A Hardware Perspective to Evaluating Probabilistic Circuits

Jelin Leslin jelin.leslin@aalto.fi
Antti Hyttinen antti.hyttinen@helsinki.fi
Karthekeyan Periasamy karthekeyan.periasamy@aalto.fi
Lingyun Yao lingyun.yao@aalto.fi
Martin Trapp martin.trapp@aalto.fi
Martin Andraud martin.andraud@aalto.fi

Abstract

The always-increasing development of AI-enhanced Internet-of-Things devices has recently pushed the need for on-device computation of AI models. As these tasks require making robust predictions under uncertainty, probabilistic (graphical) models have recently gained interest also for these applications. However, embedded computation requires high computational efficiency (*i.e.*, high speed and low power) through hardware acceleration. Although the acceleration of deep learning models has shown extensive benefits, this has not translated to probabilistic models as of yet. Probabilistic circuits (PCs), a family of tractable probabilistic models, allow a direct hardware view as they are represented in the form of a computational graph. Over the years, various approaches for structure learning of PCs have been proposed, however, without consideration of their potential hardware cost. In this work, we propose to take a hardware perspective in the evaluation of PC structures. We compare several structure learning strategies, associating each PC with hardware costs (computation power, speed, efficiency), and evaluate which one leads to more hardware-friendly implementations. Our results show that models imposing additional structural constraints on the PC are competitive models in terms of performance while being generally more hardware-efficient, making them suitable candidates for energy-constrained applications.

Keywords: Probabilistic circuits, Hardware accelerators, Structure learning

1. Introduction

In the last decade, we have entered an *internet-of-things* (IoT) era with AI-enhanced connected devices flourishing all around us (Mohammadi et al., 2018). The next generation of IoT devices will need to be smarter and more energy-efficient, for example enabling AI-assisted 5G or 6G communication devices (Mahmood et al., 2020). Smarter, to increase their level of awareness and decision-making. More energy-efficient, as they would embed AI directly on the device, providing several benefits. Firstly, this will minimize network latency and increase data privacy, as data does not need to be exchanged to the cloud. Secondly, it will significantly increase energy-efficiency and sustainability, as all computations can be carried out on-device, and no more communication with the cloud would be needed. However, generic processors are not always well suited to efficiently compute AI workloads, which consist of simple and repetitive operations (see a detailed explanation in

Section 3). For these reasons, dedicated compute platforms, known as hardware *accelerators*, have recently flourished, in particular targeting neural networks (NNs) (Deng et al., 2020).

Yet, for reliable AI-enhanced IoT, it is crucial for devices to be able to reason under uncertainty. For example, a health monitoring device that should detect anomalies and react in case of emergency. This system would gather information about the neighbouring environment through sensors, extract features, process this information, and reach conclusions about the task at hand using, *e.g.*, a pre-trained classifier. This system would operate under different forms of uncertainty related, for instance, to patient-specific issues or the quality of the information received while monitoring (Mohammadzadeh and Safdari, 2014). This can be translated to requirements for the embedded AI model: 1.) reason under uncertainty, as uncertainty is unavoidable in real-world applications, 2.) answer different questions/queries with the same model, as online retraining is challenging, and 3.) enable efficient hardware implementation to be used on resource-constrained devices. A modelling family satisfying these requirements are probabilistic circuits (PCs), a recently introduced family of tractable probabilistic models. PCs enable exact and efficient computation of many probabilistic inference queries, such as the computation of marginals.

Although there is an extensive literature comparing probabilistic models and PCs in terms of predictive performance, *e.g.*, in (Butz et al., 2018; Scutari et al., 2018, (both in the PGM conference)), less focus has been put on defining, evaluating and comparing PCs regarding their *hardware-friendly* nature, *i.e.*, their ability to be implemented on hardware. In this work, we first propose a detailed analysis on the computation of PCs on hardware to understand the challenges related to the acceleration of PCs. Then, we build on previous hardware-aware training methods proposed for specific PCs models (Shah et al., 2019; Olascoaga et al., 2019; Sommer et al., 2018) and extend the analysis toward a systematic comparison of different structure learning algorithms. In addition, we extend the hardware cost to also consider parallelism abilities and elaborate on hardware reuse, specifically targeting new vectorized PC models such as Einsum networks (Peharz et al., 2020). The associated research questions related to this work are: (A) How should we learn the structure of a PC if it will be implemented in hardware? (B) Which structure learning strategy helps in achieving hardware-friendly models? (C) Should we constrain the structure of the PC for better hardware implementation?

2. Probabilistic Circuits and Their Learning Methods

Probabilistic circuit (PC) is an umbrella term that has been introduced in Choi et al. (2020) to unify existing models (*e.g.*, Poon and Domingos (2011); Kisa et al. (2014); Peharz et al. (2020)) of the same tractable model family. In PCs, the directed acyclic graph (DAG) structure encodes dependencies between (latent) variables as well as the computational alignment.

