Understanding the Distillation Process from Deep Generative Models to Tractable Probabilistic Circuits

Xuejie Liu [*1] **Anji Liu** [*2] **Guy Van den Broeck** [2] **Yitao Liang** [3 4]

Abstract

Probabilistic Circuits (PCs) are a general and unified computational framework for tractable probabilistic models that support efficient computation of various inference tasks (e.g., computing marginal probabilities). Towards enabling such reasoning capabilities in complex real-world tasks, Liu et al. (2022) propose to distill knowledge (through latent variable assignments) from less tractable but more expressive deep generative models. However, it is still unclear what factors make this distillation work well. In this paper, we theoretically and empirically discover that the performance of a PC can exceed that of its teacher model. Therefore, instead of performing distillation from the most expressive deep generative model, we study *what properties the teacher model and the PC should have in order to achieve good distillation performance*. This leads to a generic algorithmic improvement as well as other data-type-specific ones over the existing latent variable distillation pipeline. Empirically, we outperform SoTA TPMs by a large margin on challenging image modeling benchmarks. In particular, on ImageNet32, PCs achieve 4.06 bits-per-dimension, which is only 0.34 behind variational diffusion models (Kingma et al., 2021).

1. Introduction

Developing Tractable Probabilistic Models (TPMs) that are capable of performing various inference tasks (e.g., computing marginals) is of great importance as they enable a wide range of downstream applications such as constrained generation (Peharz et al., 2020a; Correia et al., 2020), causal inference (Wang & Kwiatkowska, 2022), and data compression (Liu et al., 2021). Probabilistic Circuits (PCs) (Choi et al., 2020) refer to a class of TPMs with similar representations, including Sum-Product Networks (Poon & Domingos, 2011), and-or search spaces (Marinescu & Dechter, 2005), and arithmetic circuits (Darwiche, 2002). To take full advantage of the attractive inference properties of PCs, a key challenge is to improve their modeling performance on complex real-world datasets.

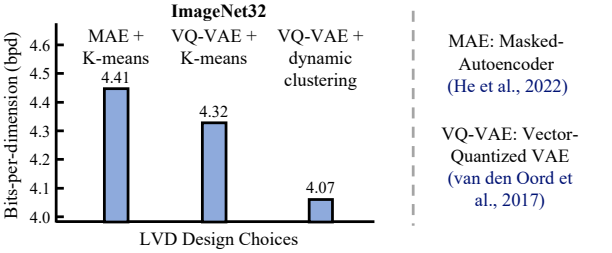

Figure 1. Different design choices in the LVD pipeline lead to drastically different performance (lower is better) on ImageNet32. All LVD-learned PCs have $\sim 200M$ parameters.

There have been significant recent efforts to scale up and improve PCs from both algorithmic (Correia et al., 2022; Shih et al., 2021; Dang et al., 2022; Peharz et al., 2020b) and architectural (Peharz et al., 2020a; Dang et al., 2021) perspectives. In particular, Liu et al. (2022) propose the Latent Variable Distillation (LVD) pipeline that uses less-tractable yet more expressive Deep Generative Models (DGMs) to provide extra supervision to overcome the suboptimality of Expectation-Maximization (EM) based PC parameter learners. With LVD, PCs are able to achieve competitive performance against some widely used DGMs on challenging datasets such as ImageNet32 (Deng et al., 2009).

However, despite its great potential, we have a limited understanding of when and how LVD leads to better modeling performance. As a result, the success of existing instantiations of the LVD pipeline relies heavily on trial and error. For example, as shown in Figure 1, modeling performance varies significantly as we change the DGM or the strategy to obtain supervision from them, even when the size of the PCs are similar.

This work aims to demystify the LVD pipeline and provide practical design guidelines for image data. By interpret-

[*]Equal contribution [1]Department of Automation, Tsinghua University, P.R. China [2]Computer Science Department, University of California, Los Angeles, USA [3]Institute for Artificial Intelligence, Peking University, P.R. China [4]Beijing Institute for General Artificial Intelligence (BIGAI). Correspondence to: Xuejie Liu <liebenxj@gmail.com>.

