table 6.3, reduced energy/inference results are obtained w.r.t. both implementations in (BUSCHJÄGER; MORIK, 2018) and the one in (KANG et al., 2018). Despite the fact that the solution (BUSCHJÄGER; MORIK, 2018)-2 uses significantly fewer LUTs and FFs, this is compensated through the use of 5 BRAMs as reported in the table. Moreover, it is important to emphasize the fact that both solutions use a Zedboard that contains both the Artix-7 FPGA and an ARM processor. Hence, part

of the processing is also being performed in the ARM, which also can explain the low area in terms of LUTs in solution 2.

Table 6.3: Comparison with related works. The operating frequencies of the architectures from this thesis are set to equalize their throughput to the respective related work for DT or RF

	Work				
	(BUSCHJÄGER; MORIK, 2018)-1	**(BUSCHJÄGER; MORIK, 2018)-2**	**This Work**	**(KANG et al., 2018)***	**This Work†**
Technology	Artix-7	Artix-7	Artix-7 XC7A100T	65nm CMOS	65nm CMOS
Decision Tool	DT	DT	DT	RF	RF
Frequency (MHz)	100	100	1.1	1000	0.3644
Inferences/s	≈ 1.1M	≈ 1.1M	1.1M	364.4k	364.4k
Power (mW)	23	8	0.07	0.71	0.073
Energy/Inference (nJ/decision)	20.9	7.27	0.064	1.94	0.2
Area	9606 LUTs 3183 FF	31 LUTs 64 FF, 5 BRAM	3473 LUTs 536 FF	$0.1mm^2$	$0.078mm^2$

*estimated as 10% of the total power/energy/area
†worst-case power/area result of the 2 tested data sets

6.4 C2PAx

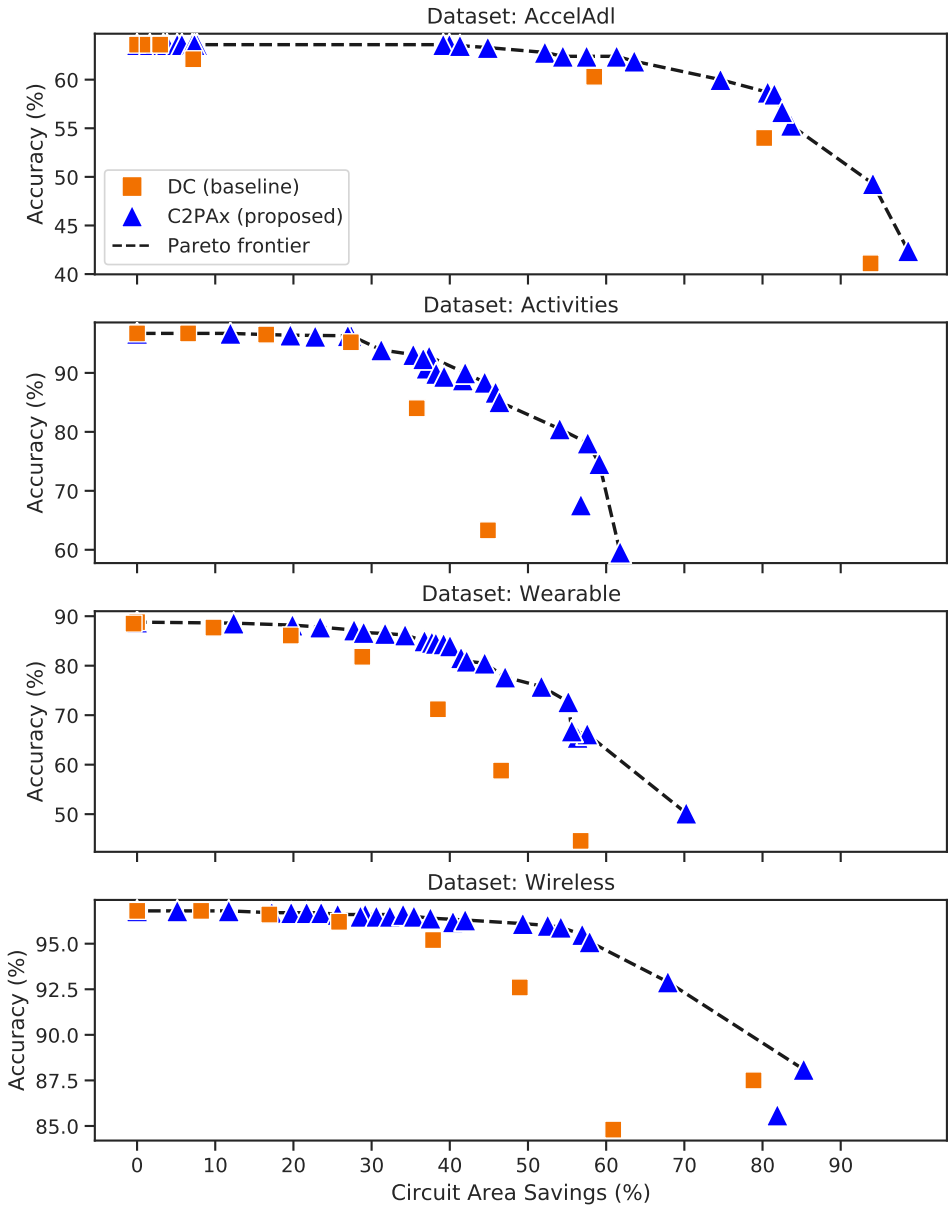
This section presents the results regarding the C2PAx proposal. The results are divided in four parts: first, the C2PAx proposal defined in Section 5.2 is explored, w.r.t. the DC (baseline) technique, in terms of their trade-offs between area and accuracy, as well as their accuracy tuning granularity. Then, after demonstrating the best technique to be further explored, C2PAx is analyzed in terms of energy consumption, area, and accuracy, with the variations in maximum tree depth, for the four data sets, for the RFs synthesized. C2PAx is also compared against the conventional model without any sort of approximation, unchanged from the Scikit-Learn library. After that, the C2PAx proposal is analyzed in terms of the critical path delay as a function of the step value. Finally, the use of generic (non-mapped) synthesis is analyzed.

For the following analyses, as mentioned in Section 5.2, 10-bit comparators were employed. For the DC method (baseline), DC values from 0 to 9 LSBs were tested. For the C2PAx method, the step values of 1 to 15, 20, 25, 30, 35, 40, 45, 50, 100, 150, and 200 were tested (along with the precise case with a step of 0). The analyses were conducted considering only RFs (with 5 trees), given that this is the most generic scenario, with maximum tree depths ranging from 6 to 10, considering the four data sets. This leads to a total of 720 architectures synthesized for the evaluations. The frequency was set to 50MHz, which is a value that can encompass other scenarios besides only the wearable contexts herein analyzed.

6.4.1 Area-Accuracy Trade-offs of the Adjustment Techniques

This analysis compares the DC and C2PAx techniques in terms of the Pareto frontier, to compare how they perform their area-accuracy trade-offs, for the four data set considered. This analysis is presented in Figure 6.8, in which each blue triangle represents a different step configuration for C2PAx, and each orange square denotes a different configuration of number of DC signals in the constants. The maximum tree depth was fixed to 10 for this preliminary analysis, and the results are presented for the RF models only, given that this is the more generic case of a tree-based method. The remaining analyses in the next sub-sections will present variations in the maximum tree depth.

Figure 6.8: Pareto fronts of every technique being analyzed, for the four data sets considered in this work. For this analysis, RFs with maximum tree depths of 10 were considered.



This analysis only showed the points that surpass 50% of the maximum accuracy achieved for each specific data set. This was chosen given that, below this limit, the DC technique is the only one that is presented, as it severely harms the accuracy, and such low accuracy values are not feasible for practical applications, given that such high accuracies can be achieved, as presented in Figure 6.8.

Moreover, we can observe the varied ranges of accuracy between the different data sets. For example, the Wearable data set presents accuracy values below 60%, whereas the Wireless data set remains mostly above 85%. This is due to the fact that the data sets have different numbers of output classes, as it was seen in Section 6.1, along with the different number of features and samples provided by each data set.

As it can be seen, the C2PAx proposal seems to dominate the Pareto frontier (i.e., the Pareto curve crosses most of the points that use C2PAx) for every data set. In some cases, it was not possible to find equivalent configurations in terms of accuracy for the C2PAx and DC, given that DC significantly reduces the accuracy. However, it can be noticed that the only values where the DC technique seems to obtain competitive results with the C2PAx proposal are very close to the precise cases (upper left of each plot). In such cases, the DC does not harm the applications that much. However, when the number of LSBs to apply DC signals increases, the accuracy severely drops w.r.t. C2PAx.

Besides that, the fact that DC does not allow for a fine-grained analysis as in C2PAx is another disadvantage. In this proposal, we can opt for any step value between 0 and 1023, whereas DC would only allow for values from 0 to 10. Therefore, C2PAx is applicable to more systems with varied energy/area constraints.

Some specific points using C2PAx are out of the Pareto curve. These specific cases may occur when modifications in the constants of the comparators may lead to a change in the outcome of specific trees in the RF for several samples in the test set, resulting in a change in the overall accuracy of the model.

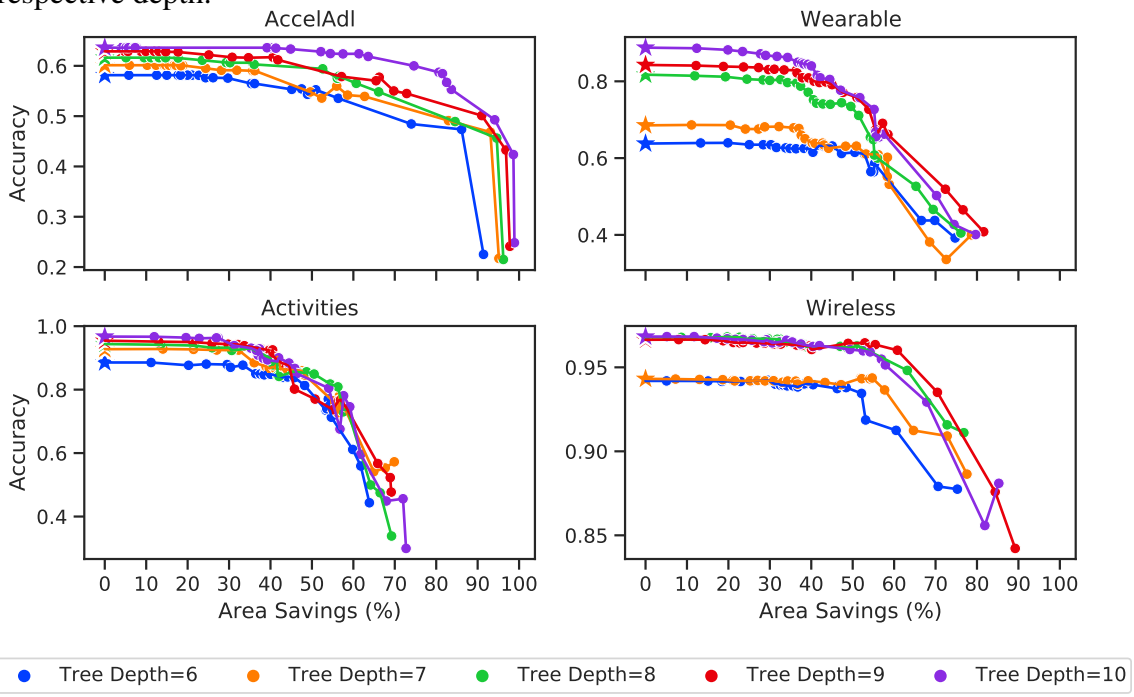
6.4.2 C2PAx Area- and Energy-Accuracy Trade-offs

This next analysis presents the area- and energy-accuracy trade-offs of the C2PAx technique. The curves for power savings versus accuracy are equivalent to the ones presented for energy savings versus accuracy, given that the clock frequency and throughput of every kernel being compared are the same, i.e., one inference per cycle. We present the analyses for RFs, for the four aforementioned data sets. Each of the plots contains the

points for the five different maximum tree depths (from 6 to 10), and varying steps.