The structure of a PC is commonly learned from data (*e.g.*, Gens and Domingos (2013); Peharz et al. (2020)), and the PC is parametrised with a set of weights ($\mathbf{w}$) and parameters (θ) at the input distributions. A PC on a set of RVs ($\mathbf{X} = \{X_d\}_{d=1}^{D}$) is a tuple $(\mathcal{G}, \psi)$ consisting of a computational graph $\mathcal{G}$, given by a DAG, and a scope function $\psi\colon \mathbf{N} \to \mathcal{P}(\mathbf{X})$ that assigns each node N in $\mathcal{G}$ a scope (*i.e.*, a subset or the set $\mathbf{X}$). At a minimum, a PC compromises three types of nodes: sums, products and leaves. Sum nodes compute a convex combination of their children ($\mathrm{S}(\mathbf{x}) = \sum_{\mathrm{N}\in\mathrm{ch(S)}} w_{\mathrm{S,N}}\,\mathrm{N}(\mathbf{x})$), product nodes encode

independence assumptions ($\mathrm{P}(\mathbf{x}) = \prod_{\mathrm{N}\in\mathrm{ch(S)}} \mathrm{N}(\mathbf{x})$), and leave nodes evaluate their associated input distribution ($\mathrm{L}(\mathbf{x}) = p(\mathbf{x} \mid \theta_{\mathrm{L}})$). For hardware-compatibility reasons, we will assume that leave nodes are equipped with indicator functions, *i.e.*, the PC represents a discrete distribution.

Once the PC has been learned and deployed onto hardware, many queries can be answered without having to re-train the model, *i.e.*, a PC is a multi-purpose probabilistic model. The set of queries that can be answered tractably is related to constraints on the PC structure. We briefly review the most relevant constraints for this work.

Definition 1 (smoothness) *A sum node is* smooth *if its children depend on the same variables:* $\psi(\mathrm{S}) = \psi(\mathrm{N}), \forall \mathrm{N} \in \mathrm{ch(S)}$. *A PC is* smooth *if all of its sum nodes are* smooth.

Definition 2 (decomposability) *A product node is* decomposable *if the scopes of its children do not share variables:* $\psi(\mathrm{N}) \cap \psi(\mathrm{N}') = \emptyset, \forall \mathrm{N}, \mathrm{N}' \in \mathrm{ch(P)}, \mathrm{N} \neq \mathrm{N}'$. *A PC is* decomposable *if all of its product nodes are* decomposable.

Definition 3 (structured-decomposability) *Given a* vtree*, a binary tree encoding a hierarchical decomposition of RVs, a PC is* structured-decomposable *if every product node decomposes its scope as its corresponding node in the vtree.*

In general, we assume that a PC is *smooth* and *decomposable*, guaranteeing that, *e.g.*, marginals, can be computed tractably. Moreover, in our analysis, we also consider *smooth* and *structured-decomposable* PCs, which render a broader set of probabilistic queries tractable than *smooth* and *decomposable* circuits, to investigate possible benefits in terms of their hardware costs. We refer to Choi et al. (2020) for a more detailed discussion of the structural constraints in PC and their implications on tractability.

2.1 Specific PC Learning Methods Compared in This Work

This comparative study uses various PC models, which all have achieved competitive results in the literature. We focus our evaluation on *hardware characteristics* of PCs (number and type of nodes, depth, parallelism, etc.), which are closely linked to the method employed for structure learning and thus heavily depend on the chosen PC model. Note that, even though methodologies using ensembles of PCs achieve state-of-the-art performance (Trapp et al., 2019; Dang et al., 2020), we focus on single PC models, as we wish to first evaluate in detail the hardware characteristics of the different learning methodologies.

LearnSPN LearnSPN, initially proposed in (Gens and Domingos, 2013), is a recursive structure learning framework. At each step, the algorithm attempts to divide the current scope into approximately independent subsets. If successful, it returns a product of recursive calls on the subsets; otherwise, it clusters the data and returns the sum of recursive calls on a cluster from the current training subset. LearnSPN is a popular structure learning algorithm due to its simplicity of implementation (París et al., 2022).

Einets Einsum networks (Einets) (Peharz et al., 2020) use a vectorized representation of the PC. One Einet is composed of several layers, each performing an Einstein summation (einsum). Essentially, in Einets a sum and product layer of a scalar PC are concatenated

into a single vectorized computational unit. It consists of one vector sum node with a single child, being the product of two other vector children. The Einet structure can be determined in various ways, one being based on randomized trees (RAT-SPNs) (Peharz et al., 2019). Here, we use the Einsum implementation of (Peharz et al., 2020) based on RAT-SPN structures governed by shared hyperparameters. The global Einet structure is determined by the following hyperparameters: the number of elements in each vector K, the number of replicas R and the split-depth D. These parallelization properties enable Einets to have more computationally efficient training and inference, while being competitive in terms of performance, thus making them suitable for hardware implementation.

