Scaling Up Probabilistic Circuits via Monarch Matrices

Honghua Zhang*[1] Benjie Wang*[1] Meihua Dang*[2]
Nanyun Peng[1] Stefano Ermon[2] Guy Van den Broeck[1]

[1]University of California, Los Angeles [2]Stanford University

Abstract

Probabilistic circuits (PCs) are a tractable representation of probability distributions allowing for exact and efficient computation of likelihoods and marginals. Recent advancements have focused on improving the scalability and expressiveness of PCs by leveraging their sparse properties or tensorized operations. However, no existing method fully exploits both aspects simultaneously. In this paper, we propose a novel structured sparse parameterization for the sum blocks in PCs. By replacing dense matrices with sparse Monarch matrices, we significantly reduce memory and computation costs, enabling scalable training of PCs. From a theory perspective, our method arises naturally from circuit multiplication; from a practical perspective, the structured sparsity of Monarch matrices facilitates efficient tensorization and parallelization. Experimental results demonstrate that our approach not only achieves state-of-the-art performance among tractable models on challenging tasks, including density estimation on ImageNet32 and language model distillation, but also demonstrates superior computational efficiency, achieving the same performance with less computation as measured by the number of floating-point operations (FLOPs) during training.

1 Introduction

Probabilistic circuits (PCs) are a unifying representation of tractable probability distributions through computation graphs (Choi, Vergari, and Van den Broeck 2020; Darwiche 2003). The key property that separates PCs from other deep generative models such as flow-based models (Papamakarios et al. 2021) and VAEs (Kingma and Welling 2013) is their *tractability*. This property enables PCs to compute various queries, including marginal probabilities, exactly and efficiently (Vergari et al. 2021). The tractability of PCs have been exploited in a number of domains, including fair and explainable machine learning (Choi, Dang, and Van den Broeck 2021; Wang, Khosravi, and Van den Broeck 2021), causal inference (Zečević et al. 2021; Wang, Wicker, and Kwiatkowska 2022; Wang and Kwiatkowska 2023), and neuro-symbolic AI (Ahmed et al. 2022; Maene, Derkinderen, and Martires 2024).

Recent advancements in PC learning (Liu et al. 2023; Gala et al. 2024) and efficient implementations (Peharz et al. 2020; Dang et al. 2021; Liu, Ahmed, and Van den Broeck 2024) have significantly enhanced the expressiveness and scalability of PCs. However, to further boost the performance of PCs, simply scaling up the model size is insufficient; we need to better utilize the available capacity. To this end, Dang, Liu, and Van den Broeck (2022) leverage the inherent sparsity property of PCs and iteratively learn their sparse structures through pruning and growing. However, the sparse structures learned are arbitrary, making them difficult to tensorize and parallelize effectively.

In this paper, we focus on leveraging structured sparse parameterizations for the sum blocks in PCs, which represent linear maps. To the best of our knowledge, all previous circuit architectures utilizing tensorized operations have relied on dense matrices to parameterize these linear maps (Peharz et al. 2020). Inspired by recent advances in low-rank approximations for transformers (Hu et al. 2022), we propose a more efficient parameterization for the sum blocks. We begin by illustrating how our parameterization naturally arises through *circuit multiplication*. Previous analysis (Vergari et al. 2021) showed that multiplying two (compatible) circuits could result in a quadratic increase in size in the worst case. However, we observe that the linear maps in the sum blocks generated by multiplication are not dense but rather the *tensor product* of the linear maps from the original blocks, which can be implemented more efficiently. Further, by explicitly materializing this map as interleaving sums and permutations, we identify a connection between these product circuits and *Monarch* matrices (Dao et al. 2022), a recently introduced class of structured matrices. Building on this insight, we propose replacing the dense linear maps in PCs with Monarch layers.

In the experiments section, we demonstrate that replacing dense matrices with Monarch sparse matrices achieves state-of-the-art density estimation results on image datasets, including ImageNet32, as well as on a language model distillation task. Furthermore, we show that, compared to circuits with dense layers, the ones with Monarch layers yield better scaling curves: they can achieve the same performance with fewer floating-point operations (FLOPs) in training.

