Periasamy, Karthekeyan; Leslin, Jelin; Korsman, Aleksi; Yao, Lingyun; Andraud, Martin

# AutoPC: An Open-Source Framework for Efficient Probabilistic Reasoning on FPGA Hardware

*Published in:*
2024 22nd IEEE Interregional NEWCAS Conference, NEWCAS 2024

*DOI:*
10.1109/NewCAS58973.2024.10666359

Published: 01/01/2024

*Document Version*
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

# AutoPC: an open-source framework for efficient probabilistic reasoning on FPGA hardware

Karthekeyan Periasamy*, Jelin Leslin*, Aleksi Korsman*, Lingyun Yao*, Martin Andraud*†
*Aalto University, Department of Electronics and Nanoengineering, Espoo, Finland
{firstname.lastname}@aalto.fi
†UC Louvain, ICTEAM, Belgium {firstname.lastname}@uclouvain.be

***Abstract*—In the quest for more advanced and energy-efficient edge AI, probabilistic reasoning models can complement or replace deep learning (DL) models, as they are generative, explainable, and trustworthy. However, their hardware implementation and acceleration are still in the early stages compared to DL due to more ad-hoc implementations and challenges translating them into computational steps. This recently evolved with Probabilistic Circuits (PCs), which can be trained with mainstream software and lead to more hardware-efficient inference. Yet, there is currently no single open-source framework dedicated to computing PCs on hardware. In this work, we introduce such a framework called AutoPC, allowing us to (1) compare PCs trained with different PC algorithms to find the most suited, (2) find the optimal resolution required for hardware computation with minimal cost, and (3) automatically generate FPGA hardware for executing PC models with high speed (40-200 GOPS) up to the FPGA capacity. We hope AutoPC serves as a baseline to showcase the possibilities of probabilistic reasoning and broaden the use of PCs.**

***Index Terms*—Probablistic Circuits, HW-SW co-design, Design Automation, High level HW-Simulation**

## I. Introduction

Deep Neural Network (DNN) architectures are standard in contemporary AI, from large language models to efficient implementations on edge devices. On the latter, In-Memory Computing [1]–[3] enables DNNs to be accelerated up to 100-1000x better than in generic computing platforms. Yet, DNNs are always more computationally intensive [4], and harder to be handled on-chip. Also, DNNs can be overconfident in their predictions [5], limited to a single task without retraining [6], and even have been characterized as never truly reliable [7]. Thus, it is crucial to explore alternatives to complement or replace existing DNN solutions.

Probabilistic models can offer such an alternative. They are generative and explainable, encoding the task as a probability distribution [8]. They can be made *tractable*, i.e., performing exact queries in a computationally manageable way. However, typical probabilistic models such as Bayesian Networks or Markov models are difficult to convert into computational steps for efficient hardware acceleration.

These drawbacks are tackled with the emergence of *probabilistic circuits* (PCs) [9]. PCs are seen as *computational* graphs composed of basic arithmetic operations. Some recent PC implementations can be trained with mainstream frameworks such as TensorFlow, and various hardware accelerators have been presented both for FPGAs [10], [11] and ASICs [12]. However, there is still no fully open-source framework available