SDPCs Structured-decomposable PCs (SDPCs) (Pipatsrisawat and Darwiche, 2008) are models imposing a stronger structural constraint in favour of additional tractability. SDPCs are frequently used in the literature, taking, for instance, the form of Probabilistic Sentential Decision Diagrams (PSDDs) (Kisa et al., 2014). In this work, we use Strudel (Dang et al., 2020) as a reference method to construct them. In Strudel, SDPCs are constructed by finding a "best" initial PC, in the form of a Chow-Liu tree (CLT), and distilling a vtree from it to compile the CLT into a SDPC. Finally, an ensemble of SDPC structures is formed. we focus here on a single SDPC structure, following the first steps of Strudel, to evaluate how adding a structured-decomposability constraint impacts the hardware footprint.

Bayesian networks/ACs In contrary to the previous methods, ACs are not learned from data but compiled from Bayesian networks (BNs). Thus, in this work, we consider two basic BN structure-learning approaches: max-min hill climbing of the bnlearn R-package (Scutari, 2010) and Bayesian search of the Genie software. However, such methods may produce structures with high tree-widths and thus inference may not be tractable. Thus, we also consider a recent method (Benjumeda et al., 2019) that can enforce different treewidth bounds and hence guarantee the tractability of inference. For all methods, ACs are compiled through ACE (UCLA, 2015). We evaluate how the tree-width bound relates to the efficiency of the hardware implementation and how such ACs differ from other PC models.

3. Hardware Computations of PCs

Designing efficient processors dedicated to AI computation has recently gained significant interest. One main reason explaining this is that *generic* central processing units (CPUs), optimized for sequential and flexible computing of various operations, do not perform well for AI workloads that typically consist of simple and repetitive operations. This is illustrated for a simplistic fully connected neural network (NN) layer in Fig. 1(a): each neuron performs a weighted-sum of its inputs multiplied by their respective weights, or Multiply-and-accumulate (MAC), followed by a non-linear activation function. In a simple CPU, for each operation the data is fetched from and written back to the memory. These memory transfers heavily dominate the final computational cost, as transferring data can cost 10 times more energy than performing an addition, resulting in low efficiency (*e.g.*, Table 1). On the other hand, massively parallel computing devices such as Graphical Processing Units (GPUs) can provide extremely fast computation, yet they are composed of thousands of parallel cores which need to be synchronized and balanced in terms of workload, which is challenging. This triggered the development of *dedicated* computing platforms, as a neces-

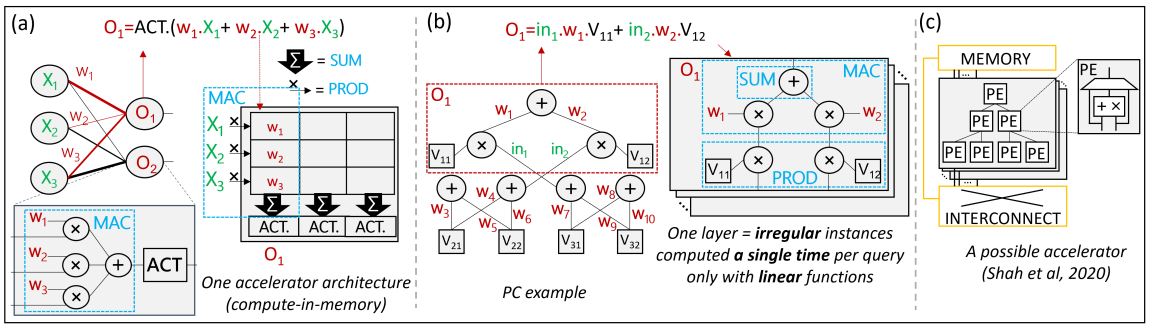

Figure 1: (a) Illustrative example of NN computations and accelerator, (b) Illustrative example of PC computations, (c) Example PC accelerator by Shah et al. (2020).

sary step to *accelerate* the computation of AI workloads with processors optimized for AI computation. Hardware acceleration seeks primarily: 1.) efficient computation, with dedicated hardware blocks, 2.) hardware reusability, *i.e.*, use of Processing Elements (PEs) that can be replicated and interfaced, and 3.) parallelism, to increase the throughput. Following up on the NN example in Fig. 1(a), hardware acceleration is facilitated by 1.) building dedicated MAC operations in hardware, as it is the dominant computation, 2.) reusing MAC units for every neuron, and 3.) computing each neuron in parallel. Accelerators can take the form of vector-matrix multipliers with shared inputs and weights as a matrix.