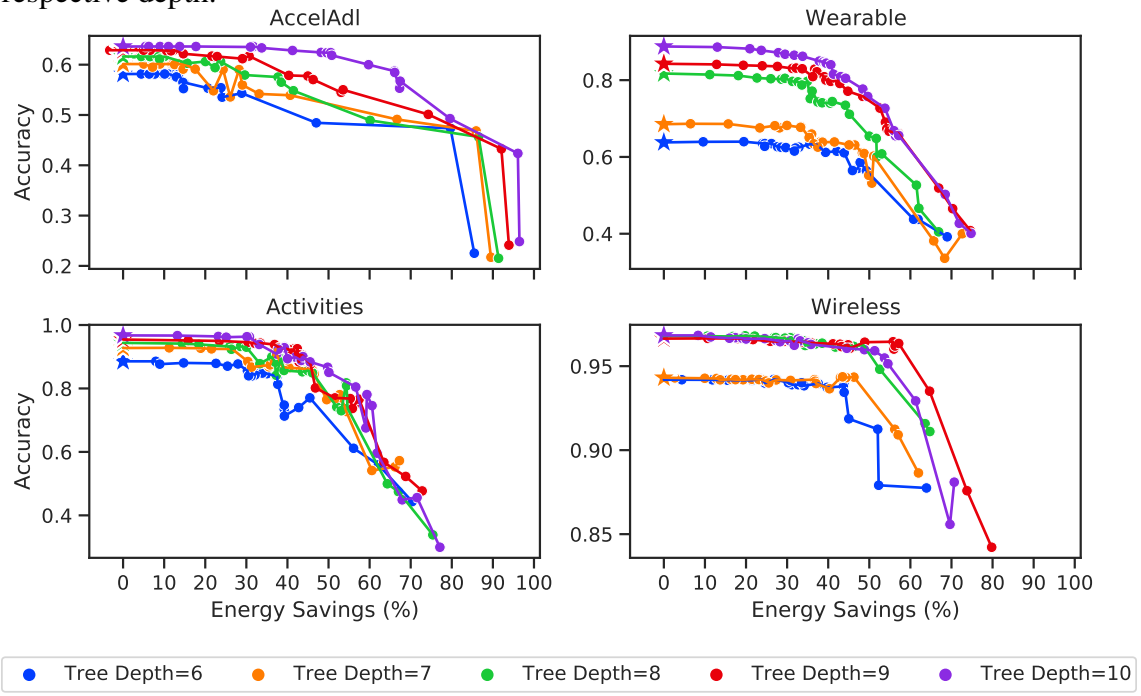
The first analysis defines the C2PAx technique as an approximation method regardless of tree depth variation, and it is presented in Figure 6.9 (for area) and in Figure 6.10 (for energy). Therefore, for each tree depth (represented in different colors/markers), the different approximation levels (using the aforementioned step values) are presented in the respective line of that same color/marker. The points marked as a star ($\star$) are used as the starting point and denote the precise RF with a step of 0, for that specific tree depth. The precise version for each tree depth (from 6 to 10) is used as the baseline for the remaining points (i.e., with steps higher than 0) of that respective line.

Figure 6.9: Accuracy as a function of the area savings, for the four data sets employed in this work, considering RFs with different levels of tree depth limitation. This analysis considers the savings of the step configurations w.r.t. the precise configuration of their respective depth.



Considering Figure 6.9, in general, we observed well-behaved and expected curves for all four data sets, in the sense that the accuracy does not seem to decay significantly in points close to the precise ones. The points where the accuracy starts to severely drop also depend on the data set being analyzed. In some cases, increasing the tree depth all the way to 10 did not lead to significant improvements w.r.t. smaller tree depths, such as in the Activities data set. However, the goal of this analysis is to verify the impacts of the C2PAx technique under different tree depths. Therefore, the most relevant result in this analysis is the relation between the baseline model (precise) of a specific depth and its

Figure 6.10: Accuracy as a function of the energy savings, for the four data sets employed in this work, considering RFs with different levels of tree depth limitation. This analysis considers the savings of the step configurations w.r.t. the precise configuration of their respective depth.



respective approximate points in the same line.

We can see that most curves seem to keep their accuracies similar to the baseline ones for small step values, while significantly saving area. The C2PAx proposal only seems to harm the accuracy results in larger step values, as expected. In these drastic cases, the accuracy may even increase for some steps when compared to a previous adjacent step value, as it can be seen in Figure 6.9 in the RF for the Wearable data set, for a tree depth of 7, in which the curve goes up in the rightmost point.

In some extreme scenarios, the accuracy of a point with a large step can even surpass its respective baseline ($\star$), such as in the Wireless data set for a tree depth of 7. This increase in accuracy with area savings of more than 50% may happen due to a decrease in overfitting, i.e., the changes in the constants of the RF led to a better generality of the model.

In general, the technique is clearly beneficial and presents high area savings for negligible accuracy reductions in most scenarios, for slight variations in the step value. For a tree depth of 10, we obtained area reductions of 25.88% with a negligible accuracy drop of 0.86%, on average, for the data sets. For area reductions of about 34.46%, the accuracy drops by only 2.92%, on average.

The behavior of the C2PAx technique in terms of energy is presented in Figure 6.10. This analysis is similar to the previous one, as it uses the same baseline (precise case $\star$). However, even though the savings are still significant, the results seem slightly more irregular than the previous ones from Figure 6.9. This is due to two reasons: (i) we used the area as an approximation metric, so the well-behaved scenarios in the previous analysis are more expected, (ii) power estimation (which was required to obtain the energy results) is not as easily determinable, as small variations in operators and in the inputs may change the leakage and dynamic powers. Despite the odd behaviors, we still managed to obtain significant energy savings with negligible accuracy drops.

The remaining set of results aims at analyzing the C2PAx proposal as a method to be used to decrease energy as a replacement to the variation in maximum tree depth. For example, if a designer were to use a precise version of this framework to decrease power/energy (due to limited contraints of their system), they could opt to decrease the maximum tree depth, as it would significantly reduce the number of comparators. However, using the C2PAx technique in varying degrees while maintaining the tree depth could achieve more savings, for the same accuracy, when compared to decreasing the tree depth.

Figure 6.11 presents this analysis for the area savings, for the four data sets, considering RFs. As it can be seen, for many cases, we can achieve larger savings for tree depth x with the C2PAx technique, with very similar or even higher accuracies when compared to the precise scenarios with depth $x - 1$. This is clearly seen, for example, in the RFs with a depth of 10 and 9 in the Wearable data set (purple and red lines, respectively). Instead of decreasing the tree depth from 10 to 9 to increase the area savings, it is better to apply the step technique and maintain the tree depth of 10, as there are points in the purple line that surpass the baseline point of tree depth 9 in both accuracy and area savings.

In some even better scenarios, we can obtain savings better than two tree depths below, as it can be seen from the RFs in the Wireless data set for a tree depth 9 (in red), which has points with better values than the precise case of depth 7 (in orange). Therefore, for most cases, the proposal is better than applying changes in the maximum tree depth of the tree by a significant margin, in general.

The energy-accuracy plot in Figure 6.12 shows a similar behavior. For most configurations, higher tree depths with the C2PAx technique can surpass precise smaller depths in both accuracy and energy savings.

Figure 6.11: Accuracy as a function of the area savings, for the four data sets employed in this work, considering RFs with different levels of tree depth limitation. This analysis considers the savings w.r.t. depth = 10.

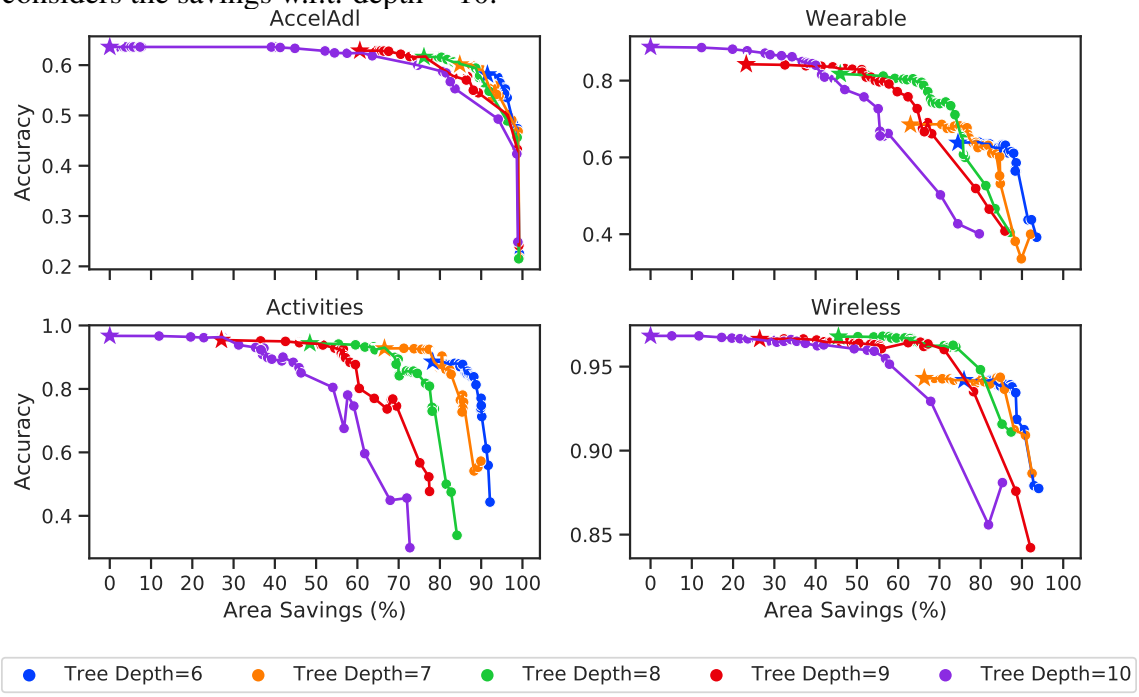


Figure 6.12: Accuracy as a function of the energy savings, for the four data sets employed in this work, considering RFs with different levels of tree depth limitation. This analysis considers the savings w.r.t. depth = 10.

