Towards Hardware-Aware Tractable Learning of Probabilistic Models

Laura I. Galindez Olascoaga†, Wannes Meert‡ , Nimish Shah†
Marian Verhelst† and Guy Van den Broeck††
† Electrical Engineering Department, KU Leuven
‡ Computer Science Department, KU Leuven
†† Computer Science Department, University of California, Los Angeles
{laura.galindez,nimish.shah,marian.verhelst}@esat.kuleuven.be
wannes.meert@cs.kuleuven.be, guyvdb@cs.ucla.edu

Abstract

Smart portable applications increasingly rely on edge computing due to privacy and latency concerns. But guaranteeing always-on functionality comes with two major challenges: heavily resource-constrained hardware; and dynamic application conditions. Probabilistic models present an ideal solution to these challenges: they are robust to missing data, allow for joint predictions and have small data needs. In addition, ongoing efforts in the field of tractable learning have resulted in probabilistic models with strict inference efficiency guarantees. However, the current notions of tractability are often limited to model complexity, disregarding the hardware's specifications and constraints. We propose a novel resource-aware cost metric that takes into consideration the hardware's properties in determining whether the inference task can be efficiently deployed. We use this metric to evaluate the performance versus resource trade-off relevant to the application of interest, and we propose a strategy that selects the device settings that can optimally meet users' requirements. We showcase our framework on a mobile activity recognition scenario, and on a variety of benchmark datasets representative of the field of tractable learning and of the applications of interest.

1 Introduction

Tractable learning aims to balance the trade-off between how well the resulting models fit the available data and how efficiently queries are answered. Most implementations focus on maximizing model performance and only factor in query efficiency by subjecting the learning stage to a fixed tractability constraint (e.g. max treewidth [2]). While recent notions of tractability consider the cost of probabilistic inference as the number of arithmetic operations involved in a query [26, 27], they still disregard hardware implementation nuances of the target application. This is of special concern for edge computing on embedded applications, where the target algorithm must run in resource constrained hardware, such as a small ARM or RISC-V embedded processor, or a microcontroller. For such architectures running a lightweight operating system, the overall compute cost is mostly determined by the cost of fundamental arithmetic operations, the interaction with sensor interfaces and the device's memory transactions [17, 12].

In addition, efforts towards hardware-efficient realizations of probabilistic inference are currently scarce [36, 21, 34]. This is in stark contrast with the tremendous progress achieved by embedded neural network implementations [37, 18, 29] .

We address these limitations of the field of tractable learning by proposing a novel resource-aware cost metric that takes into consideration the target embedded device's properties (e.g. energy consumption);

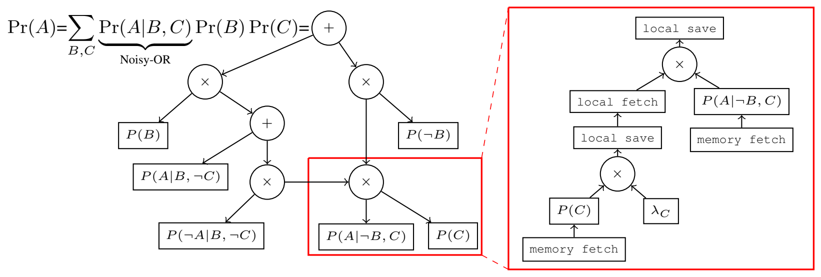


Figure 1: Arithmetic Circuit from a compiled noisy-OR and its mapping to hardware.

and system-level configuration (e.g. sensors used). We map these hardware characteristics to the cost vs. performance trade-off space, and propose a set of techniques to find the optimal system-level configuration. Specifically, we address the following points: (a) Section 3 discusses the relevant hardware-aware tractability metrics, and Section 4 defines the problem statement; (b) Section 5 discusses how to exploit the model's properties to exchange task-performance for hardware efficiency, and introduces techniques to find the optimal set of system configurations in the cost vs. performance trade-off space; and (c) Section 6 shows practical examples of these optimal solutions. This work constitutes one of the first efforts to introduce the field of tractable probabilistic reasoning to the emerging domain of edge computing. This is motivated by probabilistic models' traits, several of which are ideal for portable applications that require reasoning on the edge: robustness to missing information, small data needs, joint predictions, and expert knowledge integration. Moreover, unlike fixed neural architecture training, tractable learning allows to explicitly vary the level of complexity of the inference task, which allows us to model resource tunability.

2 Background and motivation

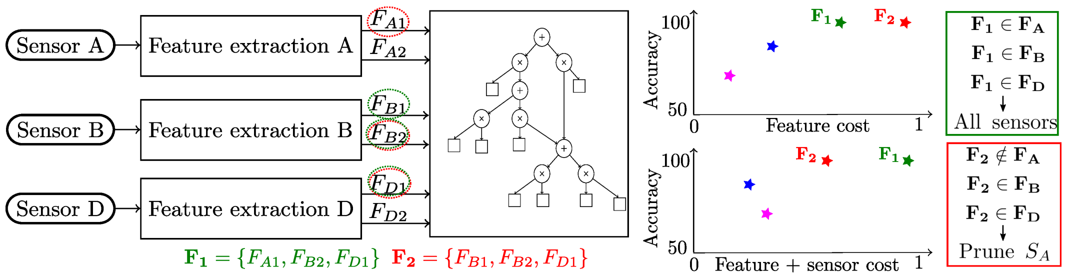
We use standard notation: random variables are denoted by upper case letters X and their instantiations by lower case letters x. Sets of variables are denoted in bold upper case $\mathbf{X}$ and their joint instantiations in bold lower case $\mathbf{x}$. Sets of variable sets are denoted with $\mathcal{X}$.

The model representation of choice in this paper is the Arithmetic Circuit (AC), a state-of-the-art, compact representation for a variety of machine learning models such as probabilistic graphical models (PGMs) [6] and probabilistic programs [10]. Recent developments show how the structure of ACs can also be learned from data [24, 23]. Furthermore, ACs can be complemented with deep learning architectures [41, 28] to achieve the best of both worlds. An alternative representation of ACs are Sum-Product Networks (SPNs), which can also encode probability distributions as a computational graph [32, 13]. SPNs can be efficiently converted to ACs and vice versa [33].

2.1 Probabilistic inference with Arithmetic Circuits

An AC is a directed acyclic graph where inner nodes represent addition or multiplication and leaf nodes are real-valued variables. ACs constitute a standard representation for computing polynomials, but they have proven to be efficient for reasoning over knowledge bases and probabilistic models when a number of additional properties are enforced on them [6]. Once the circuit is known, the complexity of executing the encoded formula is also known, since marginalization and partition function operations are polynomial in the size of the circuit [4], thus making them a well-suited representation for tractable learning. ACs represent a joint probability distribution over a set of random variables $\mathbf{X}$: the leaf nodes are either binary indicator variables $\lambda_{X=x}$, where $X \in \mathbf{X}$, or parameters θ. Figure 1 shows an example of an AC that encodes the joint probability distribution of a noisy-OR model [15].