Distinctively, PCs are computational graphs composed of sums and product operations, without non-linear activations. Non-linearities can only be present at the inputs (leaves). In terms of computation, sum layers are in fact weighted sums and can be represented as a MAC. Product layers provide an extra product operation compared to NNs. Additionally, most queries can be answered with a single pass through the PC, which makes PCs generally compact for inference. A simple PC example (here an SPN), decomposed into computational steps is depicted in Fig. 1(b). Regarding acceleration, PCs have additional particularities: 1.) High computation resolution. As computations consist of successive additions and multiplications of probabilities, computing PCs, even for inference only, require high resolution (PCs are typically computed in the log-domain to avoid underflow). Comparative studies (Shah et al., 2019; Sommer et al., 2020) highlighted that small benchmarks could require 24 – 40 floating bits depending on the error tolerance. 2.) Irregular graph structure. As the exact PC structure vary, PCs should be decomposed into sub-graphs of variable sizes. As analyzed in Shah et al. (2020), computing a PC on a GPU with one thread per decomposed sub-graph introduces inter-thread synchronization overhead and irregular shared memory access, leading to relatively poor performance. Specific graph decomposition can be required to enhance parallelism (Shah et al., 2021). An example accelerator for PCs is depicted in Fig. 1(c) (Shah et al., 2020). The general principle is to build trees of PEs, each performing either an addition or a product. Any graph decomposition can be flexibly considered, yet parallelism cannot be systematically achieved as graphs are irregular.

4. Hardware-cost Evaluation

To evaluate more exhaustively the hardware-friendly nature of each compared model, we divided the general hardware cost into three sub-categories: 1.) the energy per inference, that is directly linked to the number of computations needed for a given model, 2.) the parallelism possibilities, that is related primarily to the graph structure and 3.) the reusability, that is linked to the fact that the same model structure can be used for various applications.

Energy per Inference After training, we can obtain the number of sums and product nodes, the number of variables and the depth of each model. Each operation in the PC can be linked to a computational cost, as detailed in Olascoaga et al. (2019). Table 1 shows the cost for a 64-bit float, corresponding to "double" in software computation. The overall computational cost can be given as follows. We make here the common assumption that the hardware can locally store all intermediate results in registers, while having to fetch all weights and input parameters from memory. These to parameters are represented respectively by `MEM FETCH` local and `MEM FETCH` global. For every operation, the circuit should 1.) fetch the parameters from local and/or global memories; 2.) perform the necessary computation (`ADD` or `MULT`); and 3.) store the result locally. Thus, the energy per inference cost is the sum of all individual costs for a complete bottom-up pass through the PC, which we use as a reference. We also assume that inference is be computed in a linear fashion using 64 bits, as previous work suggest it is sufficient (Sommer et al., 2020; Shah et al., 2019).

Operation	Relative energy cost at ($N_b = 64$)
`ADD` (N_b bit float)	1
`MULT` (N_b bit float)	6
`MEM FETCH` local (N_b bits)	2
`MEM FETCH` global (N_b bits)	10

Table 1: Relative operation costs for 64 bit float operations

Parallelism Inference speed is a critical factor when developing efficient hardware. Note that in the case of PC computation, the irregularity of the graph structure can prevent a systematic parallelism for scalar models. This calculation is then only an approximation that should be refined with a given hardware structure. For example, the accelerator in (Shah et al., 2021) can compute a maximum of 64 parallel threads. As a first approximation, the amount of parallelism achievable by a given PC structure (Shah et al., 2021), denoted Par, can be given by $Par = N_{\text{nodes}}/D$, where N_{nodes} is the number of nodes in the PC and D is the depth of the PC.

Hardware Reusability Another key aspect regarding hardware acceleration is to use predefined PEs. A single PE can be tailored to the exact operation performed by the model, and interconnect of PEs can be optimized. In that regard, the irregular graph structure of PCs make them generally less suitable for hardware reuse. However, recent works around PC training have shown that by constraining the PC structure to contain only specific computation units (*e.g.*, the Einsum layer of Einets, region and partition nodes in RAT-SPNs), competitive performance can be achieved. These units could be realized as PEs for building efficient accelerators, as shown in Section 5.