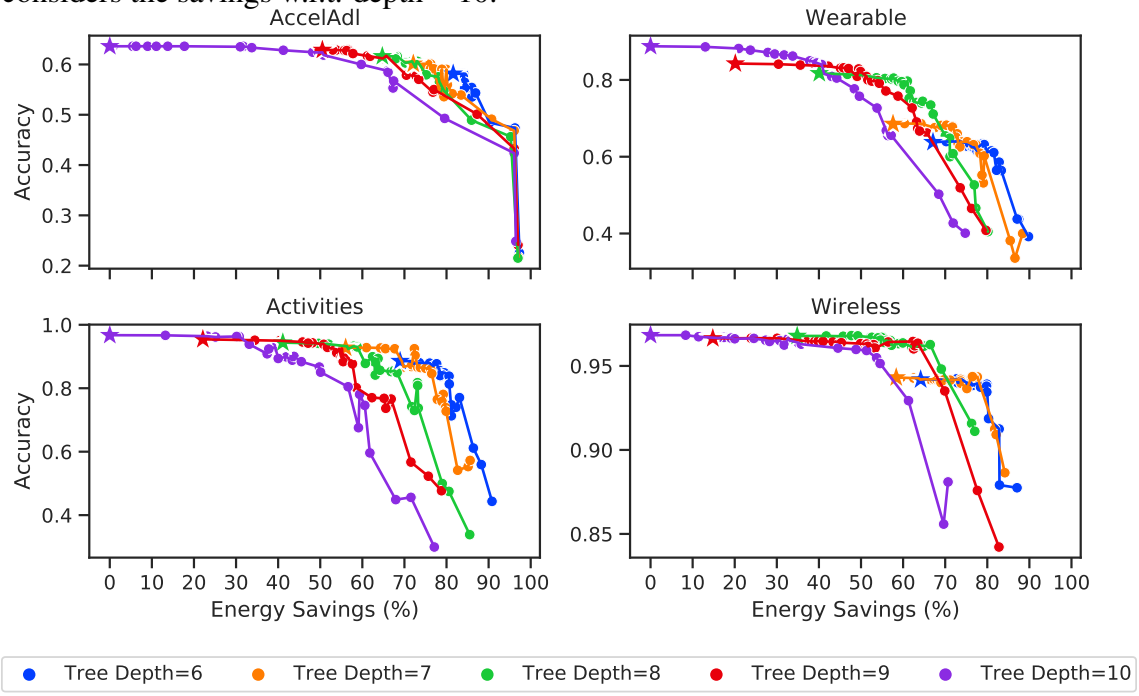
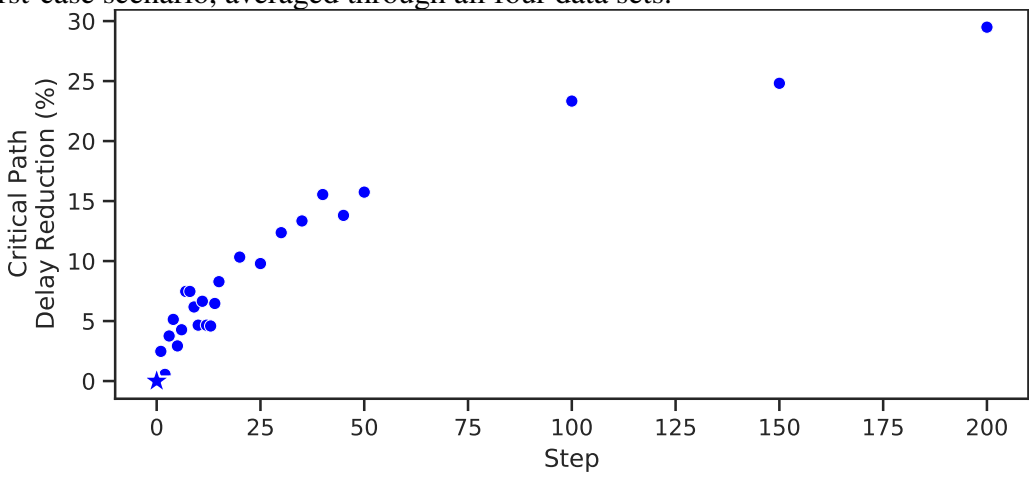


Figure 6.13: Critical path delay reduction w.r.t the precise model for different C2PAX step values. This plot is presented for RFs of maximum tree depth of 10, which is the worst-case scenario, averaged through all four data sets.



6.4.3 Critical Path Delay

Analyzing the impact of the C2PAx technique in the critical path is also important. Even though no stringent timing issues arise for this application (as wearable applications considered in this work do not require high computing throughput), there might be specific data sets for which higher clock frequencies are required. In these cases, decreasing the critical path delay (CPD) is desired, given that there will be more room for increasing the maximum clock frequency.

Figure 6.13 presents the decrease in the CPD as a function of the step, averaged across the four data sets. This analysis was considered for the worst-case scenario only, which is an RF configuration with a maximum tree depth of 10 (input width and number of trees remained the same as in the previous analyses, 10 and 5, respectively), for the maximum frequency of each step configuration.

To estimate the maximum frequency, we synthesize the architectures for a very small period that will certainly be smaller than the CPD. After obtaining the difference values from the synthesis tool (i.e., slack timing), we sum the absolute value of the slack timing to the old period we synthesized it to, and perform the synthesis process again. The slack timing refers to the difference between the target period and the arrival time. Therefore, positive slack values denote a possibility to increase the frequency. The process goes on until we obtain slack timings of 0 or very small positive values.

As it can be seen, the step always reduces the CPD, on average, when compared to the baseline model of step 0. A decrease in the CPD is the most intuitive scenario, given that the adjustment of the constants aims at decreasing the area of each comparator. This

will lead to a higher probability of fewer gates being involved in the CPD.

On the other hand, some results present an average increase in the CPD when compared to smaller step scenarios (higher than 0), even when we attempt to decrease the comparator size. This behavior is mainly due to two reasons:
1) The constant comparator offline synthesis does not measure the exact timings required for each comparator when we use them in the context of the overall RF accelerator. This may be due to differences in the timing of the RF accelerator and the standalone comparator that may lead to different operators or due to different fanout configurations (the standalone comparator synthesis is unaware of the number of gates to which it will be connected in the decision paths);
2) The optimizations performed by the synthesis tool regarding redundancies removal might lead to differences in the logic circuits of the comparators.

Although we attempt to remove some redundant comparators from the several trees in the RF with the framework, as detailed in a previous work from the author (ABREU; GRELLERT; BAMPI, 2022), most of it is still performed by the synthesis tool. Hence, changing the comparators may not always lead to a decrease in the CPD.

6.4.4 Employing Generic Synthesis

The C2PAx technique is agnostic to the mapping and can be generically implemented to any technology node. We demonstrated in the previous sub-sections the results for a non-mapped comparators cost table to demonstrate this genericity.

The purpose of this analysis is to demonstrate that using the generic option is a viable approach, as opposed to the more dedicated way of synthesizing and mapping the isolated comparators for the specific standard-cells library. More specifically, in the non-mapped scenario, we synthesize the isolated comparators without mapping (as was done in the previous analyses) and apply the C2PAx technique – which analyzes comparators in the vicinity of the precise one – to each comparator of the architectures being analyzed. This architecture is then synthesized and mapped to a standard-cells library so that more precise area, power, and critical path delay results can be gathered. In the mapped scenario, the difference is that we synthesize the isolated comparators mapping it to a standard-cells library, and use these mapped results to estimate the constants to which the comparisons will be adjusted by the C2PAx technique.

For this analysis, we only considered the C2PAx technique as well, given that

the baseline DC technique does not rely on previous isolated synthesis – it delegates the choice of operator directly to the synthesis tool.

This experiment was conducted for every tree depth as well. Considering that we have a total of 26 different steps being tested (including the baseline with step 0), for RFs, for the 5 different tree depths and the four data sets, we have a total of 390 architectures tested with the mapped analysis.

It was found that only 28 out of the 390 architectures presented differences in their synthesis results. Out of the 28 different results, 20 of them were from the Activities data set, 4 from Wearable, and 4 from Wireless. These variations are due to small differences in the relations between the best comparators in both the mapped and non-mapped approaches, which may manifest in specific comparisons. Besides the fact that there is only a small number of different results, the differences in these results were negligible: the maximum accuracy increase was 0.0183% for the mapped approach with an average power increased by 0.35%, and area increased by 0.12%, when using the mapped approach. There were small variations in both power and area between these approaches, depending on the specific tree-based configuration.

In addition to these negligible differences in accuracy, power, and area, using mapped comparators is dependent on the results from the specific standard-cells library. Therefore, for the sake of generality, the decision to use the non-mapped comparators is the most suitable one, as it represents an insignificant difference.

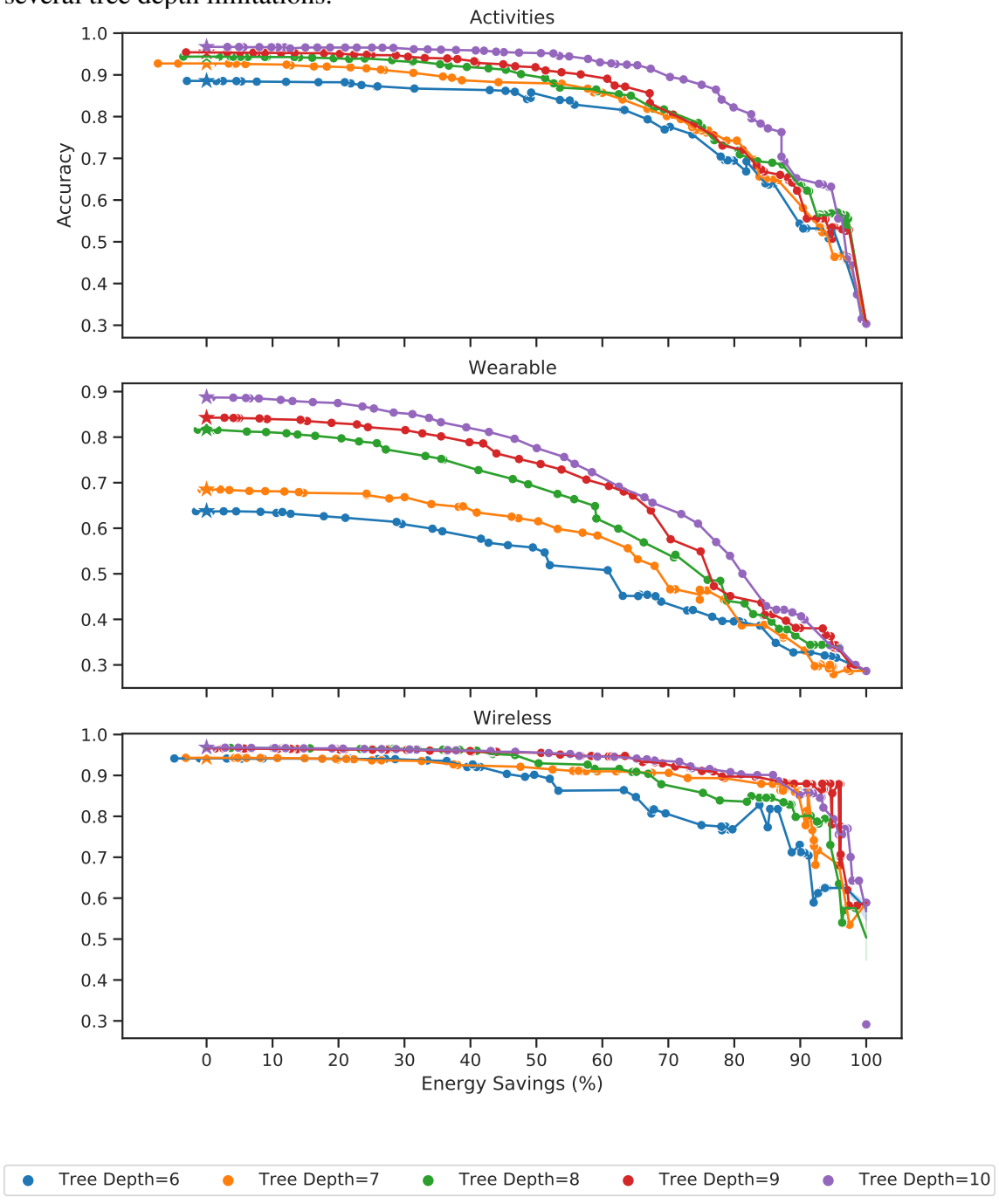
6.5 DTs/RFs with Gate-Level Pruning

6.5.1 Indiscriminate Pruning

This section presents the synthesis results of the use of the indiscriminate pruning technique from the AxLS tool (*probprun*) in the RF accelerators. For this thesis, it was considered that the nodes are eliminated from the original netlist based on a percentage of the total number of nodes. Pruning percentages from 0 to 100% were tested in steps of 1%. A variation in the maximum tree depth was also considered, from 6 to 10. Considering all the three data sets along with the variation in these other parameters, a total of 1515 architectures were synthesized. The frequency was set to 50MHz.