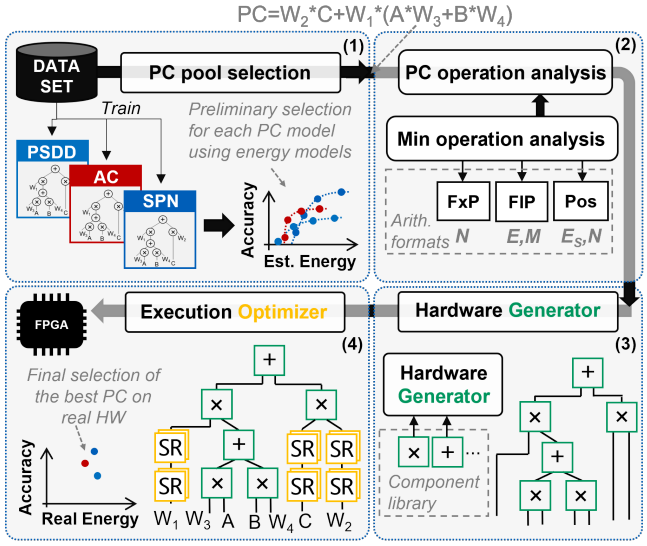


Fig. 1: Overview of the AutoPC tool flow

to train, compare, and execute PCs automatically on FPGA hardware. Existing works typically focus on one type of PC (i.e. one training software), hampering a thorough comparison between different models. Such comparison is needed as the optimal PC model changes for each task [13].

To promote probabilistic reasoning on edge devices and serve as a baseline for future developments in this area, we propose AutoPC [1], an open-source framework focused on the evaluation, comparison, and hardware implementation of PCs. AutoPC builds itself on previous hardware generation tools [11], [14], extending and generalizing the analysis for all PC models. In short, AutoPC trains and extracts a pool of suitable PC models, based on the current task and hardware requirements. Subsequently, it generates and optimizes the hardware implementation on a target FPGA. AutoPC has been designed to be highly flexible and extensible to improve hardware acceleration and include the latest PC models [15].

This paper is organized as follows. Section II introduces PCs. Section III details the proposed AutoPC framework. Section IV illustrates AutoPC in a case study. Section V concludes and elaborates on AutoPC's possible extensions.

[1]The complete code is available at https://github.com/Karthekeyan-aalto/newcas_2024.git

## II. Probablistic Circuits

Probabilistic circuits (PCs) [16] are a unifying framework introduced to harmonize various advancements in designing tractable probabilistic representations (*e.g.*, SPNs [17], PSDDs [18], and CutsetNetworks [19]). PCs subsume many classical probabilistic models such as mixture models and hidden Markov models (HMMs), and exhibit high expressive efficiency (representational power) while maintaining efficient and exact computation of many probabilistic inference queries.

PCs can be seen as a computational graph given by a directed acyclic graph compromised of sum $\oplus$ and product $\otimes$ nodes and with 'simple' distributions at the leaves, typically chosen to be tractable univariate probability distributions or indicator functions in case of discrete data domains. Intuitively, we can understand product nodes to represent independence assumptions between random variables (RVs) and sum nodes to represent mixtures, i.e., replacing the independence assumptions of product nodes with conditional independence assumptions. Each node in a PC is associated with a scope, assigned through a scope function [20], which determines the set of RVs/input dimensions a node defines a probability distribution over. Fig. 2 illustrates a PC along with a Bayesian Network both encoding the same distribution.

*a) Benefits of PCs:* An important property of PCs, which makes them stand out from classical probabilistic graphical models (*e.g.*, Bayesian networks) and modern deep learning, is that PCs are a tractable representation of many probabilistic queries and guarantee that computations can be performed exactly and efficiently. Given a class of queries/inference tasks $\mathcal{Q}$ (*e.g.*, computing marginals, or maximum-a-posterior queries) and a family of models $\mathcal{M}$, we say that $\mathcal{M}$ is a tractable representation of a class of queries $\mathcal{Q}$ if we can answer every $q \in \mathcal{Q}$ for every model $m \in \mathcal{M}$ in polynomial time measured in the model complexity $\mathcal{O}(\text{poly}(|m|))$. In the case of PCs, tractability is directly reflected in their structural properties. For details, we refer to [16].

*b) Learning PCs:* Learning a PC typically involves two steps: (i) learning the structure and (ii) learning the parameters. The structure can be learned from data (e.g., [20]–[22]) or chosen to be random but sufficiently large (e.g., [15]). Parameters are typically learned by employing expectation maximization (EM) or gradient descent (GD) to fit the model to the data.