This representation allows to perform inference to answer a number of probabilistic queries. For example, given an instantiation $\mathbf{f}$ of $\mathbf{F} \subseteq \mathbf{X}$, the marginal probability $\Pr(\mathbf{f})$ can be computed by setting the indicator variables to 1 if they correspond to instantiations consistent with the observed values, $\lambda_{X=x} \leftarrow 1_{x\sim\mathbf{f}}$, and subsequently performing an upward pass on the AC [4]. In a binary classification task, one can define a class variable C, a feature set $\mathbf{F}$ and a classification threshold T, assumed to be 0.5 in this work. For a given instance $\mathbf{f}$, the task consists of selecting the class C_T



Sensor A
Feature extraction A
F_{A1}
F_{A2}
Sensor B
Feature extraction B
F_{B1}
F_{B2}
Sensor D
Feature extraction D
F_{D1}
F_{D2}
$\mathbf{F_1} = \{F_{A1}, F_{B2}, F_{D1}\}$ $\mathbf{F_2} = \{F_{B1}, F_{B2}, F_{D1}\}$
Accuracy
100
50
0
Feature cost
1
$\mathbf{F_1}$
$\mathbf{F_2}$
Feature + sensor cost
$\mathbf{F_1} \in \mathbf{F_A}$
$\mathbf{F_1} \in \mathbf{F_B}$
$\mathbf{F_1} \in \mathbf{F_D}$
All sensors
$\mathbf{F_2} \notin \mathbf{F_A}$
$\mathbf{F_2} \in \mathbf{F_B}$
$\mathbf{F_2} \in \mathbf{F_D}$
Prune S_A

Figure 2: Sensory embedded classification example.

for which the condition $Pr(C|\mathbf{f}) \geq 0.5$ is met. The conditional probability can be calculated by performing two upward passes on the AC[1] that encodes $\Pr(C, \mathbf{F})$, after setting the indicator variables λ in accordance to instance $\mathbf{f}$. ACs' straightforward mapping to embedded hardware and the fact that they readily encode the algorithm necessary for inference, motivates our choice for this probabilistic model representation. Moreover, the process of learning them already entails a trade-off between their predictive performance and their computational efficiency. The following section motivates our proposed hardware-aware tractability metric.

2.2 Motivating example

Consider the mobile classification scenario in Figure 2, where the feature set $\mathbf{F}$=$\{F_{A1}$,F_{A2},F_{B1},F_{B2},F_{D1},$F_{D2}\}$ is extracted from sensors A, B and D, and where the AC is assumed to be the most compact model that maximizes classification accuracy. Suppose there are two feature subsets available, $\mathbf{F_1}$=$\{F_{A1}, F_{B2}, F_{D1}\}$ and $\mathbf{F_2}$=$\{F_{B1}, F_{B2}, F_{D1}\}$; and that they attain the same accuracy. Hence, the goal is to find the least expensive subset. The solution to this problem would clearly be $\mathbf{F_1}$ when considering only feature cost, a common approach to address the problem of feature subset selection. But when considering also the costs of the sensors, $\mathbf{F_2}$ turns out to be a better choice, as sensor A is unused and can be turned off. This example shows that trade-off opportunities can be missed when failing to describe realistic hardware-aware system-level costs.

3 Hardware-aware cost

In this section we formalize the notion of hardware-aware cost, the basis of our optimization framework. Let $\alpha = \langle +, \times, \theta \rangle$ be an AC that encodes a joint probability distribution over variables $\mathbf{F}$, extracted from the set of sensor interfaces $\mathbf{S}$. The *hardware-aware cost* ($\mathrm{C_{HA}}$) is defined as:

$$\mathrm{C_{HA}}(\alpha, \mathbf{S}, \mathbf{F}) = \mathrm{C_{AC}}(\alpha) + \sum_{S \in \mathbf{S}} \mathrm{C_{SI}}(S, \mathbf{F}_S), \tag{1}$$

where $\mathrm{C_{AC}}$ are the computation costs, pertaining to inference on arithmetic circuit α, $\mathrm{C_{SI}}$ are the sensor interfacing and feature extraction costs, and $\mathbf{F}_S$ is the feature subset extracted from sensor S.

Computation costs. At a high level, a typical embedded hardware architecture entails two components: an on-chip main memory (typically SRAM), which commonly houses the algorithm's parameter set; and a processing unit, where operations are performed and intermediate values are cached in a local memory. Performing an upward pass on an AC involves the following actions (see Figure 1): 1) fetching parameters from the main memory, 2) performing arithmetic operations, consisting of additions and multiplications, 3) caching intermediate values in a local memory (e.g. register file or low level cache) and 4) fetching intermediate values from local memory, as needed.[2] Each action has a significantly different hardware resource cost. For example, post synthesis energy models of a simple embedded CPU show that multiplications can require 4 times as much energy as additions, and memory transactions 5 times as much energy as multiplications [17]. When it

[1]Conditional probability can also be performed by an upward and a downward pass [6].

[2]In this work we assume that the local cache size is sufficiently large to store intermediate values, but not large enough to store parameters. However, for some learned circuits, there are about as many parameters as edges, so depending on the local memory size, one might need to store intermediate values also in main memory.

comes to the design of embedded hardware, energy efficiency is indeed one of the main challenges to address. Hence, we continue to focus on this resource as a proof of concept without loss of generality; examples of other relevant hardware resource metrics are throughput and latency. It is evident that the total hardware cost of performing a pass on an AC must factor in all the aforementioned transactions. Let nb be the number of bits used to represents parameters θ and perform arithmetic operations $+$ and $\times$. The *computation cost* ($\mathrm{C_{AC}}$) of AC α is defined as:

$$\mathrm{C_{AC}}(\alpha, nb) = \mathrm{C_{+}}(nb) + \mathrm{C_{\times}}(nb) + \mathrm{C_{mem}}(nb) + \mathrm{C_{cache}}(nb), \tag{2}$$

where the terms in $\mathrm{C_{AC}}$ define the cost incurred by each type of operation. Here, $\mathrm{C_{+}}$ and $\mathrm{C_{\times}}$ are the costs of addition and multiplication; $\mathrm{C_{mem}}$ is the cost from fetching parameter leaf nodes from main memory and $\mathrm{C_{cache}}$ is the cost from storing and fetching from local cache (as in Figure 1):

$$\mathrm{C_{+}}(nb) = \textstyle\sum_a [a =_t +] \cdot \phi_{+}(nb), \qquad \mathrm{C_{mem}}(nb) = \textstyle\sum_a [a \neq_t + \text{ and } a \neq_t \times] \cdot \phi_{\mathrm{mem}}(nb),$$
$$\mathrm{C_{\times}}(nb) = \textstyle\sum_a [a =_t \times] \cdot \phi_{\times}(nb), \qquad \mathrm{C_{cache}}(nb) = \textstyle\sum_a [a =_t + \text{ or } a =_t \times] \cdot \phi_{\mathrm{cache}}(nb),$$

where a denotes a node in α, the equality $=_t$ holds when node a matches the operation type on the right side and $[\beta]$ is equal to 1 when β is true. The function $\phi(.)$ describes the effective cost of the particular operation and can be derived from empirical benchmarks, customized to the target hardware [17, 35]. When expressing cost in terms of energy consumption, computation costs scale with the precision in number of bits used to represent parameters and perform arithmetic operations (nb), which is typically the same for all nodes in the AC. To conclude, the cost incurred by each node in an AC is determined by its type (whether addition, multiplication, local parameter fetch, or remote memory access) and the resolution of the operation or parameter (in nb).

Sensor interfacing costs The computational block described above is often part of a larger system, which repeatedly performs a task based on external inputs or observations, such as classification. In this scenario, one must factor in the costs incurred by interfacing with the environment or the user. A sensory interface consists of a set of sensors $\mathbf{S}$, which gather, process and digitize environmental information (typically in the analog domain), and a (typically digital) feature extraction unit, which generates the feature set $\mathbf{F}$ to be used by the machine learning algorithm. Let $\mathbf{S}$ be the set of available sensors and $\mathbf{F}$ the feature set extracted from them. The sensor interfacing cost ($\mathrm{C_{SI}}$) is:

$$\mathrm{C_{SI}}(S, \mathbf{F}_S) = \mathrm{C_S}(S) + \sum_{F_S \in \mathbf{F}_S} \mathrm{C_F}(F_S), \tag{3}$$

where $\mathrm{C_S}$ describes the cost incurred by sensor S and $\mathrm{C_F}$ the cost of extracting feature set $\mathbf{F}_S \subseteq \mathbf{F}$. The sensing cost function $\mathrm{C_S}$ can be customized to the target platform and applications through measurements or data sheets. Note that, if no features from a given sensor are used, it can be shut down, and its cost dropped (see Figure 1). In most systems, $\mathrm{C_F}$ can be defined from the type and number of arithmetic and memory operations involved, in a similar fashion to the computation cost estimation $\mathrm{C_{AC}}$, as will be illustrated in the experiments (Section 6.1).