2 Probabilistic Circuits and its Tensorization

Notation We use uppercase to denote variables (e.g. X) and lowercase to denote values of variables (e.g. x). We use boldface to denote sets of variables/values (e.g. $\boldsymbol{X}, \boldsymbol{x}$).

*Equal contributions.

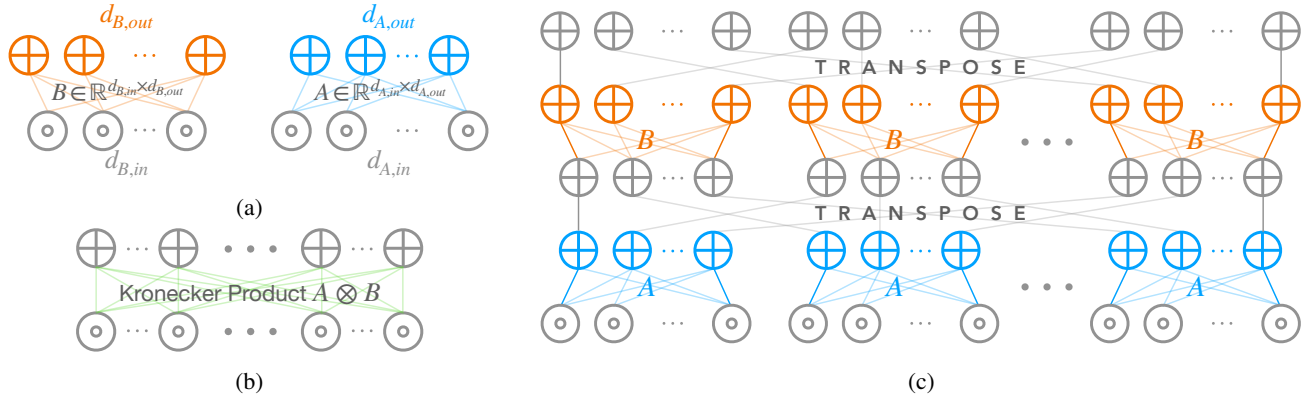

Figure 1: **Probabilistic circuits architecture illustration with Monarch matrices.** (a) Two sum blocks in PCs with weight matrices A, B of arbitrary dimensions. (b) A constructed sum block with weight matrix as the Kronecker product $A \otimes B$, representing the circuit product of two sum blocks. (c) Efficient circuit representation for the linear transformation $A \otimes B$.

Definition 2.1 (Probabilistic Circuit)**.** A PC $\mathcal{C} = (\mathcal{G}, \boldsymbol{\theta})$ represents a joint probability distribution over random variables $\boldsymbol{X}$ through a directed acyclic (computation) graph (DAG) $\mathcal{G}$ parameterized by $\boldsymbol{\theta}$. Specifically, the DAG $\mathcal{G}$ consists of three types of nodes – *sum*, *product*, and *leaf* nodes. Each leaf node n is associated with a non-negative function $f_n(X_n)$ over some variable X_n, called its *scope* $\boldsymbol{X}_n := \{X_n\}$. The scope of any sum or product node n is defined to be $\boldsymbol{X}_n := \bigcup_{c \in \mathrm{ch}(n)} \boldsymbol{X}_c$, where $\mathrm{ch}(n)$ denotes the children of n in $\mathcal{G}$. Each node n represents a probability distribution p_n over its scope $\boldsymbol{X}_n$, defined recursively by:

$$p_n(\boldsymbol{X}_n) = \begin{cases} f_n(X_n) & \text{if } n \text{ is a leaf node} \\ \prod_{c \in \mathrm{ch}(n)} p_c(\boldsymbol{X}_c) & \text{if } n \text{ is a product node} \\ \sum_{c \in \mathrm{ch}(n)} \theta_{c|n} \cdot p_c(\boldsymbol{X}_c) & \text{if } n \text{ is a sum node} \end{cases}$$

where for each leaf node, function $f_n(X_n)$ represents a normalized univariate probability mass/density function (e.g. Categorical, Gaussian); and for every sum node n, $\theta_{c|n}$ is a non-negative weight associated with the edge (n, c) in the DAG, $\sum_{c \in \mathrm{ch}(n)} \theta_{c|n} = 1$ then the PC computes a normalized joint probability mass/density function. The function represented by a PC, denoted $p_{\mathcal{C}}(\boldsymbol{X})$, is the function represented by its root node; and the size of a PC, denoted $|\mathcal{C}|$, is the number of edges in its graph.