*c) Application of PCs:* PCs have proven to be competitive in speech recognition [23] or activity/action recognition from images [24], [25]. They can be more robust and compact than convolutional NNs for speaker identification tasks, and also competitive for more complex speech recognition and verification tasks [23]. They have also been used for semantic mapping with robots [26]. We refer to [27] for a larger review of applications. PCs can advantageously replace existing DNN models. In [28], a variational autoencoder has been replaced by a PC, showing faster learning and reducing the inference cost significantly. This highlights the compactness of PCs.

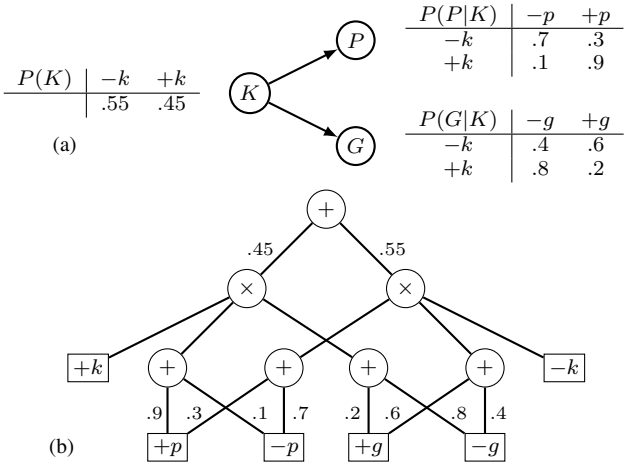


Fig. 2: (a) A Bayesian network encoding a distribution with three variables and (b) An equivalent probabilistic circuit.

## III. Proposed Framework

Fig.1 shows the AutoPC workflow comprising 4 main steps. (1) AutoPC starts by training various PC models to select a pool of suitable configurations, comparing the accuracy and estimated inference energy with a behavioral model. (2) Based on that pool, it converts each model to a common representation and analyses the optimal configuration to execute the model for various arithmetic formats and resolutions. (3) Following this analysis, it generates a digital hardware implementation of the best configuration of each PC model. (4) Each implementation is subsequently optimized to complete the selection of the best PC model on FPGA.

(1) *PC pool selection:* As each PC model has its own structure and parameter learning method, the optimal PC model differs for each application. AutoPC starts by selecting a pool of suitable PC configuration using the methodology described in [13]. The methodology is illustrated here with two PC models, ACs compiled from a Bayesian Network through ACE [29], and SPNs trained from SPFlow [30], but it can be extended to other training frameworks. AutoPC trains these models on a wide range of hyperparameters and picks the best model based on a pareto optimal plot of the accuracy (log likelihood) and the energy (extracted from energy models).

(2) *PC operation analysis:* PC models are first translated in a common equation format, following the idea of a network polynomial [31]. This PC equation is parsed into elementary blocks, keeping track of their execution order for hardware generation, using software stacks. Stacks work in the "Last in First Out" order, where push and pop mechanisms add and remove items from the stack. Each block refers to a pre-defined hardware component defined in a *Component library*. Parsing can be extended for more complex computational blocks. After decomposition, AutoPC analyzes the PC operation with multiple arithmetic formats and resolutions. This is to