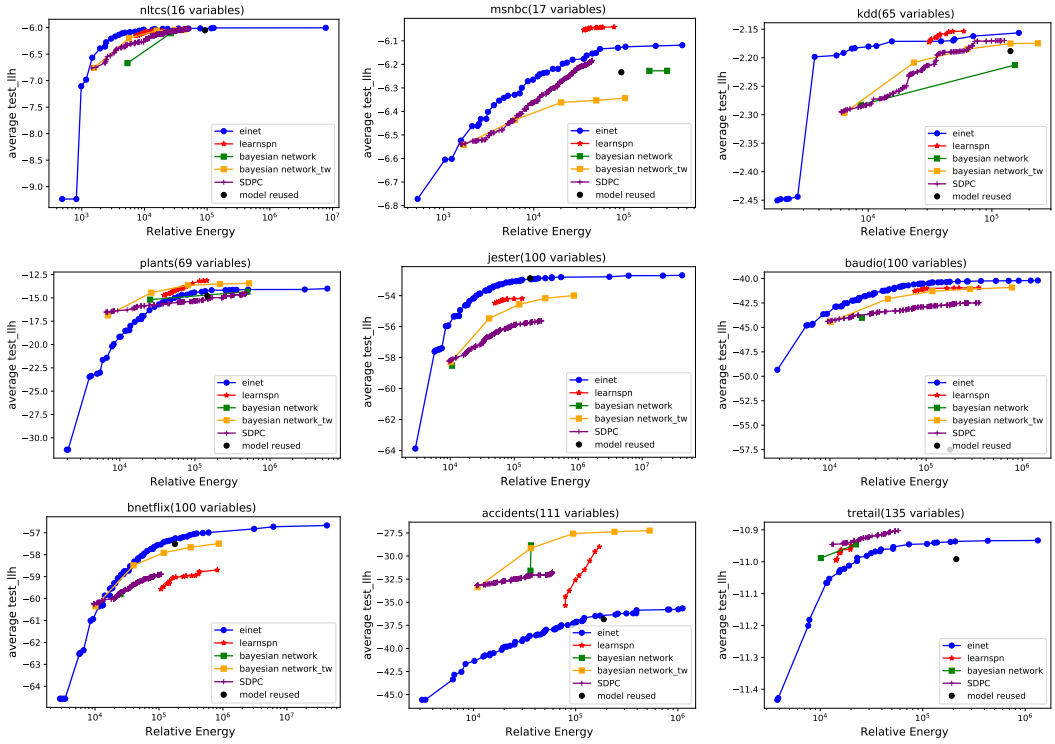

Figure 2: Energy consumption vs test-set likelihood for the small benchmarks.

5. Experiments

We used 20 commonly used benchmark data sets that consist of 16 to 1556 variables and 1600 to 291326 training samples (see Gens and Domingos (2013) for specifics). Only the training set (excluding validation set) is used for the learning of each circuit.

Einets were trained for 50 epochs and the hyperparameter range included all the combinations of depth $[1, 2, 3]$, replica $[1, \ldots, 10]$, number vector elements $[1, \ldots, 10]$, number of sum nodes per inner region $[1, \ldots, 10]$, which means a total 3000 combinations. For learnSPN, we used grid search of 2 hyperparameters: g-test factor $[1, \ldots, 10]$ and number of clustering instances $[50, \ldots, 400]$, resulting in 80 different configurations. For SDPC, we obtained circuits produced after every iteration till 500. For ACs, we trained BNs using default scores and priors of the packages (bnlearn and tw: BIC, Genie: BDeu, ESS 50, graph sparsity priors) (both included as bayesian network). For bounded tree-width BNs (bayesian network_tw) we used tree widths of 1, 3, 5, 7 and 9 similarly as Benjumeda et al. (2019). ACs (BNs), learnSPN, and SDPC took approximately 10-15 minutes to learn a model, training times for Einets varied from minutes to hours, depending on the data set and structure complexity. After obtaining the models, the hardware cost was calculated as detailed in Section 4. The Energy is calculated with the reference value of 1 for an adder. Every node is decomposed in blocks with two inputs to have uniformity in the cost calculation.

Differences in PC Structures First we observed the differences in terms of PC structures. LearnSPN does not put constraints in the PC structure, other than keeping a smooth

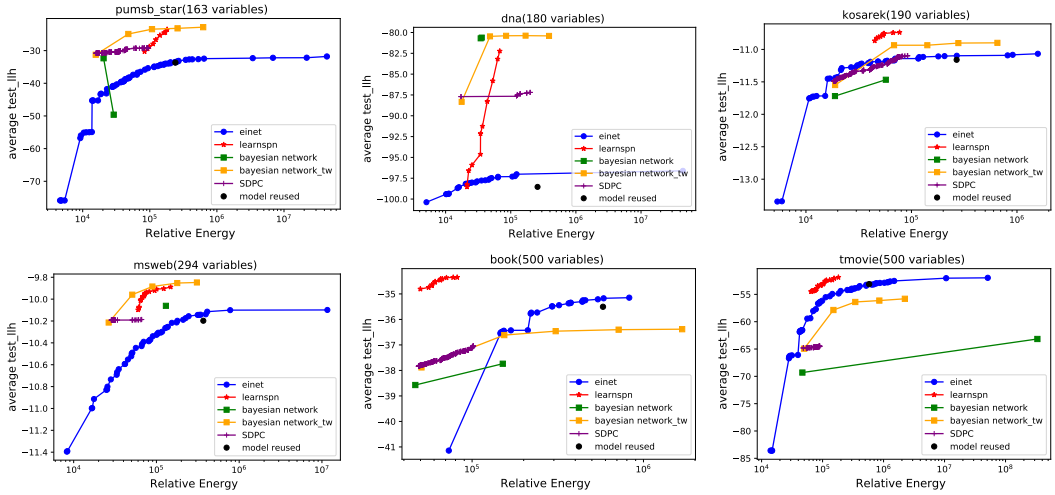

Figure 3: Energy consumption vs test-set likelihood for the medium size benchmarks.