4 Problem statement

We have seen so far that $\mathrm{C_{HA}}$ depends on four system properties:1) the complexity of model α, determined by the number and type of its operations; 2) the size and type of the feature set $\mathbf{F}$; 3) the size and type of the available sensor set $\mathbf{S}$; and 4) the number of bits nb used within α. We refer to an instantiation of these four properties $\sigma = \{\alpha, \mathbf{F}, \mathbf{S}, nb\}$ as a *system configuration*. Clearly, the system configuration also determines the algorithm's performance, defined according to the application of interest. The methods proposed in this work can accommodate any performance metric or miss-classification cost, but we will only consider accuracy, due to its generality. Specifically, we set the classification threshold to $T = 0.5$, and we consider the accuracy of the Bayes-optimal predictions (*Acc*) over a set of feature instantiations $\{\mathbf{f}_1, ..., \mathbf{f}_l\}$.

Section 2.2 asks to identify the system configuration that incurs the lowest cost for a desired accuracy. Similarly, we might be interested in the configuration that achieves the highest accuracy for a given cost constraint. Thus, the problem we aim to address is how to select the system configurations that map to the Pareto-frontier on the hardware-cost vs. accuracy trade-off space. The inputs to our problem are the class variable C, the available features $\mathbf{F}$ and sensors $\mathbf{S}$ sets, and the set of available precisions $\mathbf{nb}$. The output is the set of Pareto-optimal system configurations $\boldsymbol{\sigma}^* = \{\{\alpha_i^*, \mathbf{F}^*{}_i, \mathbf{S}^*{}_i, nb_i^*\}_{i=1:p}\}$.

5 Trade-off space search

We propose to search the cost vs. accuracy trade-off space by scaling four properties (see Section 4):

Model complexity scaling. We learn a set of ACs $\boldsymbol{\alpha}$ of increasing complexity. Each maps to a specific classification accuracy and computation cost C_{AC} (see Eq. 2). Although discriminative AC learners have shown state-of-the-art classification accuracy [24], we have opted for a generative learning strategy: the LearnPSDD algorithm [23]. The motivation for this choice is twofold: this algorithm improves the model incrementally, but each iteration already leads to a *valid* AC, that can be used to populate the set $\boldsymbol{\alpha}$. Moreover, the learned ACs encode a joint probability distribution, which ensures they are robust to missing data, as demanded by the application range of interest: embedded reasoning tasks must often deal with missing values, either due to malfunction (e.g., a sensor is blocked in an autonomous driving system), or to enforce hardware-cost efficiency (e.g., when energy consumption is excessive, the driving system has the choice to turn off an expensive sensor and the features extracted from it).

Feature and sensor set scaling. We scale the feature set $\mathbf{F}$ by sequentially pruning individual features (see Section 5.1). The effect of feature pruning on classification accuracy has been discussed in numerous works [11, 5, 22] and the impact on the hardware-aware cost is clear from Eq. 3. Pruning features can also have an impact on the computation costs C_{AC}: if a variable is always unobserved, its indicator variables are fixed (see Section 2.1), which we exploit to simplify the circuit (see Algorithm 2). In addition, sensor $S \in \mathbf{S}$ can be pruned when none of the features it originates is used anymore; a strategy that has not been explored by the state of the art, but that is straightforward with our approach, since it considers the full system.

Precision scaling. We consider four different standard IEEE 754 floating point representations, as they can be implemented in almost any embedded hardware platform. Reducing the precision of arithmetic operations and numerical representations entails information loss and results in performance degradation [35]. The effect on computation costs C_{AC} is clear from Eq. 2.

5.1 Search strategy

Finding the smallest possible AC that computes a given function is Σ_2^p-hard [3], thus computationally harder than NP. No single optimal solution is known for this problem; it is a central question in the field of knowledge compilation [7]. Optimizing for the lowest-cost/highest-accuracy AC, further increases complexity. We therefore opt for a greedy optimization algorithm. Specifically, we rely on a series of heuristics to search the trade-off space. In each step, we independently scale one of the available configuration properties $\langle \alpha, \mathbf{F}, \mathbf{S}, nb \rangle$, as described in the previous section, and aim to find its locally optimal setting. The search begins by learning the model set $\boldsymbol{\alpha}$=$\{\alpha_k\}_{k=1:n}$. Then, as shown by Algorithm 1, starting from each model α_k, we perform a greedy neighborhood search that aims to maximize cost savings and minimize accuracy losses by sequentially pruning the sets $\mathbf{F}$ and $\mathbf{S}$, and simplifying α_k accordingly (Algorithm 2). At each iteration, we evaluate the accuracy and cost of m feature subset candidates, where each considers the impact of removing a feature from the user-defined prunable set $\mathbf{F}_{prun} \subseteq \mathbf{F}$. We then select the feature and sensor subsets $\mathbf{F}_{sel} \subseteq \mathbf{F}$, $\mathbf{S}_{sel} \subseteq \mathbf{S}$ and the simplified model α_{sel} that maximizes the objective function $\mathrm{OF} = acc/cost_{norm}$, where $cost_{norm}$ is the evaluated hardware-aware cost C_{HA}, normalized according to the maximum achievable cost (from the most complex model available α_n). Note that feature subset selection drives sensor subset selection $\mathbf{S}_{sel}$, as described before, and defined in lines 12 and 19 of Algorithm 1.

The output of Algorithm 1, $\langle \mathcal{F}^{(k)}, \mathcal{S}^{(k)}, \mathcal{M}^{(k)} \rangle$, is a set of system configurations of the form $\{\{\mathbf{F}_{sel,1}, \mathbf{S}_{sel,1}, \alpha_{sel,1}\}, \ldots, \{\mathbf{F}_{sel,q}, \mathbf{S}_{sel,q}, \alpha_{sel,q}\}\}$, where q=$|\mathbf{F}_{prunable}|$, and the superscript (k) denotes the number of the input model α_k, taken from $\boldsymbol{\alpha}$. For each configuration resulting from Algorithm 1, we can sweep the available precision configurations $\mathbf{nb}$, for a final space described by $\boldsymbol{\sigma}$=$\langle \mathcal{F}, \mathcal{S}, \mathcal{M}, \mathcal{N} \rangle$ of size $|\boldsymbol{\alpha}| \cdot |\mathbf{F}_{prunable}| \cdot |\mathcal{N}|$, where $\mathcal{N}$ contains the selected precision. In the experimental section we show a work-around to reduce search space size and the number of steps required by the Pareto-optimal search. Regarding complexity, the feature selection in Algorithm 1 is a greedy search, thus its complexity is linear in the number of features times the number of iterations needed for convergence. [3] The AC pruning routine consists of an upward pass on the AC and its complexity is therefore linear in the size of the AC.

[3]In Alg. 1 the user can provide the desired accuracy or cost as the while-loop break criterion.