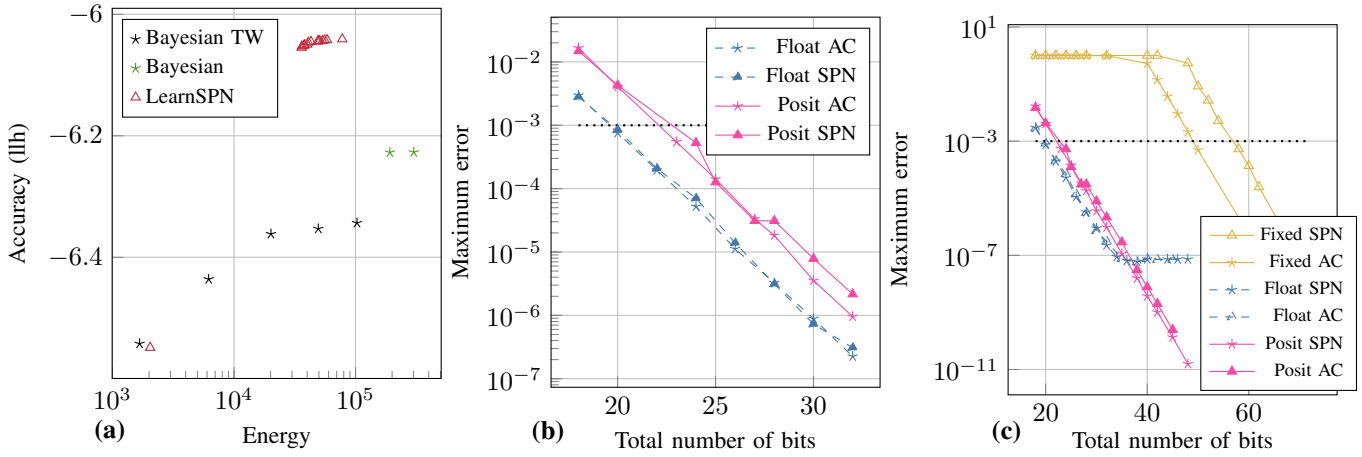


Fig. 3: (a) Illustration of the PC model selection comparing implementations of various PC models (b) Max error comparison of SPN versus AC on the same benchmark; (c) Max error comparison for three arithmetic formats for SPNs and ACs

1.) avoid underflow, as the PC processes small numbers, 2.) ensure that the representation error of any value is smaller than a pre-defined error setting and 3.) optimize the hardware efficiency. Existing methodologies perform either a theoretical bound analysis of the maximum computation error for a given graph structure [14], iterating over several resolutions, or an exhaustive search based on the data [11], which can be time-consuming. AutoPC takes a middle approach: start with a theoretical resolution analysis for fast exploration and fine-tune the resolution with a data-driven analysis for accuracy. Three formats are currently considered: fixed point (FxP), floating point (FlP), and Posit (Pos). To avoid underflow, the analysis determines the smallest positive non-zero probability at the top node, $MV$, found by replacing all sum nodes by *min* operators and traversing the graph bottom-up, with all indicators set to '1' [14]. The analysis uses relative error tolerance as a constraint, $\epsilon = |\tilde{x} - x|/|x|$, where $x$ is the real value and $\tilde{x}$ is the represented value. $\epsilon$ is then compared with the quantization error of each format. The error estimation is calculated for marginal queries. The resolution requirements for the three arithmetic formats can be determined with $MV$ and $\epsilon$ (see the details in section IV). In practice, the minimum computed value $MV$ is by definition rarely encountered. Hence, we use a data-driven analysis to fine-tune the final resolution, by computing the maximum relative error observed on the test data. This ensures that the final error fits within the pre-defined error bound, while not overestimating the resolution.

*3 and 4 Hardware generation and optimization operation analysis:* Upon completion of the previous steps, AutoPC generates a high-level hardware of each PC model and is called a Python PC tree. It contains the description of the hardware blocks in VHDL. FloPcoCo library [32] supports hardware generation of float and posit number system representation. Then, the generated hardware is optimized by AutoPC. In general, AutoPC performs two types of optimisation 1) Forward path delay optimisation and 2) Automatic pipelining of the Python PC tree. First, AutoPC uses tree traversal techniques such as breadth-first traversal and depth-first traversal techniques across the whole Python PC tree to extract necessary information on the total number of shift registers and their associated delays. Then, AutoPC optimizes the throughput of the hardware tree by performing automatic pipelining of all the levels of the hardware tree. Combining the two optimization steps helps maximize the throughput of the hardware compared to the latest FPGA implementation of PCs and is discussed in the section IV.