The first set of results is presented in Figure 6.14 and reveals the energy savings as a function of the model accuracy, for the configurations tested. The power and en-

Figure 6.14: Energy savings vs. inference accuracy, for the pruning levels tested, for several tree depth limitations.



ergy savings are the same, given that the operating frequency and the throughput do not change. In the charts, the precise models are presented as a star ($\star$). We can see that the use of pruning at the netlist level allows us to obtain fine-grained approximations in the architectures, generating a wide range of possible results in terms of the trade-off between energy and accuracy.

Given that we are dealing with trade-offs, the most suitable solutions for a spe-

cific system depend on the battery constraints and the accuracy drops tolerable. The area results are omitted here, given that the plots would be almost redundant. However, some slightly irregular behaviors occurred in the energy plot in Figure 6.14 (that did not occur with the area), considering that some pruned models obtained results with higher energy consumption when compared to the baseline precise models. This behavior only manifests for small percentages of pruned nodes (it can be seen in the Activities data set for a tree depth of 7, for instance), and it occurs due to the fact that the resynthesis of the pruned netlist may slightly modify some gates of the architecture, and these changes may lead to higher switching activities in these gates.

For every tree depth configuration of each application, the area and energy consumption results were already zero before pruning 100% of the nodes. This is due to the fact that the gates that led to the outputs were never the last ones to be pruned. Thus, even though there are still gates instantiated in the pruned netlist, none of them connect to an output, so the synthesis tool simplifies the circuit. In such cases, the accuracy is never zero, given that the fixed values for the outputs also lead to a specific classification.

Table 6.4 presents the overall results under several target energy savings, from 10% to 80%, for the tree depths of 6 and 10, for simplification purposes. This analysis was performed considering that different applications with different battery resources would opt for different approximation levels. In this analysis, the data sets were averaged. The baseline models for computing the energy and area savings, the accuracy losses, and the slack increases were the precise models without any pruning applied.

The rows denoted as real savings are presented to verify the actual savings of the approximate model that was the closest to the desired target energy savings. As it can be seen, due to how fine-grained the pruning technique is, the real savings were always close to the target savings. Moreover, these results could be even closer if we opted to prune with a step smaller than 1% between the approximation levels.

We can see that the area savings are higher than the energy savings, in relative terms. This is due to the fact that the AxLS tool prioritizes pruning in gates that have the least switching activity, hence the impacts in power/energy are smaller. However, no verification on the size of the gates is performed, so nodes of any size (small or large) with small switching activity will be pruned, hence saving more relative area than energy.

The slack timing increases of the approximate solutions were also computed, for the critical paths. As it can be seen, the slack timing, on average, were always increased for these scenarios, in a rate similar to the energy and area savings.

In general, we can obtain significant energy savings for small or negligible accuracy drops. On average for all tree depths, we can reduce energy by about 20%, for instance, for accuracy drops of only 0.68%. If we increase the energy savings constraints for about 40%, accuracy losses are still below 4%, on average.

Table 6.4: Overall accuracy drops, area savings, and slack increases of the configurations for several energy savings targets.

		Target Energy Savings*							
		10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**
Tree Depth = 6	**Real Savings**	10.31%	21.55%	32.14%	42.00%	50.10%	64.47%	72.32%	80.24%
	Accuracy Loss	0.26%	0.55%	1.98%	3.47%	6.76%	11.05%	16.01%	20.32%
	Area Savings	19.71%	27.55%	41.40%	50.06%	61.08%	72.36%	79.39%	86.99%
	Slack Increase	9.54%	14.75%	20.61%	25.01%	33.49%	39.79%	45.73%	55.83%
Tree Depth = 10	**Real Savings**	10.71%	20.45%	30.85%	41.14%	54.44%	62.10%	70.80%	81.92%
	Accuracy Loss	0.28%	0.81%	2.05%	4.20%	8.20%	10.52%	17.59%	22.74%
	Area Savings	16.64%	26.71%	39.84%	50.34%	63.34%	69.60%	78.05%	86.98%
	Slack Increase	24.08%	39.35%	62.92%	54.29%	80.71%	108.39%	138.41%	165.98%

*The baseline configuration for savings calculation is the one without any pruning, for each respective tree depth.

The proposal of iterative pruning was also briefly analyzed. However, the results did not differ significantly from the regular pruning approach. An enormous amount of pruning iterations are possible, for each number of nodes that one initially desires to prune. Hence, even though some specific combination of iterative pruning may present better reductions w.r.t. to regular pruning, the exact combination is likely dependent on the data set. Moreover, this would require an unfeasible amount of synthesis processes to exhaustively analyze and find a combination with significant reductions. This was not found for initial analyses performed with iterative pruning. For these reasons, this approach was not deeply analyzed.

6.5.2 Hierarchical Pruning

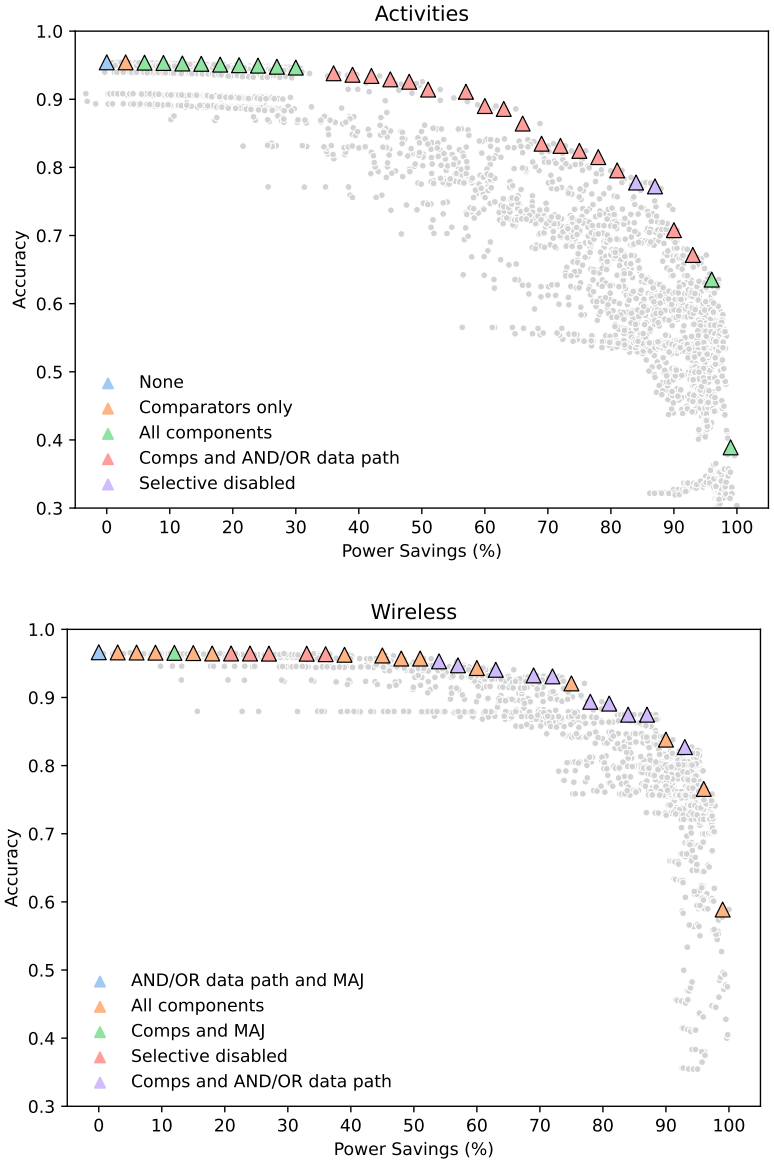
This section presents the results for the hierarchical pruning approach in RF accelerators, using the implemented framework. For this proposal, the nodes are also eliminated based on a percentage of the total number of nodes, but this time the total number of nodes is relative to each module. An analysis using the regular AxLS with indiscriminate pruning was applied from 0% to 100% of the gates, in steps of 1%, to serve as a baseline for comparison.

In order to allow for a very detailed analysis on the effects of using hierarchical pruning by running several combinations, only the Activities and Wireless data sets are presented in this analysis. For the hierarchical pruning, the three different modules were

pruned from 0% to 100% in steps of 5% each. Additionally, to increase the analysis in configurations with small pruning percentages, the grain of pruning for small percentages was increased from 0% to 10% in steps of 1%.

The results of the RFs for both the Activities and Wireless data sets are presented in Figure 6.15. Here, power savings are presented, but the energy savings are equivalent. The baseline, in each scenario, is the configuration with 0% of pruning. The colored triangles are the data in the Pareto curve.

Figure 6.15: Pruning results for both indiscriminate and hierarchical pruning of RFs for the Activities and Wireless data sets.



The results from Figure 6.15 clearly show the number of possibilities that can be obtained with the hierarchical pruning. Furthermore, the results show that significant savings can be obtained for the same accuracy w.r.t. the baseline model, especially in the

range of about 50% and 70%, respectively, for both data sets, which is where the baseline model starts to decay significantly.

It is important to notice that several points are in the lower portion of the plots, indicating low accuracy for varied energy savings. It occurs in the RFs given that small changes in the voter module can lead to the optimization of the rest of the circuit. Besides, even for small pruning percentages in the voter, other prunings will have occurred in both comparisons or AND/OR gates, which may be responsible for optimizing and pruning the voter.

Overall, it is clear that the hierarchical pruning, besides having to test more points, is more present in the Pareto curve when compared to the indiscriminate pruning. This is clearly seen by the colors of the plots in the Pareto and their respective legends, as seen in each plot. This pattern is maintained for both data sets. Even though the gains are less significant for small pruning percentages (since most versions of the circuit are still working properly), the smaller decay in the plot for the hierarchical pruning is evident for both data sets. For the Activities data set, only two points of the Pareto curve are from the indiscriminate pruning (selective disabled), and the Wireless data set has only five. The remaining points are all combinations of pruning in the three different modules, which shows that employing hierarchical pruning is a better alternative for DTs/RFs.

6.6 Approximations on MAJ-based Decision Trees and Random Forests

This section presents the results for the two approximation proposals presented in Section 5.4, regarding MAJ-based DTs/RFs. The first proposal is the adaptation of the C2PAx proposal for MAJ-based DTs/RFs, and the second one is a pruning analysis on such models.