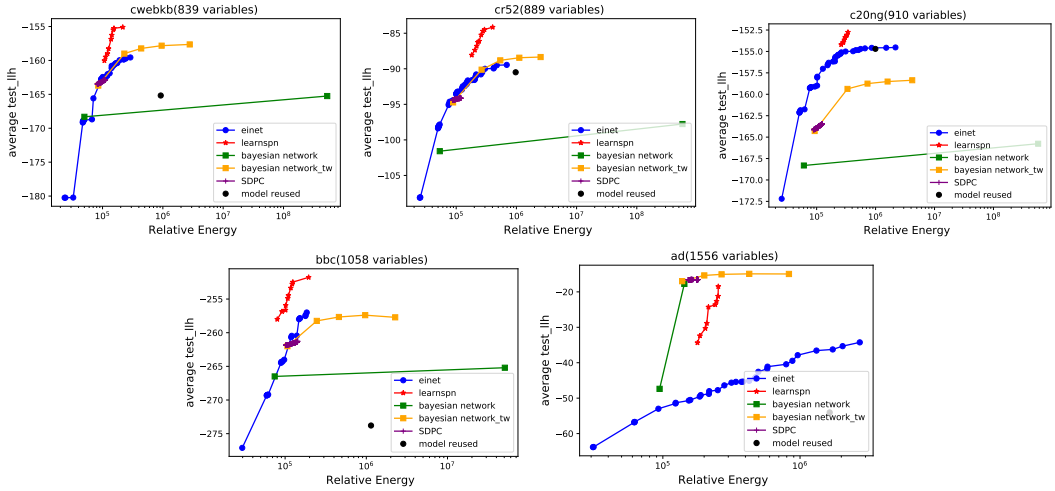

Figure 4: Energy consumption vs. test-set likelihood for the large benchmarks.

and decomposable PC. As such, the tailored graph structure can be more irregular, which can impact the parallelism. The depth of the converted AC is directly proportional to the number of random variables in the network leading to deeper and more shallow networks. Einets are learned over a predefined structure and are overall very regular. SDPCs usually lead to a compact structure even with the additional structural-decomposability constraint.

Model Accuracy versus Energy Consumption Figs. 2, 3, 4 show the obtained test-set likelihoods and energy values for all benchmarks. For each method we only include hyperparameter combinations for which there no higher likelihood with less energy. Overall we can make the following observations: 1.) LearnSPN produces PCs with highest test-set likelihood for 10 out of 20 benchmarks. This can be explained because learnSPN can learn very tailored structures with less structural constraints giving more freedom to fit the input

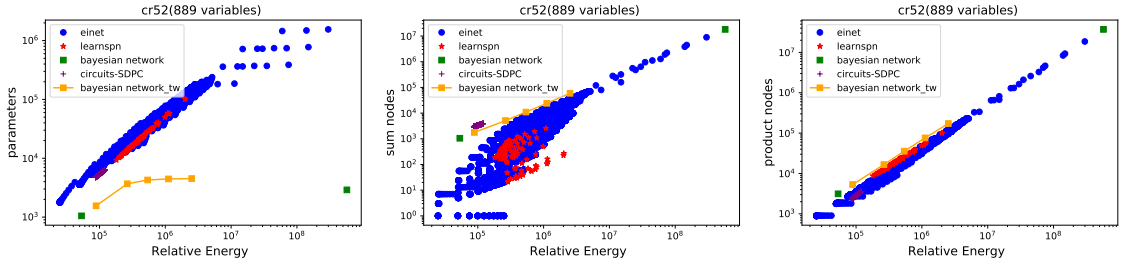

Figure 5: Energy consumption vs. the number of parameters, sums, and products for cr52.

data. 2.) Einets are performing well for most benchmarks, although they also have the most generic and replicated structure. We interpret it as the fact that there is always possible a configuration of hyperparameters that can approximate reasonably well the distribution of the data. 3.) Einets offer a good energy/accuracy trade off. By restricting the energy budget, *e.g.*, to be maximum 10^4 for small benchmarks, 10^5 for bigger ones, Einets perform well even with a simple structure. Also, the representation of Einets allows to obtain an excellent trade off between accuracy and energy for all benchmarks (sweeping hyperparameters). We focused here on smaller Einets for energy constraints, but their structure could be extended for better accuracy. 4.) The additional constraints of SDPCs produces does not significantly impact their accuracy or energy. The fact that this comes with a better tractability makes them also a very good candidate for real time inference applications. 5.) For BNs without a tree-width bound, the energy efficiency varies greatly with the learning method. Some BNs failed to compile to ACs (e.g. jester, baudio, bnetflix and msweb), likely due to larger tree-widths. Different tree-width bounds offer a similar trade-off of fit and efficiency as the previous methods. Likelihoods are excellent for some data sets but not for all.