TABLE I: The effect of AutoPC analysis on exponent bits and es bits of Floating and Posit number system

| Configuration | Number of bits | Observed Error on test set |
|---|---|---|
| MSNBC,SPN, Float,$\epsilon = 10^{-3}$ | $22(E=7, M=15)$ | 0.0002 |
| | $23(E=8, M=15)$ | 0.0002 |
| | $24(E=9, M=15)$ | 0.0002 |
| MSNBC,AC, Float,$\epsilon = 10^{-3}$ | $21(E=6)$ | $UF$ |
| | $22(E=7)$ | 0.0002 |
| | $23(E=8)$ | 0.0001 |
| | $24(E=9)$ | 0.0001 |

## IV. Results

In this section, AutoPC is illustrated with a typical density estimation benchmark MSNBC, part of commonly used benchmarks in the PC literature [33]. It compares two PC models (ACs and SPNs) and three arithmetic formats (fixed-point, floating point and posit).

*1 PC pool selection:* SPNs are trained from the SPFlow library with default settings [30]. For AC, a Bayesian Network (BN) is trained using default settings in the bnlearn package [34] and the model is converted to an AC with ACE [29]. To increase the variety of models to be compared, BNs are trained with either regular settings or with additional constraints on the treewidth (tw). Figure 3(a) plots the comparison results in terms of log-likelihood (i.e. how well the model represents the underlying distribution, the higher the better) and energy,

| Work | Model | Error Tolerance | PC nodes Sums | PC nodes Prod. | Format | Resolution E | Resolution M | Hardware occupation FF | Hardware occupation LUT | Hardware occupation DSP | Freq. (MHz) | Power (W) | Computing Metrics GOPS | Computing Metrics GOPS/W |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OURS | SPN | $10^{-3}$ | 35 | 103 | Float | 7 | 15 | 34372 | 23145 | 103 | 277 | 1.53 | 38.2 | 24.9 |
| OURS | AC | $10^{-3}$ | 391 | 1220 | Float | 7 | 15 | 385402 | 224765 | 1220 | 153 | 10.290 | 246.4 | 23.9 |
| OURS | SPN | $10^{-6}$ | 35 | 103 | Float | 7 | 25 | 72786 | 43902 | 206 | 277 | 2.841 | 38.2 | 13.4 |
| OURS | AC | $10^{-6}$ | 391 | 1220 | Float | 7 | 25 | 742926 | 467913 | 2440 | 161 | 25.856 | 259.3 | 10.0 |
| [11] | SPN | $10^{-6}$ | 30 | 165 | Float | 8 | 26 | 236007 | 223704 | 990 | 200 | NA | 39 [a] | NA |
| [12][b] | SPN | $10^{-3}$ | NA | NA | Posit | NA | NA | NA | NA | NA | 278 | 0.23 | 33.7 | 248 |

[a]Operations taken without considering memory transfers. A direct comparison is challenging as it has not been possible to use the exact same PC structure in our experiments, since LearnSPN is stochastic and hence does not learn the exact same structure twice.

[b]Results are given as the average of many SPN benchmarks with 32 bit posit configuration, with an operating frequency of 116 MHz

TABLE II: FPGA hardware generation results for different datasets and PC models. Results presented for Posit format with AutoPC's suggested number of bits and various hardware synthesis settings

estimated from behavioral models [13]. SPNs perform slightly better for this task, with a lower energy than ACs.

*(2) PC operation analysis:* AutoPC first performs the *minAnalysis* by calculating the minimum value $MV$, with a fixed error tolerance (here $10^{-3}$). Based on this it suggests a given number of bits, for hardware PC generation. To assess the validity of the analysis, the test set of each benchmark is stimulated using the generated hardware PC, and the respective maximum relative error is calculated. On MSNBC, Fig.3(c) shows the observed error for both SPNs and ACs while sweeping the resolution of different formats (the fraction bits in fixed point, the mantissa bits in float, and the total number of bits with fixed $es$ in posit). Specifically, from Fig.3(b) it is clear that float leads to a better tradeoff achieving the target error for 2 to 3 bits less than Posit. After the *minAnalysis* is performed, the *resAnalysis* will generate several hardware PCs with neighboring resolutions so that the user can fine-tune the final resolution.