6.6.1 MAJ-based C2PAx

The C2PAx proposal for MAJ-based circuits was applied in two different ways. As mentioned in Section 5.4, one can decide to either use the regular or the MAJ-based table to change the constants of the comparators, and this analysis will show the effects of applying each of them. The step values tested for each table were the same as the first C2PAx analysis from Section 6.4: 1 to 15, 20, 25, 30, 35, 40, 45, 50, 100, 150, and 200,

along with the precise case with a step of 0.

The Pearson correlation coefficient between the two tables was calculated and the value found was 93.04%. Hence, both tables present a very similar correlation, which means that the change from a regular standard-cell netlist to a MAJ-based netlist in the isolated comparators do not lead to significant relative changes in the cost of the comparators.

Figure 6.16 shows the comparison in terms of power-accuracy trade-offs between the use of the two tables in MAJ-based DTs/RFs. The analysis is presented for the four data sets, for RFs with 5 trees, with tree depths from 6 to 9. The baselines (i.e., the configurations with a step of 0) are presented with a $\star$, just as in the analyses from Section 6.4. The results for the area-accuracy trade-offs are not presented here given that the plot formats do not present any significant differences w.r.t. the power savings.

As it can be seen from Figure 6.16, aside from specific cases, applying either table does not lead to significant changes in power savings. In these specific scenarios, both solutions can achieve the best trade-offs. Hence, it can be concluded that, even though applying a MAJ-based table for MAJ-based DTs/RFs can lead to slightly better results in some points, maintaining a regular table for MAJ-based trees is still a good approach in terms of the power-accuracy trade-off.

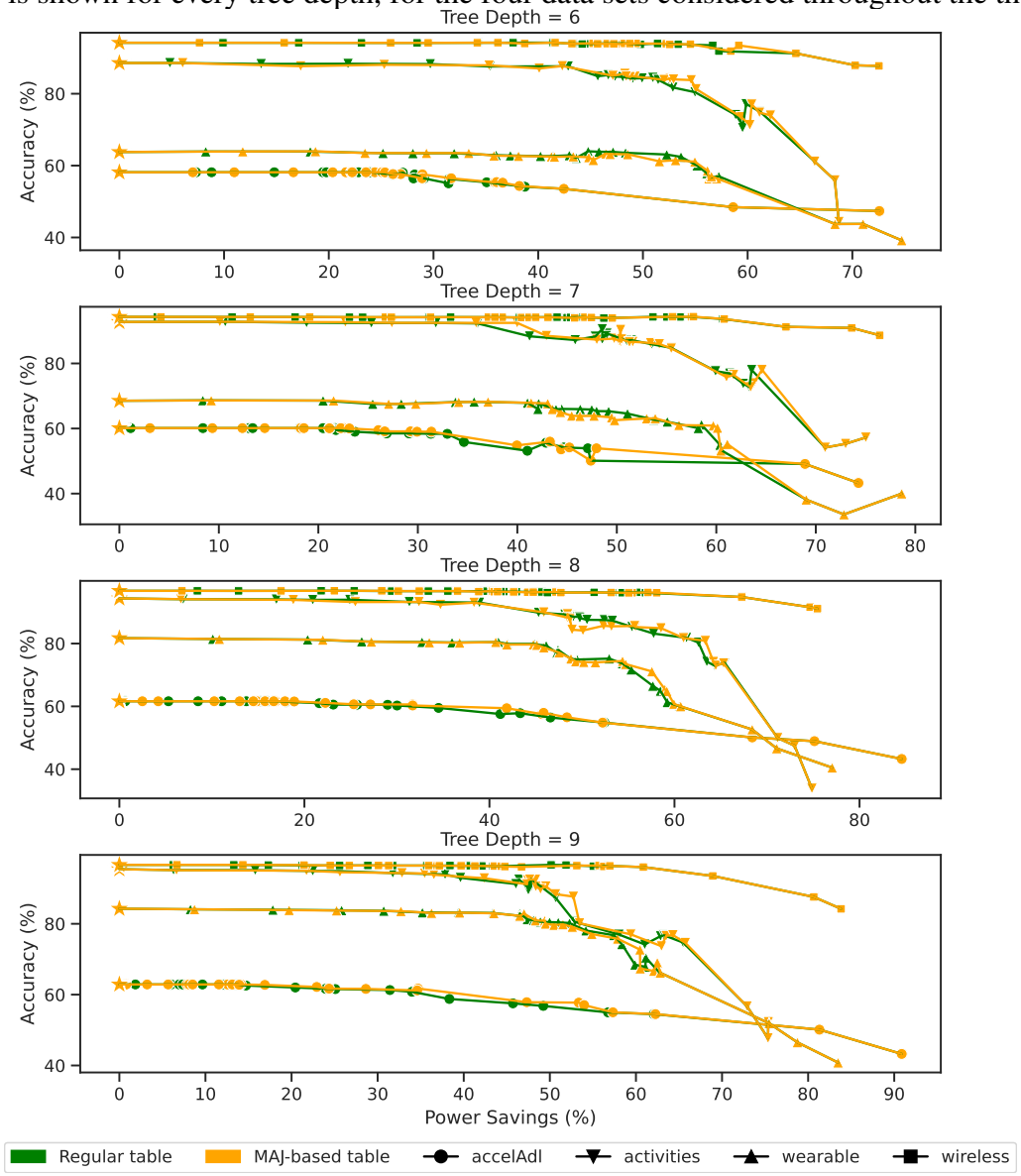
6.6.2 Gate-Level Pruning on MAJ-based DTs/RFs

The gate-level pruning exploration for MAJ-based circuits was analyzed similarly to the results from Section 6.5.1. However, the pruning percentages used in this analysis ranged from 0% to 100% in steps of 0.5%, instead of 1%, to show that the granularity of the technique can be enhanced (limited by steps of a single gate).

The results are presented in Figure 6.17, for the four data sets, considering tree depths from 6 to 9. The baseline points with 0% of pruning are represented as a $\star$. Just as in the MAJ-based C2PAx analyses, the results are presented only for the power savings, given that the curves of the area savings plots are very similar.

As shown in Figure 6.17, every data set presented expected curves, in the sense that no abrupt decreases in accuracy are observed in points close to the baseline. Significant power savings can be observed for every configuration, given that almost every curve is approximately a straight line parallel to the x-axis (aside from wearable, which decays slightly more with smaller pruning percentages w.r.t. the other data sets).

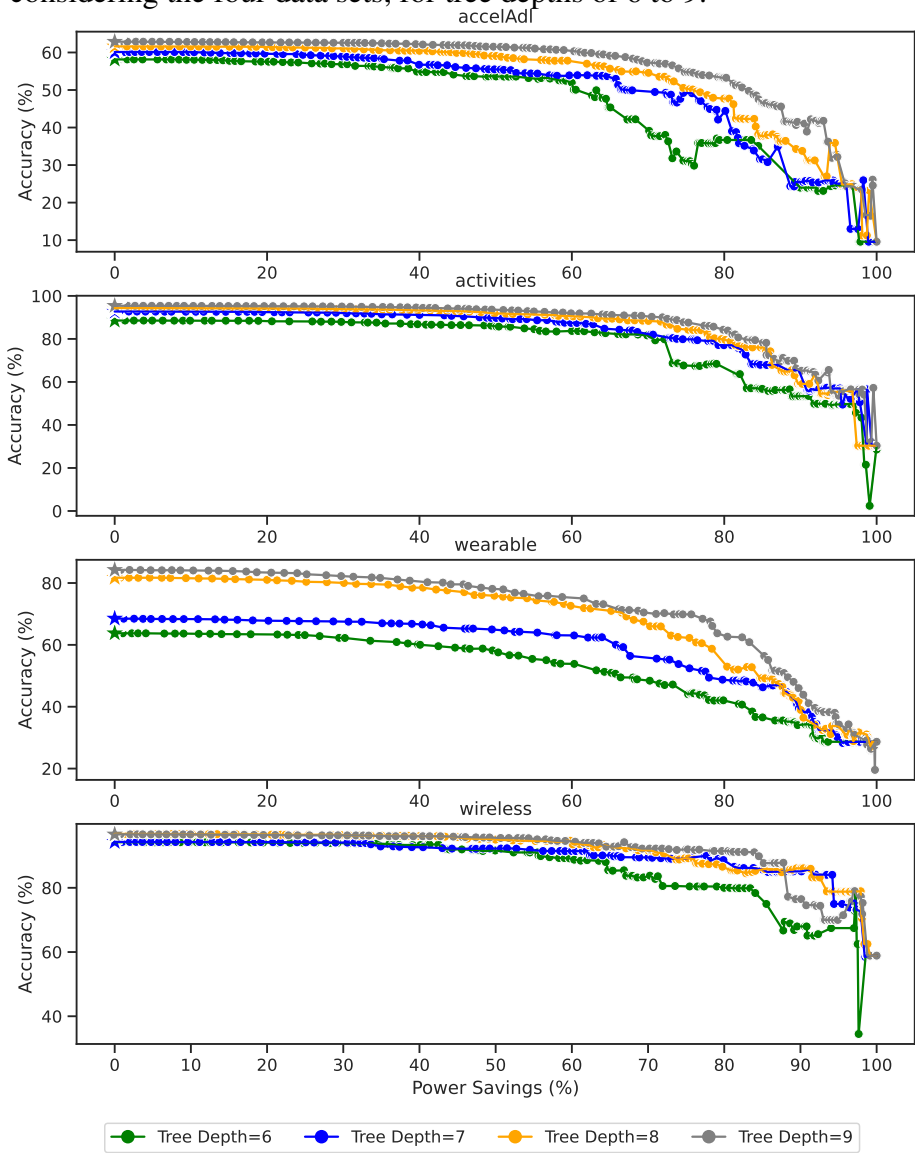
Figure 6.16: Comparison between the C2PAx technique being applied in MAJ-based RFs with regular and a MAJ-based table, in terms of power-accuracy trade-off. The comparison is shown for every tree depth, for the four data sets considered throughout the thesis.



6.7 Chapter Summary

This chapter presented the main results and findings of the work developed so far, for the frameworks regarding tree-based VLSI accelerators, comparator approximation, and both indiscriminate and hierarchical gate-level pruning. Several analyses were performed to verify the resource decrease when applying approximate computing in terms of power-, energy-, and area-accuracy trade-offs. Limitations on resources were obtained through quantization, constant approximation, gate-level pruning, and limitations in tree depth and number of trees. Additionally, some analyses on critical path delay were also presented in this chapter, to verify how these optimizations impact the maximum operat-

Figure 6.17: Power-Accuracy trade-off results of applying pruning to MAJ-based RF models, considering the four data sets, for tree depths of 6 to 9.



ing frequency, when required. Lastly, results were presented regarding some approximations in MAJ-based DTs/RFs: adaptations in C2PAx and pruning for MAJ-based circuits.

7 CONCLUSION

The work presented in this thesis led to important findings and contributions to the field of power-efficient hardware architectures for machine learning. This chapter concludes the report of the thesis, discussing the main contributions of each portion of the work developed.