It is immediate from the definition that one can evaluate a PC's function with a single traversal through its computation graph. The distinguishing feature of PCs compared to other computation graphs such as neural networks is that one can also efficiently compute *marginals* under the following restrictions on the node scopes:

Definition 2.2 (Smoothness and Decomposability)**.** A sum node is *smooth* if all of its children have the same scope. A product node is *decomposable* if its children have disjoint scope. A PC is smooth (resp. decomposable) if all of its sum (resp. product) nodes are smooth (resp. decomposable).

In practice, probabilistic circuit graphs are typically designed in a tensorized manner, in which sets of nodes of the same type (sum, product, leaf) and with the same scope are grouped together as a *block*; the computation graph is then specified through connections between the blocks (Peharz et al. 2020; Liu, Ahmed, and Van den Broeck 2024; Loconte et al. 2024). We write $\boldsymbol{n}$ to denote a node block and $|\boldsymbol{n}|$ for the number of nodes in the block.

Definition 2.3 (Sum Block)**.** A sum block $\boldsymbol{n}$ has a set of child blocks $\{\boldsymbol{c}^{(i)}\}_{i=1}^{m}$, such that each sum node in the block is connected to every node in each of the child blocks. We can write $W \in \mathbb{R}_{\geq 0}^{|\boldsymbol{n}| \times (\sum_{i=1}^{m} |\boldsymbol{c}^{(i)}|)}$ for the weight matrix.

Definition 2.4 (Product Block)**.** A product block $\boldsymbol{n}$ has a set of child blocks $\{\boldsymbol{c}^{(i)}\}_{i=1}^{m}$. We define two types of product node block with different connectivity:

- Hadamard $\bigodot$: If $|\boldsymbol{c}^{(i)}| = |\boldsymbol{n}|$ for all $i = 1, \ldots, n$, then we define a Hadamard product block where $\boldsymbol{n} = \bigodot_{i=1}^{m} \boldsymbol{c}^{(i)}$.
- Kronecker $\bigotimes$: If $|\boldsymbol{n}| = \prod_{i=1}^{m} |\boldsymbol{c}^{(i)}|$, then we can define a Kronecker product node block where $\boldsymbol{n} = \bigotimes_{i=1}^{m} \boldsymbol{c}^{(i)}$.

Sum blocks represent parameterized linear maps, while product blocks represent fixed *multi*linear maps. Typically, for smooth and decomposable PCs, one builds the circuit by alternating between sum and product blocks (i.e., children of sum blocks are product blocks, children of product blocks are sum/leaf blocks). In this paper, we focus on the parameterization of the sum blocks, which is independent from the choice of Hadamard or Kronecker product blocks (we use Hadamard product blocks for our experiments).

3 From Circuit Multiplication to Generalized Monarch Matrices

We start by considering the task of circuit multiplication: given two circuits $\mathcal{A}$ and $\mathcal{B}$, the goal is to construct a tractable circuit $\mathcal{C}$ such that $p_{\mathcal{C}}(\boldsymbol{x}) \propto p_{\mathcal{A}}(\boldsymbol{x}) \cdot p_{\mathcal{B}}(\boldsymbol{x})$. It has been shown that if $\mathcal{A}$ and $\mathcal{B}$ are structured-decomposable with respect to the same vtree, i.e., the product nodes in $\mathcal{A}$ and $\mathcal{B}$ always factor the same way, then $\mathcal{C}$ can be constructed in polynomial-time (Shen, Choi, and Darwiche 2016; Vergari et al. 2021).

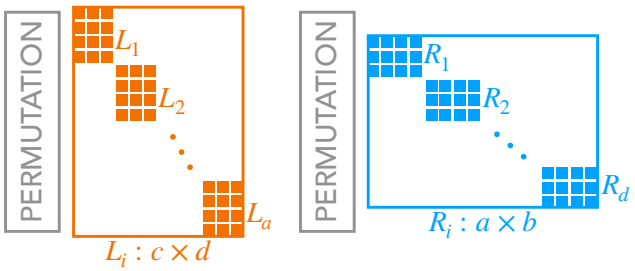

Figure 2: Generalized monarch matrices.