Algorithm 1: ScaleSI(α_k,$\mathbf{F}_{prun}$,$\mathbf{S}_{prun}$)

```
Input: $\alpha_k$: the $k$th model in $\boldsymbol{\alpha}$ , $\mathbf{F}_{prun}$,$\mathbf{S}_{prun}$: set of
        prunable features and sensors.
Output: $\langle \mathcal{F}^k, \mathcal{S}^k, \mathcal{M}^k \rangle, \mathbf{acc}, \mathbf{cost}$: $k$th collection of
        pruned features, sensors and model sets, their
        accuracy and cost.
1  $\mathbf{F}_{sel} \leftarrow \mathbf{F}_{prun}$, $\mathbf{S}_{sel} \leftarrow \mathbf{S}_{prun}$, $\alpha_{sel} \leftarrow \alpha_k$
2  $\langle \mathcal{F}^k, \mathcal{S}^k, \mathcal{M}^k \rangle \leftarrow \langle \mathbf{F}_{sel}, \mathbf{S}_{sel}, \alpha_{sel} \rangle$
3  $acc_{sel}$=Acc($\alpha_{sel}, \mathbf{F}_{sel}$), $cost_{sel}$=$\mathrm{C_{HA}}(\alpha_{sel}, \mathbf{F}_{sel}, \mathbf{S}_{sel})$
4  $\langle \mathbf{acc}, \mathbf{cost} \rangle \leftarrow \langle acc_{sel}, cost_{sel} \rangle$
5  while $|\mathbf{F}_{sel}| > 1$ do
6      $ob_{max} \leftarrow 0$        // Initialize objective value
7      foreach $F \in \mathbf{F}_{prun}$ do
8          $\mathbf{F}_{ca} \leftarrow \mathbf{F}_{sel} \setminus F$
9          $\mathbf{S}_{ca} \leftarrow \mathbf{S}_{sel}$
10         foreach $S \in \mathbf{S}_{prun}$ do
11             if $\mathbf{F}_{ca} \cap \mathbf{F}_S = \varnothing$ then
12                 $\mathbf{S}_{ca} \leftarrow \mathbf{S}_{ca} \setminus S$        // Prune sensor
13         $\alpha_{ca} \leftarrow$PruneAC($\alpha_{sel}$,$\mathbf{F}_{ca}$)
14         $acc_{ca} \leftarrow$ Acc($\alpha_{ca}$,$\mathbf{F}_{ca}$)
15         $cost_{ca} \leftarrow \mathrm{C_{HA}}$($\alpha_{ca}$,$\mathbf{F}_{ca}$,$\mathbf{S}_{ca}$)
16         $ob_{ca} \leftarrow$ OF($acc_{ca}, cost_{ca}$)
17         if $ob_{ca} > ob_{max}$ then
18             $ob_{max} \leftarrow ob_{ca}$
19             $\mathbf{F}_{sel} \leftarrow \mathbf{F}_{ca}$, $\mathbf{S}_{sel} \leftarrow \mathbf{S}_{ca}$, $\alpha_{sel} \leftarrow \alpha_{ca}$
20             $acc_{sel} \leftarrow acc_{ca}$,$cost_{sel} \leftarrow cost_{ca}$
21     $\mathcal{F}^k$.insert($\mathbf{F}_{sel}$), $\mathcal{S}^k$.insert($\mathbf{S}_{sel}$), $\mathcal{M}^k$.insert($\alpha_{sel}$)
22     $\mathbf{acc}$.insert($acc_{sel}$), $\mathbf{cost}$.insert($cost_{sel}$)
23 return $\langle \mathcal{F}^k, \mathcal{S}^k, \mathcal{M}^k \rangle, \mathbf{acc}, \mathbf{cost}$
```

Algorithm 2: PruneAC(α,$\mathbf{F}$)

```
Input: $\alpha$: the input AC, $\mathbf{F}$: the observed feature set
       used to guide the pruning of $\alpha$.
Output: $\alpha_{pr}$: the pruned AC.
1  $\alpha_{pr} \leftarrow$ copy($\alpha$)
   /* Loop through AC, children before parents   */
2  foreach $a$ in $\alpha_{pr}$ do
3      if $a$ is an indicator variable $\lambda_{F=f}$ and $F \notin \mathbf{F}$ then
4          replace $a$ in $\alpha_{pr}$ by a constant 1
5      else if $a$ is $+$ or $\times$ with constant children then
6          replace $a$ in $\alpha_{pr}$ by an equivalent constant
7  return $\alpha_{pr}$
```

Algorithm 3: GetPareto($\boldsymbol{\sigma}$,**acc**,**cost**)

```
Input: $\boldsymbol{\sigma}$, $\mathbf{acc}$, $\mathbf{cost}$: Configuration set, their accuracy
       and cost.
Output: $\boldsymbol{\sigma}^*, \mathbf{acc}^*, \mathbf{cost}^*$: Pareto optimal
        configurations, their accuracy and cost.
1  $\langle \mathbf{cost}^*, \boldsymbol{\sigma}^*, \mathbf{acc}^* \rangle \leftarrow \langle \{\}, \{\}, \{\} \rangle$ ;
   /* Sort according to ascending cost             */
2  $\langle \mathbf{cost}, \boldsymbol{\sigma}, \mathbf{acc} \rangle \leftarrow sorted(\langle \mathbf{cost}, \boldsymbol{\sigma}, \mathbf{acc} \rangle)$;
3  $i \leftarrow |\boldsymbol{\sigma}| + 1$;
4  while $i > 0$ do
5      $i \leftarrow \arg\max \mathbf{acc}_{0:i}$
6      $\boldsymbol{\sigma}^*$.insert($\sigma_i$)
7      $\mathbf{acc}^*$.insert($\mathbf{acc}_i$)
8      $\mathbf{cost}^*$.insert($\mathbf{cost}_i$)
9      $i \leftarrow i - 1$
10 return $\boldsymbol{\sigma}^*$, $\mathbf{acc}^*$, $\mathbf{cost}^*$
```

5.2 Pareto-optimal configuration selection

Algorithm 3 describes how we extract the Pareto-optimal configuration subset, but any convex hull algorithm can be used. The input is the configuration set $\boldsymbol{\sigma}$=$\langle \mathcal{F}, \mathcal{S}, \mathcal{M}, \mathcal{N} \rangle$ and their corresponding accuracy ($\mathbf{acc}$) and cost ($\mathbf{cost}$). The output of this algorithm is the set of Pareto-optimal system configurations $\boldsymbol{\sigma}^*$=$\{\{\alpha_i^*, \mathbf{F}^*{}_i, \mathbf{S}^*{}_i, nb_i^*\}_{i=1:p}\}$, each guaranteed to achieve the largest reachable accuracy for any given cost; or the lowest reachable cost for any given accuracy ($\mathbf{acc}^*$, $\mathbf{cost}^*$).

Note that the framework introduced thus far can balance the trade-off between hardware-aware cost and any other application-specific performance metric, by simply replacing the accuracy terms in Algorithms 1, 2 and 3 with such a metric. For instance, medical applications often aim to balance precision and recall, and may favor the latter at night under scarce medical supervision. Furthermore, our framework can be used density estimation tasks by deploying the model complexity scaling followed by precision scaling stages and forgoing the pruning stages of Algorithms 1 and 2, in order to keep the full joint distribution. The next section illustrates how our methodology can reap the benefits of scalable embedded hardware.

6 Experimental evaluation

We empirically evaluate the proposed techniques on a relevant embedded sensing use case: the Human Activity Recognition (HAR) benchmark [1]. Additionally, we show our method's general applicability on a number of other publicly available datasets [8, 14, 20, 25, 30], two of them commonly used for density estimation benchmarks and the rest commonly used for classification (see Table 1).[4]

Computational costs. The computation costs $\mathrm{C_{AC}}$ are based on the energy benchmarks discussed in [17] and [35]. Table 2 shows the relative costs of each term in $\mathrm{C_{AC}}$ and how they scale with

[4]Code available at `https://github.com/laurago894/HwAwareProb`.

precision nb. The baseline is 64 floating point bits because it is the standard IEEE representation in software environments. For the rest of the experiments, we consider three other standard low precision representations: 32 bits (8 exponent and 24 mantissa), 16 bits (5 exponent and 11 mantissa) and 8 bits (4 exponent and 4 mantissa) [19].

Dataset pre-processing. For the classification benchmarks, we discretized numerical features using the method in [9]. We then binarized them using a one-hot encoding and subjected them to a 75%-train, 10%-validation and 15%-test random split. For the HAR benchmark, we preserved the timeseries information by using the first 85% samples for training and validation and the last for testing. For the density estimation datasets, we used the splits provided in [25] and we assumed the last feature in the set to be the class variable. On all datasets, we performed wrapper feature selection (evaluating the features' value on a Tree Augmented Naive Bayes classifier) before going through the hardware-aware optimization process to avoid over-fitting on the baseline model and ensure it is a fair reference point. The number of effectively used features $|\mathbf{F}|$ is shown in Table 1. In addition, we consider all the features to be in the prunable set $\mathbf{F}_{prun}$ for datasets with less than 30 features. For the rest, we consider the 20 with the highest correlation to the class variable. Within the context of an application, the prunable set can be user-defined. For instance, in a multi-sensor seizure detection application, medical experts might advise against pruning features extracted from an EEG monitor.