The claim of this thesis is related to showing that the use of frameworks for automatic generation of tree-based VLSI accelerators can ease the design space exploration when searching for efficient solutions in terms of power efficiency. This thesis was split into three main contributions developed: (i) the framework for automatically generating tree-based VLSI accelerators; (ii) the approximation in the comparator operators; and (iii) the use of indiscriminate and hierarchical gate-level pruning in tree-based models. Additionally, some of the contributions are also extended for MAJ-based circuits, to verify whether they apply in the context of emerging technologies. Other alternative works related to tree-based models were also developed during the doctorate period, not necessarily related to power dissipation. These works include an analysis on IP security, and the use of tree-based models for logic optimization, but these are not presented here given that they are out of the scope of the thesis claims. Even so, all of the works developed here (both the contributions aligned with the scope of this thesis and the alternative ones) make the use of frameworks to ease the analysis, which shows the usefulness of automating the design space exploration.

Section 5.1 presented the baseline framework for generating the VLSI accelerators for DT and RF models. This framework allows for extensive design space explorations to define the best possible configurations, allowing for the variation of tree depth, number of trees, and quantization level. With that, several trade-off analyses were performed to assess the advantages of the variation in each parameter w.r.t. power, energy, area, and accuracy. This framework is made available for designers who seek to employ DT/RF accelerators, and is an important contribution to the scientific community to perform analyses, comparisons and include new proposals within it.

A new method to approximate the comparators of tree-based models was proposed in Section 5.2. This method is specific to comparators that employ explicit constants in one of its operators. In order to verify the impacts of this proposal, it was implemented and evaluated as an extension of the baseline framework, and compared to a more naïve approximation method.

Section 5.3 presented an implementation that merges DTs/RFs and indiscriminate gate-level pruning. This proposal unifies the first framework developed along with an existing tool denoted as AxLS, which approximates the netlist representation of the VLSI descriptions generated by the framework. By automating the generations and pruning configurations, designers can test several different levels of approximation and decide on the best ones. Significant reductions in power, energy, and area were reported, with very small or negligible losses in accuracy. Furthermore, this thesis also proposed the use of hierarchical pruning, as an enhancement of tools such as AxLS. This allows for designers to go even further beyond the limits of indiscriminate pruning, by allowing them to choose the modules to prune.

Section 5.4 presents contributions regarding MAJ-based DTs/RFs, given that MAJ and MIN gates are very convenient to be implemented in emerging technologies. Two main analyses were presented: an adaptation of the C2PAx technique for MAJ-based DTs/RFs, and an analysis of gate-level pruning in these MAJ-based accelerators.

Overall, the current findings, made possible by the use of the frameworks, are related to the way that tree-based VLSI accelerators behave under different types of approximations (some of which are also proposed in this thesis), for several configurations. This is of utmost importance given that tree-based models can be conveniently employed when there are strict battery limitations that prohibit the use of more complex models.

Several future works are possible, also due to the fact that the frameworks containing the VLSI translator, the C2PAx implementation, and AxLS, have all been made available as open-source projects. Among the possibilities, one can opt to implement a DT/RF generator with generic kernels, that have a fixed number of comparators and can work for different data sets using the same VLSI implementation (using generic comparators in this case). Other tree-based models aside from DTs and RFs can also be implemented, such as the ones from the XGBoost framework, which has received significant attention lately. Alternatively, one can opt to maintain the same dedicated implementations of the trees as presented in this thesis, and apply approximate comparators, or modify the adjustment of the constants performed by C2PAx based on some other heuristic (e.g., the step value does not need to be a constant for the entire DT/RF). Other proposals for gate-level pruning using heuristics other than just the standard *probprun* technique from AxLS can be possible, such as prioritizing pruning when the gates are in the critical paths, or pruning based on the gate fanout. All the available frameworks can be conveniently extended to include other similar proposals.

REFERENCES

ABREU, B.; GRELLERT, M.; BAMPI, S. VLSI Design of Tree-Based Inference for Low-Power Learning Applications. In: **2020 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2020. p. 1–5.

ABREU, B.; GRELLERT, M.; BAMPI, S. A framework for designing power-efficient inference accelerators in tree-based learning applications. **Engineering Applications of Artificial Intelligence**, v. 109, p. 104638, 2022. ISSN 0952-1976. Available from Internet: <https://www.sciencedirect.com/science/article/pii/S0952197621004462>.

ABREU, B. et al. On the Netlisit Gate-level Pruning for Tree-based Machine Learning Accelerators. In: **IEEE Latin American Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2022.

ABREU, B. A. de et al. Compact CMOS-Compatible Majority Gate Using Body Biasing in FDSOI Technology. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, v. 13, n. 1, p. 86–95, 2023.

ABREU, B. A. de et al. C2PAx: Complexity-Aware Constant Parameter Approximation for Energy-Efficient Tree-Based Machine Learning Accelerators. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 69, n. 7, p. 2683–2693, 2022.

AGGARWAL, C. C. **Data Mining: The Textbook**. Cham: Springer, 2015. ISBN 978-3-319-14141-1.

AJIRLOU, A. F.; PARTIN-VAISBAND, I. A Machine Learning Pipeline Stage for Adaptive Frequency Adjustment. **IEEE Transactions on Computers**, v. 71, n. 3, p. 587–598, 2022.

AL-GUNAID, M. A. et al. Forecasting energy consumption with the data reliability estimatimation in the management of hybrid energy system using fuzzy decision trees. In: **2016 7th International Conference on Information, Intelligence, Systems Applications (IISA)**. [S.l.: s.n.], 2016. p. 1–8.

AMARU, L.; GAILLARDON, P.-E.; MICHELI, G. Majority-Inverter Graph: A New Paradigm for Logic Optimization. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 35, p. 1–1, 01 2015.

ASSOCIATION, G. **The Mobile Economy 2022**. 2022. Available from Internet: <https://www.gsma.com/mobileeconomy>.

BARTELS, J. et al. A 216 W, 87Tree on FPGA with Interpolated Arctan2. In: **2021 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2021. p. 1–5.

BAUER, D.; WUTZKE, R.; BAUERNHANSL, T. Wear@Work – A New Approach for Data Acquisition Using Wearables. **Procedia CIRP**, v. 50, p. 529–534, 12 2016.

BAUMGARTNER, S.; HUEMER, M.; LUNGLMAYR, M. Efficient Majority Voting in Digital Hardware. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 69, n. 4, p. 2266–2270, 2022.

BEUKENHORST, A. et al. Consumer Smartwatches for Collecting Self-Report and Sensor Data: App Design and Engagement. **Studies in health technology and informatics**, v. 247, p. 291–295, 01 2018.

BOCKERMANN, C. et al. Online Analysis of High-Volume Data Streams in Astroparticle Physics. In: BIFET, A. et al. (Ed.). **Machine Learning and Knowledge Discovery in Databases**. Cham: Springer International Publishing, 2015. p. 100–115. ISBN 978-3-319-23461-8.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

BREIMAN, L. et al. **Classification and Regression Trees**. Taylor & Francis, 1984. ISBN 9780412048418. Available from Internet: <https://books.google.com.br/books?id=JwQx-WOmSyQC>.

BRUNO, B. et al. Human motion modelling and recognition: A computational approach. In: **2012 IEEE International Conference on Automation Science and Engineering (CASE)**. [S.l.: s.n.], 2012. p. 156–161.

BRUNO, B. et al. Analysis of human behavior recognition algorithms based on acceleration data. In: **2013 IEEE International Conference on Robotics and Automation**. [S.l.: s.n.], 2013. p. 1602–1607.

BUSCHJÄGER, S.; MORIK, K. Decision Tree and Random Forest Implementations for Fast Filtering of Sensor Data. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 65, n. 1, p. 209–222, 2018.

CADENCE. **Cadence EDA tools**. 2019. Available from Internet: <https://www.cadence.com/>.

CAI, F. et al. Spectrum sharing for LTE and WiFi coexistence using decision tree and game theory. In: **2016 IEEE Wireless Communications and Networking Conference**. [S.l.: s.n.], 2016. p. 1–6.

CASTRO-GODíNEZ, J. et al. AxLS: A Framework for Approximate Logic Synthesis Based on Netlist Transformations. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 68, n. 8, p. 2845–2849, 2021.

CHANG, S. et al. An Ultra-Low-Power Dual-Mode Automatic Sleep Staging Processor Using Neural-Network-Based Decision Tree. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 66, n. 9, p. 3504–3516, 2019.

CHEN, T.; GUESTRIN, C. XGBoost: A Scalable Tree Boosting System. In: **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. New York, NY, USA: ACM, 2016. (KDD '16), p. 785–794. ISBN 978-1-4503-4232-2. Available from Internet: <http://doi.acm.org/10.1145/2939672.2939785>.

CHEN, Y. et al. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. **IEEE Journal of Solid-State Circuits**, v. 52, n. 1, p. 127–138, Jan 2017.

CHIPPA, V. K. et al. Analysis and characterization of inherent application resilience for approximate computing. In: **2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2013. p. 1–9.

CHOUDHURY, R.; AHAMED, S. R.; GUHA, P. FPGA Implementation of Batch-Mode Depth-Pipelined Two Means Decision Tree. **IEEE Embedded Systems Letters**, v. 15, n. 1, p. 17–20, 2023.

DAGHERO, F. et al. Adaptive Random Forests for Energy-Efficient Inference on Microcontrollers. In: **2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2021. p. 1–6.

DENNARD, R. Past Progress and Future Challenges in VLSI Technology: From DRAM and Scaling to Ultra-Low-Power CMOS. **IEEE Solid-State Circuits Magazine**, v. 7, p. 29–38, 03 2015.

DUA, D.; GRAFF, C. **UCI Machine Learning Repository**. 2017. Available from Internet: <http://archive.ics.uci.edu/ml>.

FRUMUSANU, A. The iPhone XS & XS Max Review: Unveiling the Silicon Secrets. 2018.

GARG, B.; PATEL, S.; DUTT, S. LoBA: A Leading One Bit Based Imprecise Multiplier for Efficient Image Processing. **Journal of Electronic Testing**, v. 36, jun. 2020.

GAURA, E.; KEMP, J.; BRUSEY, J. Leveraging Knowledge From Physiological Data: On-Body Heat Stress Risk Prediction With Sensor Networks. **IEEE Transactions on Biomedical Circuits and Systems**, v. 7, n. 6, p. 861–870, 2013.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GRELLERT, M. et al. Fast coding unit partition decision for hevc using support vector machines. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 29, n. 6, p. 1741–1753, 2019.

HAN, J.; ORSHANSKY, M. Approximate computing: An emerging paradigm for energy-efficient design. In: **2013 18th IEEE European Test Symposium (ETS)**. [S.l.: s.n.], 2013. p. 1–6.

IEEE. IEEE Standard for Floating-Point Arithmetic. **IEEE Std 754-2008**, p. 1–70, 2008.

IGNATOV, D.; IGNATOV, A. Decision Stream: Cultivating Deep Decision Trees. In: **2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)**. [S.l.: s.n.], 2017. p. 905–912.

KANANI, A.; VAIDYA, S.; AGARWAL, H. LightFPGA: Scalable and Automated FPGA Acceleration of LightGBM for Machine Learning Applications. In: **2021 25th International Symposium on VLSI Design and Test (VDAT)**. [S.l.: s.n.], 2021. p. 1–6.

KANG, M. et al. A 19.4-nJ/Decision, 364-K Decisions/s, In-Memory Random Forest Multi-Class Inference Accelerator. **IEEE Journal of Solid-State Circuits**, v. 53, n. 7, p. 2126–2135, 2018.