To multiply two structured-decomposable PCs, we proceed bottom-up and locally multiply sum/product blocks with respect to the same scope. Here we focus on the operation of multiplying two sum blocks. Given two sum blocks with weight matrices $A \in \mathbb{R}_{\geq 0}^{d_{A,\text{out}} \times d_{A,\text{in}}}$ and $B \in \mathbb{R}_{\geq 0}^{d_{B,\text{out}} \times d_{B,\text{in}}}$ (Figure 1a), their product can be represented as a sum block with its weight matrix given by the Kronecker product $A \otimes B$ (Vergari et al. 2021). Figure 1b shows a circuit materialization of $A \otimes B$, consisting of $O(d_{A,\text{out}} d_{B,\text{out}} d_{A,\text{in}} d_{B,\text{in}})$ edges. We make the key observation that the linear transformation given by $A \otimes B$ can actually be executed in a significantly more efficient way: let $\boldsymbol{x} \in \mathbb{R}_{\geq 0}^{d_{A,\text{in}} d_{B,\text{in}}}$ be an input vector, we compute the linear transformation $(A \otimes B)\boldsymbol{x}$ as

$$
\begin{aligned}
&((A \otimes B)\boldsymbol{x})_{ij} \\
&= \sum\nolimits_{kl} (A \otimes B)_{ij,kl} \boldsymbol{x}_{kl} = \sum\nolimits_{k,l} A_{ik} B_{jl} \boldsymbol{x}_{kl} \\
&= \sum\nolimits_{l} B_{jl} \sum\nolimits_{k} A_{ik} \boldsymbol{x}_{kl} = \sum\nolimits_{l} B_{jl} (A\boldsymbol{x})_{il} \\
&= \sum\nolimits_{l} B_{jl} (A\boldsymbol{x})^T_{li} = (B(A\boldsymbol{x})^T)_{ji} = (B(A\boldsymbol{x})^T)^T_{ij};
\end{aligned}
$$

hence, we have

$$
(A \otimes B)\boldsymbol{x} = (B(A\boldsymbol{x})^T)^T. \tag{1}
$$

Figure 1c shows the materialization of Equation 1 as a circuit and the total number of edges is bounded by $O(d_{A,\text{out}} d_{A,\text{in}} d_{B,\text{in}} + d_{B,\text{out}} d_{A,\text{out}} d_{B,\text{in}})$. For a rough comparison against the naive construction of $A \otimes B$, if $A, B \in \mathbb{R}^{m \times m}$, then the naive circuit construction contains $O(m^4)$ edges while the construction based on Equation 1 contains only $O(m^3)$ edges.

Despite the fact that the circuit shown in Figure 1c is obtained by multiplying two sum blocks, its structure immediately gives rise to a sparse representation for some linear transformation $\mathcal{M}$ from $\mathbb{R}_{\geq 0}^{d_{A,\text{in}} d_{B,\text{in}}}$ to $\mathbb{R}_{\geq 0}^{d_{A,\text{out}} d_{B,\text{out}}}$ and more importantly, the A and B blocks do not need to have the same parameters for $\mathcal{M}$ to be valid. For clarity, we represent the structure of each layer of circuit 1c as a matrix and show them in Figure 2. From here, we characterize a family of linear transformations as follows:

Definition 3.1 (Generalized Monarch Matrices)**.** Given $a, b, c, d \in \mathbb{N}$, we define a family of linear transformations $\mathcal{M} : \mathbb{R}_{\geq 0}^{m} \to \mathbb{R}_{\geq 0}^{n}$, where $m = bd$ and $n = ac$, as follows. Let R be a block diagonal matrix $\text{diag}(R_1, R_2, \ldots, R_d)$ with $R_i \in \mathbb{R}_{\geq 0}^{a \times b}$; similarly, let L be a block diagonal matrix $\text{diag}(L_1, L_2, \ldots, L_a)$ with $L_i \in \mathbb{R}_{\geq 0}^{c \times d}$. Let P_R (resp. P_L) be a permutation matrix that views a vector $\boldsymbol{x} \in \mathbb{R}^{ad}$ (resp. $\mathbb{R}^{ca}$) as $\boldsymbol{x} \in \mathbb{R}^{a \times d}$ (resp. $\mathbb{R}^{c \times a}$) and takes its transpose before flattening back to a vector $\mathbb{R}^{da}$ (resp. $\mathbb{R}^{ac}$). We call $\mathcal{M} = P_L L P_R R$ a (generalized) monarch matrix.