Table 1: Experimental datasets
†: Classification , ⋆: Density est.

Dataset	$\lvert\mathbf{F}\rvert$	$\lvert\mathbf{F}_{prun}\rvert$	$\lvert\boldsymbol{\alpha}\rvert$
Banknote†	15	15	11
HAR †	28	28	11
Houses †	36	20	11
Jester ⋆	99	20	11
Madelone †	20	20	11
Nltcs ⋆	15	15	11
Six-HAR †	54	20	11
Wilt †	11	11	11

Model learning We learned the models on the train and validation sets with the LearnPSDD algorithm [23], using the same settings reported therein, and following the bottom-up vtree induction strategy. To populate the model set $\boldsymbol{\alpha}$, we retained a model after every $N/10$ iterations, where N is the number of iterations needed for convergence (in the algorithm this is until the log-likelihood on validation data stagnates). Table 1 shows $|\boldsymbol{\alpha}|$ for each dataset. Furthermore, as a baseline, we trained a Tree Augmented Naive Bayes (TAN) classifier and compiled it to an AC.[5]

Table 2: Experiment computational costs.

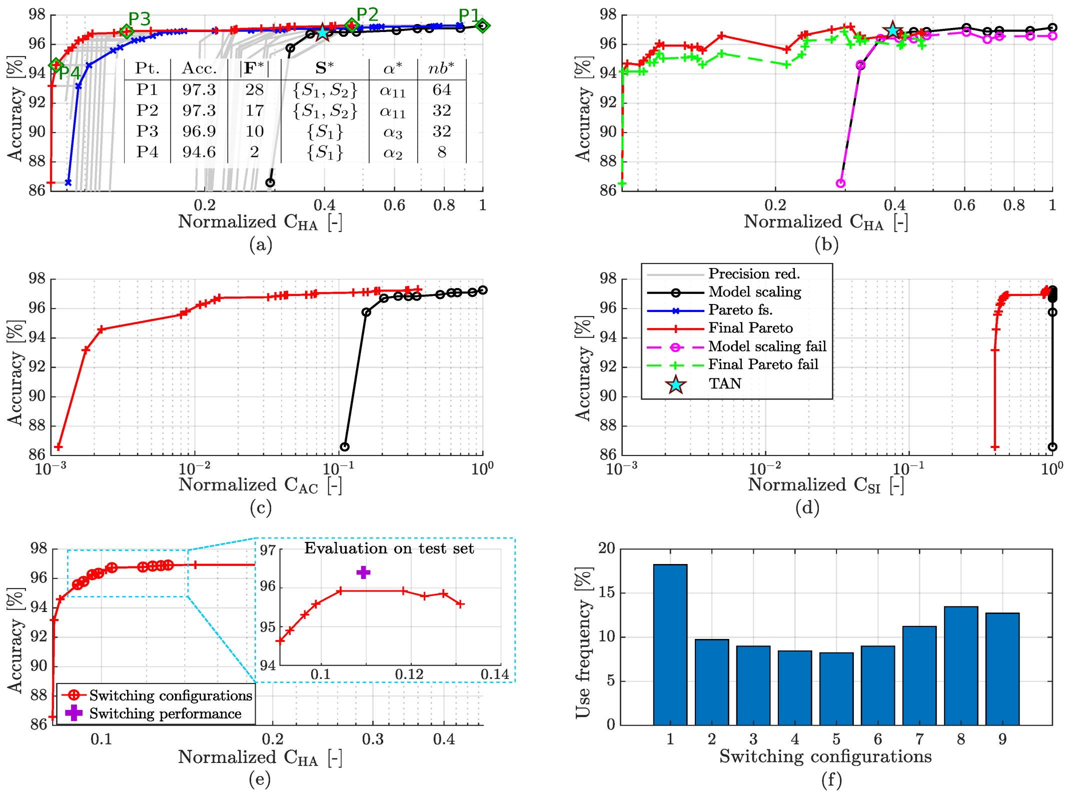
Operation	At 64 bits	Operation cost
$\mathrm{C}_{\mathrm{mem}}$	1	$\phi_{mem} = \gamma_{mem} \cdot nb$
$\mathrm{C}_{\mathrm{cache}}$	0.2	$\phi_{cache} = \gamma_{cache} \cdot nb$
$\mathrm{C}_{\times}$	0.6	$\phi_{\times} = \gamma_{\times}^2 \cdot nb^2 \cdot \log(nb)$
C_{+}	0.1	$\phi_{+} = \gamma_{+} \cdot nb$

6.1 Embedded Human Activity Recognition

The HAR dataset aims to recognize the activity a person is performing based on statistical and energy features extracted from smartphone accelerometer and gyroscope data. We perform binary classification by discerning "walking downstairs" from the other activities. For the experiments, we use a total of 28 binary features, 8 of which are extracted from the gyroscope's signal and the rest from the accelerometer, as detailed in Appendix B.4. All computation costs for this dataset are normalized according to the energy consumption trends of an embedded ARM M9 CPU, assuming 0.1nJ per operation [38]. Sensors are estimated to consume 2mWatt, while the costs of all features is defined as 30 MAC operations (see Appendix B.1 for more details).

Pareto optimal configuration. This experiment consisted of three stages, performed on the training set (Figure 3(a)): 1) We first mapped each model in $\boldsymbol{\alpha}$ to the trade-off space, as shown in black. 2) Starting from each model, we scaled the feature and the sensor sets $\mathbf{F}$, $\mathbf{S}$, as shown in blue. 3) We then scaled the precision nb of each of these pruned configurations (shown by the grey curves) and we finally extracted the Pareto front shown in red. As shown by the Pareto configurations highlighted in green, our method preserves the highest baseline train-set accuracy by pruning 11 of the available 28 features, which results in C_{HA} savings of 53%. When willing to tolerate further accuracy losses of 0.4%, our method outputs a configuration that consumes only 13% of the original cost by using a smaller model (α_3), pruning 18 features, turning one sensor off and using a 32 bit representation. Figures 3(c,d) break down the computational cost C_{AC} and the sensor costs C_{SI}. When considering only the costs of the AC evaluation (graph (c)), our method results in savings of almost two orders of

[5]Using the ACE compiler available at `http://reasoning.cs.ucla.edu/ace/`.



Pt.	Acc.	$\lvert\mathbf{F}^*\rvert$	$\mathbf{S}^*$	α^*	nb^*
P1	97.3	28	$\{S_1, S_2\}$	α_{11}	64
P2	97.3	17	$\{S_1, S_2\}$	α_{11}	32
P3	96.9	10	$\{S_1\}$	α_3	32
P4	94.6	2	$\{S_1\}$	α_2	8

Figure 3: Experiments on the Human Activity Recognition benchmark.

magnitude with accuracy losses lower than 1%, and up to 3 orders of magnitude when tolerating more losses. Indeed, these computational cost savings result from the joint effects of precision reduction and of simplifying the AC with Algorithm 2, with savings of about 2 to 10 % per feature pruned (see Appendix B.2). Sensor and feature costs, as shown in graph (d) only scale up to 50%, since at least one of the sensors must always be operating. This demonstrates the importance of taking these costs into account: even though computation costs savings are impressive, the system is still limited by the sensing costs.