*a) Example with float analysis:* It can be seen from Fig.3(b), the *minAnalysis* exhibit ideal results for MSNBC SPN and AC. But, for MSNBC AC the minimum number of mantissa bits $M$ and exponent bits $E$ suggested by AutoPC records an underflow ($UF$). Table I illustrates changing exponent bits $E$ from value 6 to value 7 reduces the error. This shows the optimal number of $E$ bits i.e., 7, can be found by performing the *resAnalysis*. The starting point of the *resAnalysis* is the suggested number of bits ($E = 6, M = 15, N = 21$). Then, performing the *resAnalysis* on only a few neighboring combinations around the suggested number of bits (in this case 15 combinations, with $E = \{6, 7, 8\}$ and $M = \{13, 14, 15, 16, 17\}$) an optimal point of 22 bits $E = 7, M = 15$ can be found.

*(3) and (4) Hardware generation and optimization operation analysis:* In this experiment, AutoPC uses open-source Flopoco Adders and Multipliers with pipeline depths of 22 and 4 for the target error tolerance of $10^{-3}$ and Flopoco Adders and Multipliers with pipeline depths of 24 and 9 for the target error tolerance $10^{-6}$ respectively. There are two FPGA targets 1) Xilinx Virtex 7 with part name xc7vx690tffg1761-3 (for MSNBC AC $(10^{-3})$ and SPN $(10^{-3})$,$(10^{-6})$ in Table II), and 2) Xilinx Ulstrascale+ board, with part name xcvu9p-flga2104-3-e (for MSNBC AC $(10^{-6})$ in Table II) which has enough hardware available for implementation.

Table II summarises the post-implementation results of the optimal PC models (best AC and best SPN). The hardware description generated by AutoPC, given sufficient hardware resources such as the number of lookup tables (LUTs), flip-flops (FFs), input-output pins (IOs), and digital signal processing units (DSPs) on the board. Additionally, the design complexity affects the operating frequency and the power consumption.

Table II also shows the computing performance of the best PC model for each dataset, according to power consumption and operational frequency. The number of operations $N_{op}$ is the addition of sum nodes and product nodes in Table II. The computation speed depends on the number of operations and tends to increase with the number of nodes as the computation is pipelined. As Auto PC is able to handle more complex models (1600+ nodes compared to 500 in [35]), the computing performance can be increased while maintaining a similar efficiency, achieving up to 246 GOPS. When both speed and power are considered, the overall computing efficiency is 24.9 GOPS/W, which is competitive with existing FPGA accelerators and around 10x slower than dedicated processors.

With its hardware-software co-integration approach, AutoPC is able to select models with the highest accuracy and lowest computation resolution to accelerate them on FPGA hardware.

## CONCLUSION

This paper presents AutoPC, an open-source framework for the automatic evaluation and hardware generation of PCs. AutoPC predicts floating point number system as the suitable number system and generates optimized high-speed hardware (40-200 GOPS) for FPGA implementation of PCs such as ACs and SPNs. This first version has been designed to be a backbone for various extension possibilities, such as supporting more PC models and query types and allowing more functionalities to increase the energy efficiency of the accelerators (finding computation patterns, enabling quantization, pruning, or approximate computing for instance)

## ACKNOWLEDGMENTS

This work is funded by Academy of Finland project WHISTLE (grant 332218).

## REFERENCES


[1] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L. Chen, B. Zhang, and P. Deaville, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, Summer 2019.

[2] B. Murmann, "Mixed-signal computing for deep neural network inference," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 3–13, 2021.

[3] J.-s. Seo, J. Saikia, J. Meng, W. He, H.-s. Suh, Anupreetham, Y. Liao, A. Hasssan, and I. Yeo, "Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs," *IEEE Solid-State Circuits Magazine*, vol. 14, no. 3, pp. 65–79, 2022.