When $R_i = A$ for all i and $L_i = B$ for all i this corresponds exactly to circuit multiplication. The number of FLOPs for applying the linear transformation $\mathcal{M}$ to a vector is bounded by $O(abd + acd) = O(am + nd) \leq O(nm)$. The original monarch matrices (Dao et al. 2022) have been proposed to capture a wide range of linear transformations as the composition of a series of sparse yet structured matrices. We refer to our construction as generalized monarch matrices in the sense that if we have $m = n$, then our construction reduces to the original definition of a square monarch matrix.

Dao et al. (2022) has shown that by replacing dense layers with monarch matrices, deep neural networks can be trained to attain the same/similar performance with less # of training FLOPs. In light of this, we propose to replace dense sum blocks in PCs with (compositions of) Monarch matrices to scale up PCs in a computationally efficient way. In our experiments we use a composition of two Monarch matrices as this is known to provide additional expressivity.

4 Density Estimation Experiments

We benchmark our method on a language model distillation task and standard image datasets to demonstrate its competitive performance on common generative modeling tasks.

4.1 Distilling from Large Language Models

In this section, we present experiments on a language modeling task, focusing on distilling a GPT-2 Large model. Specifically, we use the GPT-2 Large checkpoint (finetuned for domain adaptation) provided by Zhang et al. (2023, 2024) as our base model. Following their pipeline, we sample 25.6M examples from the base model and train the HMM and Monarch-HMM for 2304 EM update steps, with each step processing 100K examples. Models are trained with hidden state sizes ranging from 128 to 65,536.

We use the number of floating point operations (FLOPs) per token as a measurement of a model efficiency. Given hidden states of size m, the FLOPs per token is m^2 for an HMM and $3m^{3/2}$ for a Monarch-HMM. We report the test log-likelihoods as a function of FLOPs per token in Figure 3. It shows that Monarch-HMM scales more effectively than HMM across varying computational budgets.

4.2 Large-Scale Image Models

In this section, we conduct experiments on the ImageNet32 dataset, which is a downscaled 32×32 version of ImageNet (Deng et al. 2009). Following recent work on PC modeling for these datasets (Liu, Zhang, and Van den Broeck 2023; Liu, Niepert, and Van den Broeck 2024; Liu, Ahmed, and

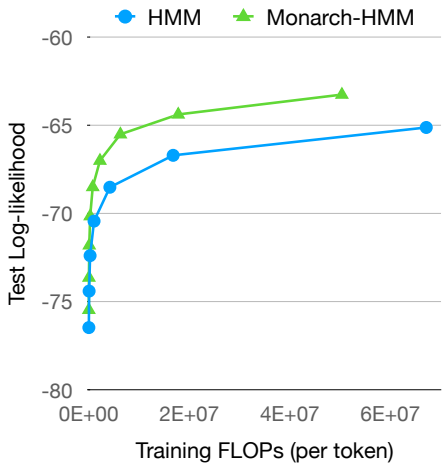

Figure 3: **Results for Distilling GPT-2 Large**. Test log-likelihoods (higher is better) as a function of training FLOPs per token. Monarch-HMM demonstrates greater efficiency than HMM across varying computational budgets.

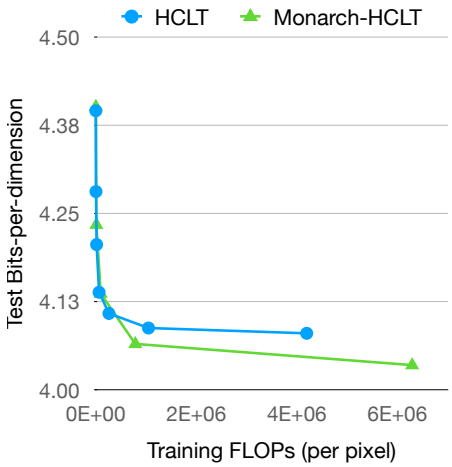

Figure 4: **Results for Training on Imagenet Dataset**. Test bits-per-dimension (lower is better) as a function of training FLOPs per pixel. Monarch-HCLT demonstrates greater efficiency than HCLT as the computation budget increases.