Robustness and online deployment. The red curve in Figure 3(b) shows that the evaluation of the selected Pareto configurations against testing data stays within a range of $\pm 1\%$ with respect to Figure 3(a). Comparing our method to the TAN classifier denoted with the cyan marker, we can see that it provides further cost saving opportunities, while achieving competitive accuracy. We also assessed the robustness of our method by simulating, per configuration, ten iterations of random failure of varying sizes of feature sets ($|\mathbf{F}|/10$,$|\mathbf{F}|/5$,$|\mathbf{F}|/2$). The green and magenta dotted curves show the median of these experiments for the Pareto configurations and for the original model set. These trials stay within a range of -2% compared to the fully functional results in red and black, which validates our choice of a generative learner that can naturally cope with missing features at prediction time.

In embedded sensing scenarios, environmental circumstances, power consumption requirements and accuracy constraints commonly vary over time. This calls for dynamic operating settings, where the system can switch between different accuracy-cost operating points at run-time. Figure 3(e) shows such a scenario for nine operating points selected off-line (as highlighted with circular markers in the background graph), which comply with hypothetical user needs of accuracy>95% and normalized cost<20%. The implemented policy assumes an energy efficient mode when the classifier has recently identified that there is no ongoing activity, and a high reliability – and costlier – mode when it has identified that there is ongoing activity (see Appendix B.3 for more details). The foreground of Figure 3(e) contrasts the test-set cost-accuracy performance attained when always using the same model (in red), with the cost-accuracy performance resulting from the implementation of our model-switching policy (purple cross). Even with its simplicity, the proposed policy attains accuracy vs. cost improvements that go beyond the static Pareto front. Figure 3(f) shows that this

Table 3: Results for benchmarking datasets [$\mathrm{C_{AC}}$,$\mathrm{Acc_{tr}}$%,$\mathrm{Acc_{te}}$%].

Dataset	Operating pt. 1	Operating pt.2	Operating pt. 3	Operating pt. 4	TAN
Banknote	[1,94.5,95.6]	[0.09,94.5,95.6]	[0.04,89.9,93.1]	[0.01,84.5,86.8]	[0.24,93.8,92.2]
Houses	[1,97.6,97.4]	[0.12,97.6,97.3]	[0.04,97.1,96.6]	[0.01,94.3,94.0]	[0.05,97.2,97.1]
Jester	[1,75.6,76.4]	[0.35,75.6,76.4]	[0.12,74.7,75.7]	[0.02,72.6,73.1]	[0.12,73.1,72.3]
Madelone	[1,68.1,68.4]	[0.05,68.6,69.1]	[0.02,66.9,68.8]	[0.01,62.6,62.9]	[0.13,66.0,65.7]
Nltcs	[1,93.5,93.9]	[0.19,93.6,93.8]	[0.03,93.4,94.2]	[0.01,91.7,92.0]	[0.11,91.4,91.9]
Six-HAR	[1,91.5,89.8]	[0.38,91.6,89.9]	[0.15,89.3,89.3]	[0.04,89.8,89.8]	[0.36,91.7,90.3]
Wilt	[1,97.1,97.5]	[0.07,97.1,97.5]	[0.03,97.1,97.5]	[0.01,96.9,97.5]	[0.25,97.1,97.5]

is achieved by making a balanced use of the nine available configurations. Note that this switching action incurs overhead only in terms of memory since the set of Pareto switching configurations is always determined off-line, and will be only fetched when needed. In most portable applications, predictions must be made at a much higher frequency than configuration changes are necessary [12]. The incurred memory overhead in our experiments is less than 3% of the total cost, since model switching is only necessary on 120 out of the 1470 predictions.

6.2 Generality of the method: evaluation on benchmark datasets

We now apply our optimization sequence to the datasets in Table 1. For lack of information on the hardware that originated these datasets, we only consider the computation cost $\mathrm{C_{AC}}$, again evaluated on the cost model of the ARM M9 CPU. Table 3 shows this cost along with the training and testing accuracy ($\mathrm{Acc_{tr}}$,$\mathrm{Acc_{te}}$) at four operating points for every dataset. Note that we have also included the six-class HAR benchmark, to demonstrate the applicability of our method beyond binary classification. We can see that all the benchmarks strongly benefit from our proposed methodology, that they are all robust when contrasted against the test dataset, and that they are competitive when compared to a TAN classifier. Appendix A shows a figure with the Pareto fronts for all the experiments herewith.

7 Related work

The problem of hardware-efficient probabilistic inference has been addressed by the probabilistic-models and the embedded-hardware communities from several perspectives. The works by Tschiatschek and Pernkopf [39] and Piatkowski et al. [31] propose reduced precision and integer representation schemes for PGMs as a strategy to address the constraints of embedded devices. In [35], Shah et al. propose a framework that automatically chooses an appropriate energy-efficient low precision representation and generates custom hardware. Other efficient hardware implementation efforts have been made by Zermani et al. [42], Schuman et al. [34], and Sommer et al. [36], who have proposed to accelerate inference on SPNs, capitalizing on their tractable properties.

Our work constitutes an effort to integrate the expertise from both communities under a unified framework, which considers the impact of all scalable aspects of the model to optimize it in a hardware-aware fashion. To that end, it leverages the properties of the selected AC representation. Such representation enables the use of our framework with any probabilistic model that is compute-efficient at prediction time: see [16] by Holtzen et al. and [40] by Vlasselaer et al. for examples of probabilistic program compilation to ACs; and [27] by Lowd and Rooshenas on how to perform efficient inference with Markov networks represented as ACs.

8 Conclusions

We proposed a novel hardware-aware cost metric to deal with the limitations of the efficiency vs. performance trade-off considered by the field of tractable learning. Our method obtains the Pareto-optimal system-configuration set in the hardware-aware cost vs. accuracy space. The proposed solution consists of a sequential hardware-aware search and a Pareto-optimal configuration selection stage. Experiments on a variety of benchmarks demonstrated the effectiveness of the approach and sacrifice little to no accuracy for significant cost savings. This opens up opportunities for the efficient implementation of probabilistic models in resource-constrained edge devices, operating in dynamic environments.

Acknowledgements

We thank Yitao Liang for the LearnPSDD algorithm and for helpful feedback. This work is partially supported by the EU ERC Project Re-SENSE under Grant ERC-2016-STG-71503; NSF grants #IIS-1633857, #CCF-1837129, DARPA XAI grant #N66001-17-2-4032, NEC Research, and gifts from Intel and Facebook Research.

References

[1] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2013.

[2] F. R. Bach and M. I. Jordan. Thin junction trees. In *Advances in Neural Information Processing Systems*, pages 569–576, 2002.

[3] D. Buchfuhrer and C. Umans. The complexity of boolean formula minimization. In *International Colloquium on Automata, Languages, and Programming*, pages 24–35. Springer, 2008.

[4] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[5] Y. Choi and G. Van den Broeck. On robust trimming of bayesian network classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.

[6] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[7] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[8] D. Dua and C. Graff. UCI machine learning repository, 2017.

[9] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

[10] D. Fierens, G. Van den Broeck, I. Thon, B. Gutmann, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted CNFs. *Theory and Practice of Logic Programming*, 15, 2015.

[11] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[12] L. Galindez Olascoaga, K. Badami, J. Vlasselaer, W. Meert, and M. Verhelst. Dynamic sensor-frontend tuning for resource efficient embedded classification. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4):858–872, Dec 2018.

[13] R. Gens and P. Domingos. Learning the structure of sum-product networks. In *International conference on machine learning*, pages 873–880, 2013.

[14] I. Guyon and S. Gunn. Nips feature selection challenge, 2003.

[15] D. Heckerman. Causal independence for knowledge acquisition and inference. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 122–127. Elsevier, 1993.

[16] S. Holtzen, T. Millstein, and G. Van den Broeck. Symbolic exact inference for discrete probabilistic programs. In *Proceedings of the ICML Workshop on Tractable Probabilistic Modeling (TPM)*, June 2019.

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

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[19] IEEE. Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008.

[20] B. A. Johnson, R. Tateishi, and N. T. Hoan. A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International journal of remote sensing*, 34(20):6969–6982, 2013.

[21] O. U. Khan and D. D. Wentzloff. Hardware accelerator for probabilistic inference in 65-nm cmos. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):837–845, 2016.