KARN, R. R.; ELFADEL, I. A. M. Confidential Inference in Decision Trees: FPGA Design and Implementation. In: **2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2022. p. 1–6.

KHDR, H. et al. Power Density-Aware Resource Management for Heterogeneous Tiled Multicores. **IEEE Transactions on Computers**, v. 66, n. 3, p. 488–501, 2017.

KIM, M. S. et al. Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks. v. 68, n. 5, p. 660–675, 2019.

KIM, S. et al. Wearable Multi-Biosignal Analysis Integrated Interface With Direct Sleep-Stage Classification. **IEEE Access**, v. 8, p. 46131–46140, 2020.

KITAMURA, Y. et al. Storage-Efficient Tree Structure with Level-Ordered Unary Degree Sequence for Packet Classification. In: **2015 Third International Symposium on Computing and Networking (CANDAR)**. [S.l.: s.n.], 2015. p. 487–490.

KRAMER, S. **Boolean.py Library**. 2009. Available from Internet: <https://booleanpy.readthedocs.io/en/latest/>.

KUEHN, J.; MANOLI, Y. An Application-Specific Field-Programmable Tree Ensemble Architecture. In: **2018 28th International Conference on Field Programmable Logic and Applications (FPL)**. [S.l.: s.n.], 2018. p. 445–4451.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, May 2015. ISSN 1476-4687. Available from Internet: <https://doi.org/10.1038/nature14539>.

LINGAMNENI, A. et al. Energy parsimonious circuit design through probabilistic pruning. In: **2011 Design, Automation Test in Europe**. [S.l.: s.n.], 2011. p. 1–6.

MIAO, F. et al. A Wearable Sensor for Arterial Stiffness Monitoring Based on Machine Learning Algorithms. **IEEE Sensors Journal**, v. 19, n. 4, p. 1426–1434, 2019.

MISHRA, V. K. et al. A Heuristic-Driven and Cost Effective Majority/ Minority Logic Synthesis for Post-CMOS Emerging Technology. **IEEE Access**, v. 9, p. 168689–168702, 2021.

MITCHELL, T. M. **Machine Learning**. [S.l.]: McGraw-Hill Education, 1997. 432 p. ISSN 18684394. ISBN 0070428077.

MITSUNARI, K. et al. Hardware Architecture for High-Speed Object Detection Using Decision Tree Ensemble. **IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences**, E101.A, p. 1298–1307, 09 2018.

MORTAZAVI, B. et al. Can Smartwatches Replace Smartphones for Posture Tracking? **Sensors (Basel, Switzerland)**, v. 15, p. 26783–26800, 10 2015.

MORTAZAVI, B. J. et al. Determining the Single Best Axis for Exercise Repetition Recognition and Counting on SmartWatches. In: **2014 11th International Conference on Wearable and Implantable Body Sensor Networks**. [S.l.: s.n.], 2014. p. 33–38.

PAGLIARI, D. J.; PONCINO, M.; MACII, E. Energy-Efficient Digital Processing via Approximate Computing. In: ____. **Smart Systems Integration and Simulation**. Cham: Springer International Publishing, 2016. p. 55–89. ISBN 978-3-319-27392-1. Available from Internet: <https://doi.org/10.1007/978-3-319-27392-1_4>.

PAIM, G. et al. A Cross-Layer Gate-Level-to-Application Co-Simulation for Design Space Exploration of Approximate Circuits in HEVC Video Encoders. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 30, n. 10, p. 3814–3828, 2020.

PAIM, G. et al. Power-, Area-, and Compression-Efficient Eight-Point Approximate 2-D Discrete Tchebichef Transform Hardware Design Combining Truncation Pruning and Efficient Transposition Buffers. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 66, n. 2, p. 680–693, 2019.

PASHAEIFAR, M. et al. A Theoretical Framework for Quality Estimation and Optimization of DSP Applications Using Low-Power Approximate Adders. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 66, n. 1, p. 327–340, 2019.

PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

POGGIO, T. A. **Center for Biological & Computational Learning (CBCL) at MIT**. 2007. Available from Internet: <http://poggio-lab.mit.edu/codedatasets>.

PRECEDENCE. **Wearable Technology Market – Global Industry Analysis, Size, Share, Growth, Trends, Regional Outlook, and Forecast 2022-2030**. 2022. Https://www.precedenceresearch.com/wearable-technology-market.

PRISACARIU, V. A. et al. Integrating Object Detection with 3D Tracking Towards a Better Driver Assistance System. In: **2010 20th International Conference on Pattern Recognition**. [S.l.: s.n.], 2010. p. 3344–3347.

RABAEY, J.; PEDRAM, M. **Low Power Design Methodologies**. Springer US, 1996. (The Springer International Series in Engineering and Computer Science). ISBN 9780792396307. Available from Internet: <https://books.google.com.br/books?id=qkpTAAAAMAAJ>.

RABAEY, J. M. **Digital Integrated Circuits: A Design Perspective**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996. ISBN 0-13-178609-1.

RAHMANI, A. M. et al. **The Dark Side of Silicon: Energy Efficient Computing in the Dark Silicon Era**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2017. ISBN 3319315943.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. USA: Prentice Hall Press, 2009. ISBN 0136042597.

SALAMAT, S. et al. Systematic approximate logic optimization using Don't Care conditions. In: **2017 18th International Symposium on Quality Electronic Design (ISQED)**. [S.l.: s.n.], 2017. p. 419–425.

SCHLACHTER, J.; CAMUS, V.; ENZ, C. Near/Sub-Threshold Circuits and Approximate Computing: The Perfect Combination for Ultra-Low-Power Systems. In: **2015 IEEE Computer Society Annual Symposium on VLSI**. [S.l.: s.n.], 2015. p. 476–480.

SCHMITZ, M. T.; AL-HASHIMI, B. M.; ELES, P. **System-Level Design Techniques for Energy-Efficient Embedded Systems**. Norwell, MA, USA: Kluwer Academic Publishers, 2004. ISBN 1402077505.

SEIDEL, H. B. et al. Approximate Pruned and Truncated Haar Discrete Wavelet Transform VLSI Hardware for Energy-Efficient ECG Signal Processing. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 68, n. 5, p. 1814–1826, 2021.

SHAFIQUE, M.; HENKEL, J. Mitigating the power density and temperature problems in the nano-era. In: **2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2015. p. 176–177.

SHARMA, K.; SHEWANDAGN, B.; BHUKYA, S. VLSI implementation of flexible architecture for decision tree classification in data mining. In: **AIP Conference Proceedings**. [S.l.: s.n.], 2017. v. 1859, p. 020092.

SHEN, C. et al. A Best-First Soft/Hard Decision Tree Searching MIMO Decoder for a 4×4 64-QAM System. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 20, n. 8, p. 1537–1541, 2012.

SHIH, X.-Y.; CHIU, Y.; WU, H.-E. Design and Implementation of Decision-Tree (DT) Online Training Hardware Using Divider-Free GI Calculation and Speeding-Up Double-Root Classifier. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 70, n. 2, p. 759–771, 2023.

SILVA, P. A.; GRELLERT, M.; MEINHARDT, C. Exploring Approximate Comparator Circuits on Power Efficient Design of Decision Trees. In: **2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2022. p. 1–6.

SILVEIRA, B. et al. Power-Efficient Sum of Absolute Differences Hardware Architecture Using Adder Compressors for Integer Motion Estimation Design. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 64, n. 12, p. 3126–3137, Dec 2017. ISSN 1549-8328.

SILVER, D. et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. **Science**, v. 362, n. 6419, p. 1140–1144, 2018.

SLIMANI, C. et al. Accelerating Random Forest on Memory-Constrained Devices Through Data Storage Optimization. **IEEE Transactions on Computers**, v. 72, n. 6, p. 1595–1609, 2023.

SOARES, L. B. et al. Design Methodology to Explore Hybrid Approximate Adders for Energy-Efficient Image and Video Processing Accelerators. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 66, n. 6, p. 2137–2150, 2019.

SOEKEN, M.; AL. et. **The EPFL Logic Synthesis Libraries**. arXiv, 2018. Available from Internet: <https://arxiv.org/abs/1805.05121>.

SONG, W. et al. Design of a Flexible Wearable Smart sEMG Recorder Integrated Gradient Boosting Decision Tree Based Hand Gesture Recognition. **IEEE Transactions on Biomedical Circuits and Systems**, v. 13, n. 6, p. 1563–1574, 2019.

STANLEY-MARBELL, P. et al. Exploiting Errors for Efficiency: A Survey from Circuits to Applications. **ACM Computing Surveys**, Association for Computing Machinery, New York, NY, USA, v. 53, n. 3, jun. 2020. ISSN 0360-0300.

STMICROELECTRONICS. **ST 65nm Standard Cell Library**. 2021. Available from Internet: <www.st.com>.

STREET, W. N.; WOLBERG, W. H.; MANGASARIAN, O. L. Nuclear feature extraction for breast tumor diagnosis. In: ACHARYA, R. S.; GOLDGOF, D. B. (Ed.). **Biomedical Image Processing and Biomedical Visualization**. SPIE, 1993. v. 1905, p. 861 – 870. Available from Internet: <https://doi.org/10.1117/12.148698>.

SYNTHESIS, B. L.; GROUP, V. **ABC: A System for Sequential Synthesis and Verification**. 2019. Http://www.eecs.berkeley.edu/ alanmi/abc/.html.

SZE, V. et al. Hardware for machine learning: Challenges and opportunities. In: **2017 IEEE Custom Integrated Circuits Conference (CICC)**. [S.l.: s.n.], 2017. p. 1–8.

TASOULAS, Z.-G. et al. Weight-Oriented Approximation for Energy-Efficient Neural Network Inference Accelerators. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 67, n. 12, p. 4670–4683, 2020.

TING, K. M. Precision and Recall. In: ____. **Encyclopedia of Machine Learning**. Boston, MA: Springer US, 2010. p. 781–781. ISBN 978-0-387-30164-8. Available from Internet: <https://doi.org/10.1007/978-0-387-30164-8_652>.

TONG, D.; QU, Y. R.; PRASANNA, V. K. Accelerating Decision Tree Based Traffic Classification on FPGA and Multicore Platforms. **IEEE Transactions on Parallel and Distributed Systems**, v. 28, n. 11, p. 3046–3059, Nov 2017.

TORRES, R. L. S. et al. Sensor enabled wearable RFID technology for mitigating the risk of falls near beds. In: **2013 IEEE International Conference on RFID (RFID)**. [S.l.: s.n.], 2013. p. 191–198.

TURING, A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE. **Mind**, LIX, n. 236, p. 433–460, 10 1950. ISSN 0026-4423. Available from Internet: <https://doi.org/10.1093/mind/LIX.236.433>.

TUSOR, B. et al. A fast fuzzy decision tree for color filtering. In: **2015 IEEE 9th International Symposium on Intelligent Signal Processing (WISP) Proceedings**. [S.l.: s.n.], 2015. p. 1–6.

UGULINO, W. et al. Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements. In: BARROS, L. N. et al. (Ed.). **Advances in Artificial Intelligence - SBIA 2012**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 52–61. ISBN 978-3-642-34459-6.

VELLOSO, E. et al. Qualitative Activity Recognition of Weight Lifting Exercises. In: **Proceedings of the 4th Augmented Human International Conference**. New York, NY, USA: ACM, 2013. (AH '13), p. 116–123. ISBN 978-1-4503-1904-1. Available from Internet: <http://doi.acm.org/10.1145/2459236.2459256>.