	ImageNet 32×32
LVD (2023)	4.38
LVD-PG (Liu et al. 2023)	4.06
QPC (Gala et al. 2024)	4.46
Monarch-HCLT (ours)	**4.04**

Table 1: **Density estimation on image datasets.** Test set log-likelihoods are in bits-per-dimension (lower is better). Our method performs favorably relative to all baselines.

Van den Broeck 2024), we transform the data from RGB using the lossy YCoCg transform. Note that likelihoods on on YCoCg transformed data are thus not comparable to likelihoods on the original RGB dataset. To improve training efficiency, we split each image into four 16×16 patches and train and evaluate our PCs over these 16×16 images. We train all models using expectation-maximization. We evaluate models using test-set bits-per-dimension (bpd); as this is normalized for dimension, our bpds are directly comparable with bpds on the entire 32×32 image.

We use hidden Chow-Liu trees (HCLT) (Liu and Van den Broeck 2021) to define the variable decomposition (*vtree*) of the PC. We also tie the parameters of the Categorical leaf distributions across pixels. We train for 1000 EM update steps over randomly selected batches of 60K examples from the ImageNet32 training set of 1.28M images. Analogously with HMMs for language modeling, we use the number of floating point operations per pixel as a measure of model efficiency, which is also m^2 for a HCLT and $3m^{3/2}$ for a Monarch-HCLT. We show the scaling plot for both PC variants in Figure 4.

It can be seen that Monarch-HCLTs show improved scaling beyond what is achievable with standard HCLTs. This enables our largest model, with $m = 16384$ hidden states, to achieve state-of-the-art performance for PCs on ImageNet32 (Table 1), beating even LVD-PG (Liu et al. 2023), which is an image-specialized PC-based model that uses a high-level PC over 4×4 patch PCs, together with a complex optimization procedure involving latent variable distillation (Liu, Zhang, and Van den Broeck 2023) for initializing the parameters and a progressive growing technique. In contrast, we simply use random initializations, the generic HCLT variable decomposition, and train using only the well-established EM algorithm for PCs. This can be attributed to the fact that, using Monarch matrices, the largest model we can fit in memory has $m = 16384$ hidden states, which is far larger than the $m = 512$ hidden states used by the largest models previously.

5 Related Work

Our work is connected to recent efforts in the probabilistic circuits community to find more efficient parameterizations of tensorized circuits. While early implementations of tensorized circuits used Kronecker product blocks (Peharz et al. 2020), the recent trend has been to prefer Hadamard product blocks. Loconte et al. (2024) noted that the composition of Kronecker/Hadamard product blocks with sum blocks can be interpreted as Tucker (Tucker 1964) / canonical-polyadic (CP) (Carroll and Chang 1970) tensor decompositions respectively. Our work tackles an orthogonal aspect in that it focuses on the sum-to-product connection, which has thus far always been implemented as a dense matrix.

6 Conclusion

We propose scaling up probabilistic circuits learning by replacing dense matrices in sum layers with structured sparse matrices, specifically Monarch matrices. Our approach demonstrates significant empirical improvements in both density estimation tasks and computational efficiency.

Acknowledgements

We thank Anji Liu for assistance with the PyJuice PC library and discussions on image dataset evaluation. BW and GVdB gratefully acknowledge support from the National Artificial Intelligence Research Resource Pilot under project NAIRR240143. This work was funded in part by the DARPA ANSR program under award FA8750-23-2-0004, the DARPA PTG Program under award HR00112220005, and NSF grant #IIS-1943641. MD and SE gratefully acknowledge the support from ARO (W911NF-21-1-0125), ONR (N00014-23-1-2159), and the CZ Biohub.

References

Ahmed, K.; Teso, S.; Chang, K.-W.; Van den Broeck, G.; and Vergari, A. 2022. Semantic probabilistic layers for neuro-symbolic learning. *Advances in Neural Information Processing Systems*, 35: 29944–29959.

Carroll, J. D.; and Chang, J.-J. 1970. Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition. *Psychometrika*, 35(3): 283–319.