[22] P. Khosravi, Y. Liang, Y. Choi, and G. Van den Broeck. What to expect of classifiers? reasoning about logistic regression with missing features. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, aug 2019.

[23] Y. Liang, J. Bekker, and G. Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[24] Y. Liang and G. Van den Broeck. Learning logistic circuits. *Proceedings of the 33rd Conference on Artificial Intelligence (AAAI)*, 2019.

[25] D. Lowd and J. Davis. Learning markov network structure with decision trees. In *2010 IEEE International Conference on Data Mining*, pages 334–343. IEEE, 2010.

[26] D. Lowd and P. Domingos. Learning arithmetic circuits. *arXiv preprint arXiv:1206.3271*, 2012.

[27] D. Lowd and A. Rooshenas. Learning markov networks with arithmetic circuits. In *Artificial Intelligence and Statistics*, pages 406–414, 2013.

[28] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018.

[29] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst. Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos. In *2018 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4. IEEE, 2018.

[30] R. K. Pace and R. Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[31] N. Piatkowski, S. Lee, and K. Morik. Integer undirected graphical models for resource-constrained systems. *Neurocomputing*, 173:9–23, 2016.

[32] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[33] A. Rooshenas and D. Lowd. Learning sum-product networks with direct and indirect variable interactions. In *International Conference on Machine Learning*, pages 710–718, 2014.

[34] J. Schumann, K. Y. Rozier, T. Reinbacher, O. J. Mengshoel, T. Mbaya, and C. Ippolito. Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. *International Journal of Prognostics and Health Management*, 2015.

[35] N. Shah, L. I. Galindez Olascoaga, W. Meert, and M. Verhelst. Problp: A framework for low-precision probabilistic inference. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 190. ACM, 2019.

[36] L. Sommer, J. Oppermann, A. Molina, C. Binnig, K. Kersting, and A. Koch. Automatic mapping of the sum-product network inference problem to fpga-based accelerators. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 350–357. IEEE, 2018.

[37] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[38] S. Tarkoma, M. Siekkinen, E. Lagerspetz, and Y. Xiao. *Smartphone energy consumption: modeling and optimization*. Cambridge University Press, 2014.

[39] S. Tschiatschek and F. Pernkopf. On bayesian network classifiers with reduced precision parameters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(4):774–785, 2015.

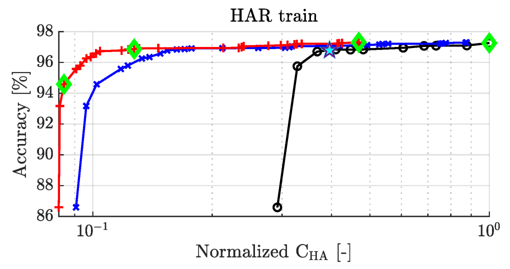
[40] J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt. Tp-compilation for inference in probabilistic logic programs. *International Journal of Approximate Reasoning*, June 2016.

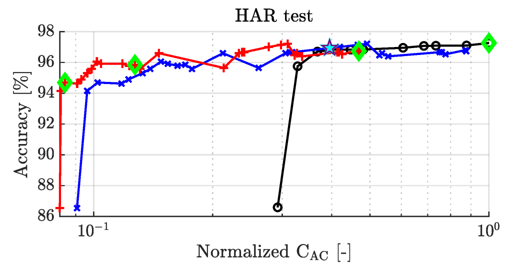
[41] J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, July 2018.

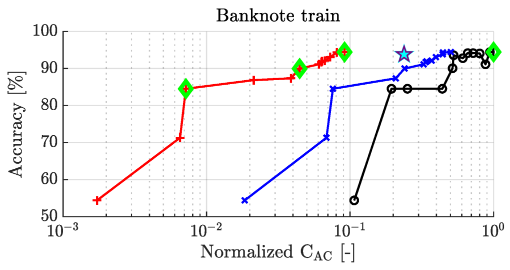
[42] S. Zermani, C. Dezan, R. Euler, and J.-P. Diguet. Bayesian network-based framework for the design of reconfigurable health management monitors. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8. IEEE, 2015.

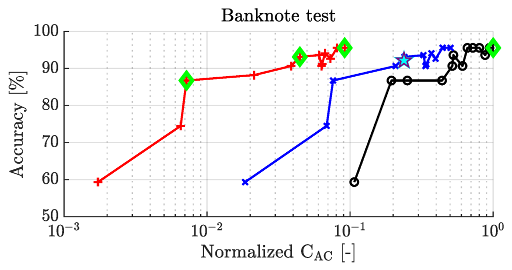
A. Experiments on benchmarking datasets

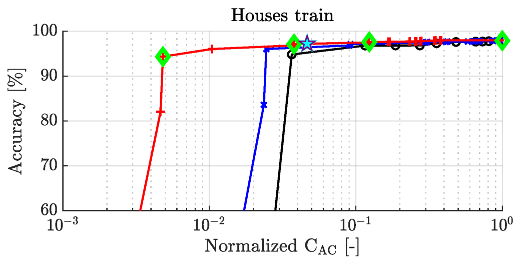
Cost vs. accuracy mapping for all datasets: Pareto curves obtained by the proposed method on the left, and their evaluation on the test dataset on the right. The black line represents the models complexity scaling step, the blue one represents the sensor interface scaling stage and the red one the final Pareto optimal configuration set. The cyan star corresponds to the performance of a Tree Augmented Naive Bayes (TAN) classifier after compilation to Arithmetic Circuit representation, and the green markers indicate the data points reported in the paper.