[4] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[5] M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 41–50.

[6] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[7] G. Marcus, "The next decade in AI: Four steps towards robust artificial intelligence," *arXiv preprint arXiv:2002.06177*, 2020.

[8] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[9] Y. Choi, A. Vergari, and G. Van den Broeck, "Probabilistic circuits: A unifying framework for tractable probabilistic models," oct 2020. [Online]. Available: http://starai.cs.ucla.edu/papers/ProbCirc20.pdf

[10] 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 *AAAI*. AAAI Press, 2018, pp. 3828–3835.

[11] 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)*, 2018, pp. 350–357.

[12] 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 *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 150–152.

[13] J. Leslin, A. Hyttinen, K. Periasamy, L. Yao, M. Trapp, and M. Andraud, "A hardware perspective to evaluating probabilistic circuits," in *Proceedings of The 11th International Conference on Probabilistic Graphical Models*, ser. Proceedings of Machine Learning Research, vol. 186. PMLR, 2022, pp. 349–360.

[14] N. Shah, L. I. G. Olascoaga, W. Meert, and M. Verhelst, "Problp: A framework for low-precision probabilistic inference," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[15] 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," 2020.

[16] Y. Choi, A. Vergari, and G. Van den Broeck, "Probabilistic circuits: A unifying framework for tractable probabilistic models," UCLA, Tech. Rep., 2020.

[17] H. Poon and P. M. Domingos, "Sum-product networks: A new deep architecture," in *UAI*. AUAI Press, 2011, pp. 337–346.

[18] D. Kisa, G. V. den Broeck, A. Choi, and A. Darwiche, "Probabilistic sentential decision diagrams," in *Proc. KR*, 2014.

[19] T. Rahman, P. Kothalkar, and V. Gogate, "Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II 14*. Springer, 2014, pp. 630–645.

[20] M. Trapp, R. Peharz, H. Ge, F. Pernkopf, and Z. Ghahramani, "Bayesian learning of sum-product networks," in *NeurIPS*, 2019, pp. 6344–6355.

[21] R. Gens and P. Domingos, "Learning the structure of sum-product networks," in *International Conference on Machine Learning*, 2013, pp. 873–880.

[22] 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," 2020.

[23] A. Nicolson and K. K. Paliwal, "Sum-product networks for robust automatic speaker identification," in *Proc. Interspeech 2020*, 2020, pp. 1516–1520.

[24] M. R. Amer and S. Todorovic, "Sum product networks for activity recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 4, pp. 800–813, 2016.

[25] J. Wang and G. Wang, "Hierarchical spatial sum–product networks for action recognition in still images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 1, pp. 90–100, 2018.

[26] K. Zheng and A. Pronobis, "From pixels to buildings: End-to-end probabilistic deep networks for large-scale semantic mapping," 2019.

[27] I. París, R. Sánchez-Cauce, and F. J. Díez, "Sum-product networks: A survey," *arXiv preprint arXiv:2004.01167*, 2020.

[28] K. Stelzner, R. Peharz, and K. Kersting, "Faster attend-infer-repeat with tractable probabilistic models," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 5966–5975.

[29] UCLA. Ace compiler. [Online]. Available: http://reasoning.cs.ucla.edu/ace/

[30] A. Molina, A. Vergari, K. Stelzner, R. Peharz, P. Subramani, N. Di Mauro, P. Poupart, and K. Kersting, "Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks," *arXiv preprint arXiv:1901.03704*, 2019.

[31] M. Chavira and A. Darwiche, "On probabilistic inference by weighted model counting," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 772–799, 2008.

[32] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

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

[34] M. Scutari, "Learning bayesian networks with the bnlearn r package," *arXiv preprint arXiv:0908.3817*, 2009.

[35] L. Sommer, L. Weber, M. Kumm, and A. Koch, "Comparison of arithmetic number formats for inference in sum-product networks on fpgas," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 75–83.