Choi, Y.; Dang, M.; and Van den Broeck, G. 2021. Group fairness by probabilistic modeling with latent fair decisions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12051–12059.

Choi, Y.; Vergari, A.; and Van den Broeck, G. 2020. Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models.

Dang, M.; Khosravi, P.; Liang, Y.; Vergari, A.; and Van den Broeck, G. 2021. Juice: A Julia Package for Logic and Probabilistic Circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*.

Dang, M.; Liu, A.; and Van den Broeck, G. 2022. Sparse Probabilistic Circuits via Pruning and Growing. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*.

Dao, T.; Chen, B.; Sohoni, N. S.; Desai, A.; Poli, M.; Grogan, J.; Liu, A.; Rao, A.; Rudra, A.; and Ré, C. 2022. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, 4690–4721. PMLR.

Darwiche, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, 50(3): 280–305.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Gala, G.; de Campos, C.; Vergari, A.; and Quaeghebeur, E. 2024. Scaling Continuous Latent Variable Models as Probabilistic Integral Circuits. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Liu, A.; Ahmed, K.; and Van den Broeck, G. 2024. Scaling Tractable Probabilistic Circuits: A Systems Perspective. In *Proceedings of the 41th International Conference on Machine Learning (ICML)*.

Liu, A.; Niepert, M.; and Van den Broeck, G. 2024. Image Inpainting via Tractable Steering of Diffusion Models. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

Liu, A.; and Van den Broeck, G. 2021. Tractable Regularization of Probabilistic Circuits. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*.

Liu, A.; Zhang, H.; and Van den Broeck, G. 2023. Scaling Up Probabilistic Circuits by Latent Variable Distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Liu, X.; Liu, A.; Van den Broeck, G.; and Liang, Y. 2023. Understanding the Distillation Process from Deep Generative Models to Tractable Probabilistic Circuits. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Loconte, L.; Mari, A.; Gala, G.; Peharz, R.; de Campos, C.; Quaeghebeur, E.; Vessio, G.; and Vergari, A. 2024. What is the Relationship between Tensor Factorizations and Circuits (and How Can We Exploit it)? *arXiv preprint arXiv:2409.07953*.

Maene, J.; Derkinderen, V.; and Martires, P. Z. D. 2024. KLay: Accelerating Neurosymbolic AI. *arXiv preprint arXiv:2410.11415*.

Papamakarios, G.; Nalisnick, E.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2021. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*.

Peharz, R.; Lang, S.; Vergari, A.; Stelzner, K.; Molina, A.; Trapp, M.; Van den Broeck, G.; Kersting, K.; and Ghahramani, Z. 2020. Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Shen, Y.; Choi, A.; and Darwiche, A. 2016. Tractable operations for arithmetic circuits of probabilistic models. *Advances in Neural Information Processing Systems*, 29.

Tucker, L. R. 1964. The extension of factor analysis to three-dimensional matrices. In Gulliksen, H.; and Frederiksen, N., eds., *Contributions to mathematical psychology.*, 110–127. New York: Holt, Rinehart and Winston.

Vergari, A.; Choi, Y.; Liu, A.; Teso, S.; and Van den Broeck, G. 2021. A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*.

Wang, B.; and Kwiatkowska, M. 2023. Compositional probabilistic and causal inference using tractable circuit models. In *International Conference on Artificial Intelligence and Statistics*, 9488–9498. PMLR.

Wang, B.; Wicker, M. R.; and Kwiatkowska, M. 2022. Tractable uncertainty for structure learning. In *International Conference on Machine Learning*, 23131–23150. PMLR.

Wang, E.; Khosravi, P.; and Van den Broeck, G. 2021. Probabilistic Sufficient Explanations. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*.

Zečević, M.; Dhami, D.; Karanam, A.; Natarajan, S.; and Kersting, K. 2021. Interventional sum-product networks: Causal inference with tractable probabilistic models. *Advances in neural information processing systems*, 34: 15019–15031.

Zhang, H.; Dang, M.; Peng, N.; and Van den Broeck, G. 2023. Tractable Control for Autoregressive Language Generation. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Zhang, H.; Kung, P.-N.; Yoshida, M.; den Broeck, G. V.; and Peng, N. 2024. Adaptable Logical Control for Large Language Models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.