Energy Consuming Operations Fig. 5 shows the effect of the number of parameters, sum and products on the energy consumption for PCs on the "cr52" benchmark. Although the number of parameters is strongly correlated to the energy consumption, we can observe that, for an equivalent number of parameters, the energy can vary greatly, suggesting that some structures lead to more energy-efficient implementations, even with a constant number of parameters. Fig. 6 shows the energy breakdown of every model on the "book" benchmark. It shows that the main bottleneck is memory fetches, as although storing the intermediate values can be performed using local registers, the weights have to be fetched from the global memory, costing more energy.

Model Accuracy versus Inference Speed/Parallelism Parallelism is primarily intended to improve the inference speed, which is also related to the overall efficiency of the computing platform. In terms of hardware, a more tailored structure may lead to more irregular graphs, hence less parallelism possibilities. Parallelism is correlated to the number of layers in the PC. Einets have 3-5 layers (defined by user) where each layer performs a block of computation that can as well be executed in parallel, learnSPN produced models with 20-50 layers, ACs from Bayesian networks are very deep as new layer is formed for each RV. For instance, by analyzing LearnSPN structure, the maximum parallelism Par is limited

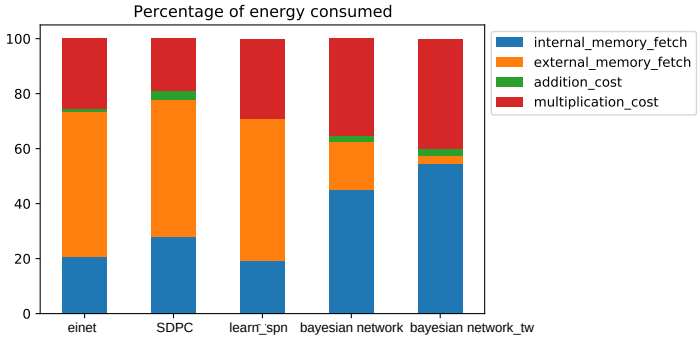

Figure 6: Split share of average energy consumption for the book benchmark.

to be around 100. Einets have the advantage here because the operations are vectorized and the depth is limited, in theory parallelization possibilities up to two or three orders of magnitude higher with a dedicated hardware (for instance, one PE can compute a full Einet layer). A detailed analysis is out of the scope as it is bound to a hardware configuration.

Hardware Reusability Another key feature for hardware implementation is to reuse the same hardware structure for different applications. This can be done by building configurable hardware, or by using the same PC structure for several applications. Among the model tested, Einets have the potential for reusability as per their flexible structure. In our experiment, we picked one Einsum configuration only (depth 2, number of vector elements 9, replica 4, number of sums per inner region 8), and evaluated how this single structure could perform learning for all benchmarks. This performance is marked by black square in Figs. 2, 3, and 4. As seen, this single structure performs variably for all benchmarks, yet showing the potential of Einets to be versatile. A hardware configuration supporting multiple PC (Einet) structures would lead to more consistent accuracy across multiple benchmarks, but more research is still needed to find PC structures usable for a variety of applications.

6. Related Work

Evaluations of PCs have appeared before, *e.g.*, in (Butz et al., 2018) whch compared several SPNs. Here we take here a more hardware-focused perspective, and extend to more recent PC implementations. Efficient hardware implementations targeting specific types of PCs have been recently presented. For instance, the framework in Sommer et al. (2021) enables to compile a SPN either on a CPU or a GPU, using a tool flow based on MLIR and LLVM compilation frameworks. Execution on FPGAs has been studied both for ACs (or Bayesian networks) (Zermani et al., 2015; Dormiani et al., 2005; Geist et al., 2014) and SPNs Molina et al. (2018); Sommer et al. (2018). Custom processors have also been implemented (Shah et al., 2020, 2021). Hardware-aware training methodologies have been previously presented, but they typically target one type of PC (ACs in (Shah et al., 2019), PSDDs in (Olascoaga et al., 2019) and SPNs from learnSPN/SPFlow in (Sommer et al., 2018)).

7. Conclusion

This paper presents a comparative study evaluating the hardware costs of various PC models. Our results show that different structure learning methods can find good and energy efficient models, but constraining to pre-defined structures, *e.g.*, with Einsum networks, allows for achieving better parallelism and reusability properties. Possible extensions of this work include: 1.) generalize towards hardware implementations of ensemble models, 2.) provide efficient hardware implementations for generic and dedicated processors, and 3.) extend towards novel hardware-aware structure learning methods for PCs.

8. Acknowledgements

This work was supported by Academy of Finland through the WHISTLE project (grant 332218); and by grants 315771 (A. Hyttinen), 347279 (M. Trapp).

References

M. Benjumeda, C. Bielza, and P. Larranaga. Learning tractable bayesian networks in the space of elimination orders. *Artificial Intelligence*, 274:66–90, 2019.