VENKATARAMANI, S. et al. Scalable-effort Classifiers for Energy-efficient Machine Learning. In: **Proceedings of the 52Nd Annual Design Automation Conference**. New York, NY, USA: ACM, 2015. (DAC '15), p. 67:1–67:6. ISBN 978-1-4503-3520-1. Available from Internet: <http://doi.acm.org/10.1145/2744769.2744904>.

WANG, C. et al. Selecting Power-Efficient Signal Features for a Low-Power Fall Detector. **IEEE Transactions on Biomedical Engineering**, v. 64, n. 11, p. 2729–2736, 2017.

WANG, J. et al. Non-Stationary Representation for Continuity Aware Head Pose Estimation Via Deep Neural Decision Trees. **IEEE Access**, v. 7, p. 181947–181958, 2019.

XU, X. et al. Quantization of fully convolutional networks for accurate biomedical image segmentation. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 8300–8308.

YANG, Y.; BOLING, C. S.; MASON, A. J. Power-area efficient VLSI implementation of decision tree based spike classification for neural recording implants. In: **2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings**. [S.l.: s.n.], 2014. p. 380–383.

ZENDEGANI, R. et al. RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing. v. 25, n. 2, p. 393–401, 2017.

ZERVAKIS, G. et al. Approximate Computing for ML: State-of-the-art, Challenges and Visions. In: **2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2021. p. 189–196.

ZHANG, Q. et al. ApproxANN: An approximate computing framework for artificial neural network. In: **2015 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2015. p. 701–706.

ZHAO, S.; SUN, Y.; CHEN, S. A Discretization Method for Floating-Point Number in FPGA-based Decision Tree Accelerator. In: **2018 IEEE 4th International Conference on Computer and Communications (ICCC)**. [S.l.: s.n.], 2018. p. 2698–2703.

ZHENG, R.; HU, Q.; JIN, H. GPUPerfML: A Performance Analytical Model Based on Decision Tree for GPU Architectures. In: **2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)**. [S.l.: s.n.], 2018. p. 602–609.

ZHOU, A. et al. Incremental network quantization: Towards lossless cnns with low-precision weights. **arXiv preprint arXiv:1702.03044**, 2017.

APPENDIX A — PAPERS PUBLISHED DURING PH.D. PROGRAM

PAPERS PUBLISHED IN JOURNALS

1. **ABREU, Brunno**; MEMA, Albi; THOMANN, Simon; PAIM, Guilherme; FLORES, Paulo; BAMPI, Sergio; AMROUCH, Hussam. *Compact CMOS-Compatible Majority Gate Using Body Biasing in FDSOI Technology*. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2023.
2. ARBELETCHE, Yuri; PAIM, Guilherme; **ABREU, Brunno**; ALMEIDA, Sérgio; COSTA, Eduardo; FLORES, Paulo; BAMPI, Sergio. *MAxPy: A Framework for Bridging Approximate Computing Circuits to its Applications*. IEEE Transactions on Circuits and Systems-II, 2023.
3. **ABREU, Brunno**; PAIM, Guilherme; GRELLERT, Mateus; BAMPI, Sergio. *C2PAx: Complexity-aware Constant Parameter Approximation for Energy-Efficient Tree-Based Machine Learning Accelerators*. IEEE Transactions of Circuits and Systems-I, 2022.
4. **ABREU, Brunno**; GRELLERT, Mateus; BAMPI, Sergio. *A framework for designing power-efficient inference accelerators in tree-based learning applications*. Elsevier Engineering Applications of Artificial Intelligence, 2022.
5. PEREIRA, Pedro; COSTA, Patrícia; FERREIRA, Guilherme; **ABREU, Brunno**; PAIM, Guilherme; COSTA, Eduardo; BAMPI, Sergio. *Energy-quality scalable design space exploration of approximate FFT hardware architectures*. IEEE Transactions on Circuits and Systems-I, 2022.
6. BERNDT, Augusto; **ABREU, Brunno**; CAMPOS, Isac S.; LIMA, Bryan; GRELLERT, Mateus; CARVALHO, Jonata T.; MEINHARDT, Cristina. *A CGP-based Logic Flow: Optimizing Accuracy and Size*. Journal of Integrated Circuits and Systems, 2021.
7. PAIM, Guilherme; AMROUCH, Hussam; ROCHA, Leandro M. G.; **ABREU, Brunno**; BAMPI, Sergio; HENKEL, Jorg. *A Framework for Crossing Temperature-Induced Timing Errors Underlying Hardware Accelerators to the Algorithm and Application Layers*. IEEE Transactions on Computers, 2021.
8. SILVEIRA, Bianca; PAIM, Guilherme; ABREU, Brunno; FERREIRA, Rafael; DINIZ, Claudio; COSTA, Eduardo A.; BAMPI, Sergio. *The 4-2 Fused Adder-*

Subtractor Compressor for Low-Power Butterfly-Based Hardware Architectures. Circuits, Systems, and Signal Processing, 2021.

9. **ABREU, Brunno**; GRELLERT, Mateus; PAIM, Guilherme; ROCHA, Leandro M. G.; DINIZ, Claudio M.; COSTA, Eduardo A.; BAMPI, Sergio. *Exploring Absolute Differences Arithmetic Operators for Power- and Area-Efficient SAD Hardware Architectures*. Journal of Integrated Circuits and Systems, v. 15, p. 1, 2020.
10. PAIM, Guilherme; SANTANA, Gustavo M.; **ABREU, Brunno**; ROCHA, Leandro M. G.; GRELLERT, Mateus; COSTA, Eduardo A.; BAMPI, Sergio. *Exploring High-Order Adder Compressors for Reducing Power in Sum of Absolute Differences Architecture for Real-time UHD Video Encoding*. Journal of Real-Time Image Processing, v. 1, p. 1-1, 2020.

PAPERS PUBLISHED IN CONFERENCE PROCEEDINGS

1. CAMPOS, Isac; BERNDT, Augusto; **ABREU, Brunno**; CARVALHO, Jonata; GRELLERT, Mateus; MEINHARDT, Cristina. *A Decision Tree Synthesis Flow for Precise and Approximate Circuits*. In: IEEE Dallas Circuit and System Conference (DCAS), 2022.
2. WUERDIG, Rodrigo N.; SARTORI, Marcos L. L.; **ABREU, Brunno**; BAMPI, Sergio; CALAZANS, Ney. L. V.. *Mitigating Asynchronous QDI Drawbacks on MAC Operators with Approximate Multipliers*. In: IEEE International Symposium on Circuits and Systems (ISCAS), 2022.
3. COSTA, Patricia; PEREIRA, Pedro T. L.; **ABREU, Brunno**; PAIM, Guilherme; COSTA, Eduardo; BAMPI, Sergio. *Improved Approximate Multipliers for Single-Precision Floating-Point Hardware Design*. In: IEEE Latin American Symposium on Circuits and Systems (LASCAS), 2022.
4. **ABREU, Brunno**; PAIM, Guilherme; CASTRO-GODINEZ, Jorge; GRELLERT, Mateus; BAMPI, Sergio. *On the Netlist Gate-level Pruning for Tree-based Machine Learning Accelerators*. In: IEEE Latin American Symposium on Circuits and Systems (LASCAS), 2022.
5. BERNDT, Augusto; **ABREU, Brunno**; CAMPOS, Isac; LIMA, Bryan; GRELLERT, Mateus; CARVALHO, Jonata T.; MEINHARDT, Cristina. *Accuracy and Size Trade-off of a Cartesian Genetic Programming Flow for Logic Optimization*. In: Sympo-

sium on Integrated Circuits and Systems Design (SBCCI), 2021.

6. RAI, Shubham; NETO, Walter L.; MIYASAKA, Yukio; ZHANG, Xinpei; YU, Mingfei; YI, Qingyang; FUJITA, Masahiro; MANSKE, Guilherme B.; PONTES, Matheus F.; JUNIOR, Leomar S.; AGUIAR, Marilton S.; BUTZEN, Paulo F.; CHIEN, Po-Chun; HUANG, Yu-Shan; WANG, Hoa-Ren; JIANG, Jie-Hong R.; GU, Jiaqi; ZHAO, Zheng; JIANG, Zixuan; PAN, David Z.; **ABREU, Brunno**; CAMPOS, Isac; BERNDT, Augusto; MEINHARDT, Cristina; CARVALHO, Jonata T.; GRELLERT, Mateus; BAMPI, Sergio; LOHANA, Aditya; KUMAR, Akash; ZENG, Wei; DAVOODI, Azadeh; TOPALOGLU, Rasit O.; ZHOU, Yuan; DOTZEL, Jordan; ZHANG, Yichi; WANG, Hanyu; ZHANG, Zhiru; TENACE, Valerio; GAILLARDON, Pierre-Emmanuel; MISCHENKO, Alan; CHATTERJEE, Satrajit. *Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization*. In: Design, Automation and Test in Europe (DATE), 2021.
7. **ABREU, Brunno**; BERNDT, Augusto; CAMPOS, Isac S.; MEINHARDT, Cristina; CARVALHO, Jonata T.; GRELLERT, Mateus; BAMPI, Sergio. *Fast Logic Optimization Using Decision Trees*. In: IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, 2021.
8. DICK, João; **ABREU, Brunno**; GRELLERT, Mateus; BAMPI, Sergio. *Quality and Complexity Assessment of Learning-Based Image Compression Solutions*. In: IEEE International Conference on Image Processing (ICIP), Alaska, 2021.
9. **ABREU, Brunno**; GRELLERT, Mateus; BAMPI, Sergio. *VLSI Design of Tree-Based Inference for Low-Power Learning Applications*. In: IEEE International Symposium on Circuits and Systems (ISCAS), Sevilla, 2020.
10. FERREIRA, Rafael; PAIM, Guilherme; **ABREU, Brunno**; DINIZ, Claudio M.; COSTA, Eduardo; BAMPI, Sergio. *HEVC Interpolation Filter Architecture Using Hybrid Encoding Arithmetic Operators*. In: IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), Texas, 2019.

PAPERS IN ARXIV

1. RAI, Shubham; NETO, Walter L.; MIYASAKA, Yukio; ZHANG, Xinpei; YU, Mingfei; YI, Qingyang; FUJITA, Masahiro; MANSKE, Guilherme B.; PONTES, Matheus F.; JUNIOR, Leomar S.; AGUIAR, Marilton S.; BUTZEN, Paulo F.;

CHIEN, Po-Chun; HUANG, Yu-Shan; WANG, Hoa-Ren; JIANG, Jie-Hong R.; GU, Jiaqi; ZHAO, Zheng; JIANG, Zixuan; PAN, David Z.; **ABREU, Brunno**; CAMPOS, Isac; BERNDT, Augusto; MEINHARDT, Cristina; CARVALHO, Jonata T.; GRELLERT, Mateus; BAMPI, Sergio; LOHANA, Aditya; KUMAR, Akash; ZENG, Wei; DAVOODI, Azadeh; TOPALOGLU, Rasit O.; ZHOU, Yuan; DOTZEL, Jordan; ZHANG, Yichi; WANG, Hanyu; ZHANG, Zhiru; TENACE, Valerio; GAILLARDON, Pierre-Emmanuel; MISCHENKO, Alan; CHATTERJEE, Satrajit. *Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization*. In: Arxiv, 2020.