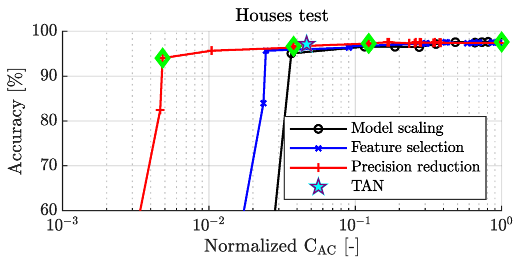


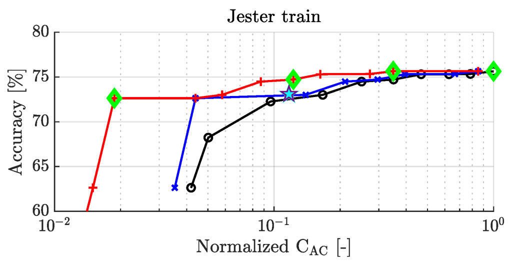


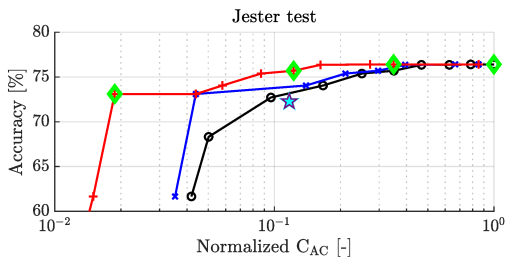


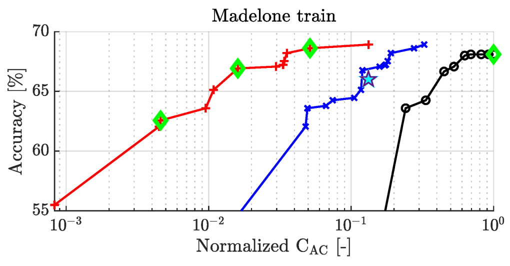


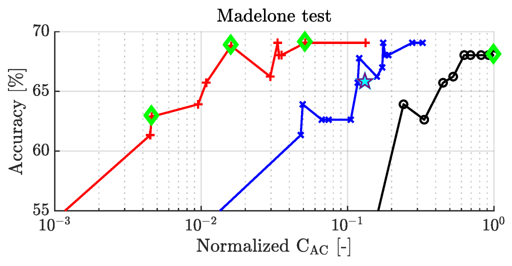


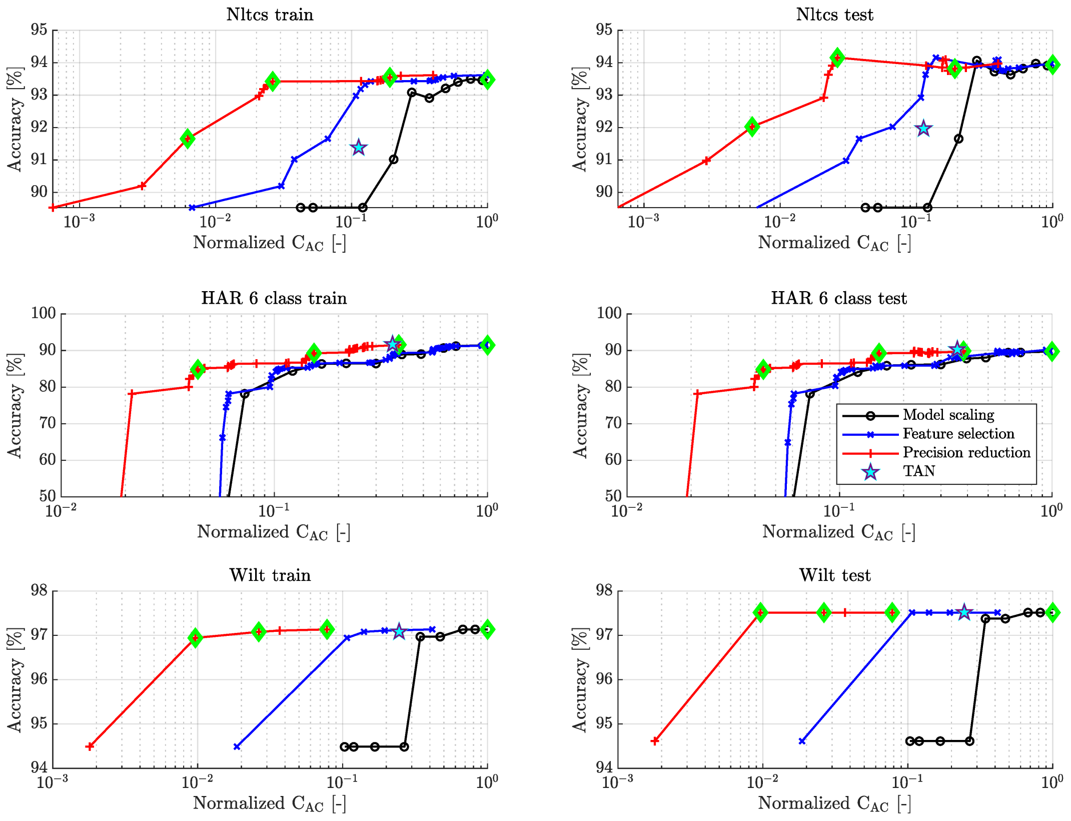












B. Experiment on the benchmark HAR

B.1 Sensor and feature costs

We assume that feature extraction and classification take place in an ARM 9 CPU, and we thus define these costs relative to computational costs.

Sensor costs We assume that feature extraction and classification take place in the aforementioned CPU, which consumes, on average, 1 Watt to execute 10G operations per second. Thus it will consume approximately 0.1nJ per operation of the CPU.[1] Considering that the largest AC requires about 20000 operations per instance classified, we assume a total computational energy consumption of 2μJ. We assume that both the gyroscope and the accelerometer consume at least 2 mW when operating at 10KHz, and that they thus consume roughly 0.2μJ per operation of the CPU.[2] Thus, we set the cost of the sensors relative to the computation costs: each sensor has a cost 10% the total cost of classifying a single instance in the most complex model available.

Feature costs The features of this dataset are extracted by sampling the sensory signal, applying three low-pass filters and calculating statistical quantities (mean,maximum/minimum,correlation and standard deviation) on the resulting signal. Sampling and extracting the statistical features require a small number of operations in comparison to filtering. For example, calculating the mean of a sample requires a single MAC (Multiply-Accumulate, consisting of a multiplication and an addition) operation, whereas a 3rd order low pass filter will require at least nine. Filtering thus takes the bulk of the computations, so we assume that each feature extraction incurs in a cost of 30 MAC operations.

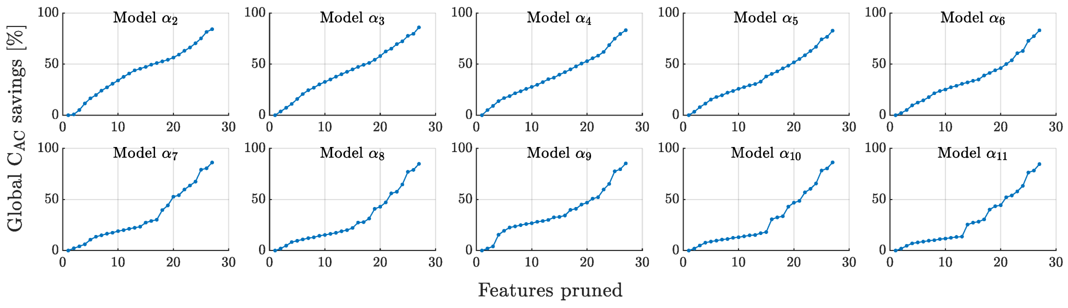
[1]https://developer.arm.com/products/processors/classic-processors

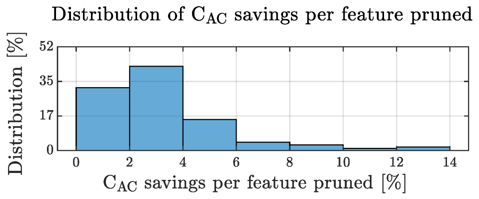
[2]Gyroscopes can consume as much as 10 times more energy than accelerometers, but we can assume that they are both part of a larger system, such as a Inertial Measurement Unit, hence we assign the same cost to both.

B.2 Computational costs savings per feature pruned

The AC pruning strategy of Algorithm 2 is driven by feature selection, i.e. every time a feature is pruned, the model can be reduced by pre-computing operations related to unobserved indicator leaf nodes. The figure below shows an analysis of global $\mathrm{C_{AC}}$ savings per feature pruned, for every model used in experiments.

The histogram shows local $\mathrm{C_{AC}}$ savings: in most cases, every time a feature is pruned, $\mathrm{C_{AC}}$ is reduced by 1% to 5 %, and up to 13%.





B.3 Model switching experiment

The Pareto-configuration model set used in this experiment was selected according to hypothetical user's needs: "accuracy must be larger than 95% and the normalized cost lower than 20%".

The policy we proposed, assumes that classifying "no activity" is a lower complexity and lower stakes task and therefore gives priority to the task of classifying "activity": The model remains fixed from $t-11$ to $t-1$. At time t, the policy switches to a more complex model when the prediction is "activity", and it switches back to a simpler model when the prediction is "no activity".

B.4 Features used

1 abs energy-acc.x-value1
2 abs energy-acc.x-value2
3 absolute sum of changes-acc.x-value1
4 absolute sum of changes-acc.x-value2
5 maximum-acc.x-value1
6 maximum-acc.x-value2
7 maximum-acc.x-value3
8 skewness-acc.x-value4
9 standard deviation-acc.y-value1
10 standard deviation-acc.y-value2
11 standard deviation-acc.y-value3
12 sum values-acc.y-value1
13 count above mean-acc.z-value1
14 count below mean-acc.z-value1
15 mean change-acc.x-value1
16 longest strike above mean-acc.z-value1
17 longest strike below mean-acc.z-value1
18 maximum-gyro.z-value1
19 median-gyro.z-value1
20 median-gyro.z-value2
21 median-gyro.z-value3
22 median-gyro.z-value4
23 number crossing-acc.z-value1
24 standard deviation-gyro.z-value1
25 median-acc.x-value1
26 median-acc.x-value2
27 standard deviation-gyro.x-value1
28 standard deviation-gyro.y-value1