C. J. Butz, J. de S. Oliveira, A. E. dos Santos, A. L. Teixeira, P. Poupart, and A. Kalra. An empirical study of methods for SPN learning and inference. In *Proc. PGM*, 2018.

Y. Choi, A. Vergari, and G. Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, 2020.

M. Dang, A. Vergari, and G. Van den Broeck. Strudel: Learning structured-decomposable probabilistic circuits. In *Proc. PGM*, pages 137–148, 2020.

L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE*, 108(4):485–532, 2020.

P. Dormiani, D. Omoto, P. Adharapurapu, and M. D. Ercegovac. A design of online scheme for evaluation of multinomials. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XV*, volume 5910, pages 235 – 246, 2005.

J. Geist, K. Y. Rozier, and J. Schumann. Runtime observer pairs and Bayesian network reasoners on-board FPGAs: Flight-certifiable system health management for embedded systems. In *Runtime Verification*, pages 215–230, 2014.

R. Gens and P. Domingos. Learning the structure of sum-product networks. In *Proc. ICML*, pages 873–880, 2013.

D. Kisa, G. V. den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *Proc. KR*, 2014.

N. H. Mahmood, O. López, O.-S. Park, I. Moerman, K. Mikhaylov, E. Mercier, A. Munari, F. Clazzer, S. Böcker, and H. Bartz (Eds.). White paper on critical and massive machine type communication towards 6G. 6G Research Visions, No. 11, University of Oulu, 2020.

M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communication Surveys and tutorial*, 2018.

N. Mohammadzadeh and R. Safdari. Patient monitoring in mobile health: Opportunities and challenges. *Med Arh*, 68:57–60, 2014.

A. Molina, A. Vergari, N. D. Mauro, S. Natarajan, F. Esposito, and K. Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *Proc. AAAI*, 2018.

L. I. G. Olascoaga, W. Meert, N. Shah, M. Verhelst, and G. Van den Broeck. Towards hardware-aware tractable learning of probabilistic models. In *Proc. NeurIPS*, 2019.

I. París, R. Sánchez-Cauce, and F. J. Díez. Sum-product networks: A survey. *IEEE TPAMI*, 44(7):3821–3839, 2022.

R. Peharz, A. Vergari, K. Stelzner, A. Molina, M. Trapp, X. Shao, K. Kersting, and Z. Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Proc. UAI*, pages 334–344, 2019.

R. Peharz, S. Lang, A. Vergari, K. Stelzner, A. Molina, M. Trapp, G. V. den Broeck, K. Kersting, and Z. Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proc. ICML*, pages 7563–7574, 2020.

T. Pipatsrisawat and A. Darwiche. New compilation languages based on structured decomposability. In *Proc. CNIA*, volume 1, pages 517–522, 01 2008.

H. Poon and P. M. Domingos. Sum-product networks: A new deep architecture. In *Proc. UAI*, pages 337–346, 2011.

M. Scutari. Learning Bayesian networks with the bnlearn r package. *Journal of Statistical Software*, 35:1–22, 2010.

M. Scutari, C. E. Graafland, and J. M. Gutiérrez. Who learns better Bayesian network structures: Constraint-based, score-based or hybrid algorithms? In *Proc. PGM*, 2018.

N. Shah, L. I. G. Olascoaga, W. Meert, and M. Verhelst. Problp: A framework for low-precision probabilistic inference. In *Proc. DAC*, pages 1–6, 2019.

N. Shah, L. I. Galindez Olascoaga, W. Meert, and M. Verhelst. Acceleration of probabilistic reasoning through custom processor architecture. In *Proc. DATE*, 2020.

N. Shah, W. Meert, and M. Verhelst. Graphopt: constrained optimization-based parallelization of irregular graphs, 2021.

N. Shah, L. I. G. Olascoaga, S. Zhao, W. Meert, and M. Verhelst. 9.4 piu: A 248gops/w stream-based processor for irregular probabilistic inference networks using precision-scalable posit arithmetic in 28nm. In *Proc. ISSCC*, 2021.

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 *Proc. ICCD*, 2018.

L. Sommer, L. Weber, M. Kumm, and A. Koch. Comparison of arithmetic number formats for inference in sum-product networks on FPGAs. In *Proc. FCCM*, pages 75–83, 2020.

L. Sommer, M. Halkenhäuser, C. Axenie, and A. Koch. SPNC: Accelerating sum-product network inference on CPUs and GPUs. In *Proc. ASAP*, pages 53–56, 2021.

M. Trapp, R. Peharz, H. Ge, F. Pernkopf, and Z. Ghahramani. Bayesian learning of sum-product networks. In *Proc. NeurIPS*, pages 6344–6355, 2019.

UCLA. Ace compiler, 2015.

S. Zermani, C. Dezan, H. Chenini, J. Diguet, and R. Euler. Fpga implementation of Bayesian network inference for an embedded diagnosis. In *Proc. PHM*, pages 1–10, 2015.