Preprint.

ing LVD from a variational inference perspective, we show that the performance of LVD-learned PCs is not necessarily upper bounded by their teacher DGMs. This is in sharp contrast with distilling knowledge from a large neural network to a smaller one, where the training performance typically degrades (Gou et al., 2021). Therefore, instead of trying to use the SoTA DGM to perform LVD, we should focus on a more fundamental question: *what properties of the teacher DGM would lead to better performance of LVD-learned PCs?* Although there is still no definite answer, we identify practical design guidelines that lead to expressive yet compact PCs.

Following the guidelines, we observe a general deficit in the existing LVD pipeline. Specifically, due to the mismatch between the discrete latent variable assignments requested by PCs and the continuous neural representations, a one-shot discretization method is often used. However, this causes significant information loss and leads to degraded modeling performance. To overcome this problem, we propose a *progressive growing* algorithm to leverage feedback from the PC to perform dynamic clustering, thus minimizing the performance loss caused by discretization. Progressive growing is also able to exploit reusable sub-structures, which leads to compact yet expressive PCs. Together with several image-specific design choices derived from the guidelines, we are able to out-perform SoTA TPMs by a large margin on three challenging image-modeling datasets: CIFAR (Krizhevsky et al., 2009) and two down-sampled ImageNet datasets (Deng et al., 2009). In particular, we achieve 4.06 bits-per-dimension on ImageNet32, which is better than some intractable Flow models and VAEs such as Glow (Kingma & Dhariwal, 2018) and only 0.34 less than the SoTA diffusion-based DGM (Kingma et al., 2021).

2. Background

This section introduces PCs (Sec. 2.1) and the Latent Variable Distillation (LVD) pipeline (Sec. 2.2).

2.1. Probabilistic Circuits

Probabilistic circuits (PCs) are a broad class of TPMs that characterize probability distributions as deep computation graphs. The syntax and semantics of PCs are as follows.

Definition 1 (Probabilistic Circuits)**.** Represented as a parameterized directed acyclic computation graph (DAG), a PC $p(\mathbf{X})$ defines a joint distribution over a set of random variables $\mathbf{X}$ by a single root node n_r. The nodes in the DAG are divided into three types of computational units: *input*, *sum*, and *product*. Notably, each leaf node in the DAG serves as an *input* unit, while an inner node can be subdivided into a *sum* unit or a *product* unit according to its mechanism for combining child distributions. In the forward path, every inner node receives *inputs* from its children (denoted $\mathsf{in}(n)$) and computes *outputs*, thus encoding a probability distribution p_n in a recursive fashion:

$$p_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & \text{if } n \text{ is an input unit,} \\ \sum_{c\in\mathrm{in}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) & \text{if } n \text{ is a sum unit,} \\ \prod_{c\in\mathrm{in}(n)} p_c(\boldsymbol{x}) & \text{if } n \text{ is a product unit,} \end{cases}$$

where $f_n(\boldsymbol{x})$ is a univariate probability distribution (e.g., Gaussian, Categorical), and $\theta_{n,c}$ represents the parameter corresponding to edge (n, c) in the DAG. Intuitively, a *sum* unit models a weighted mixture of its children's distributions, which requires all its edge parameters to be non-negative and sum up to one, i.e., $\sum_{c\in\mathsf{in}(n)} \theta_{n,c} = 1, \theta_{n,c} \geq 0$. And a product unit encodes a factorized distribution over its children. Finally, a PC represents the distribution encoded by its root node n_r. Additionally, we assume w.l.o.g. that a PC alternates between the sum and product layers before reaching its inputs.

The ability to answer numerous probabilistic queries (e.g., marginals, entropies, and divergences) (Vergari et al., 2021) exactly and efficiently distinguishes PCs from various deep generative models. Such ability is typically interpreted as *tractability*. To guarantee PCs' tractability, certain structural constraints have to be imposed on their DAG structure. For instance, *smoothness* together with *decomposability* ensure that a PC can compute arbitrary marginal probabilities in linear time w.r.t. its size, which is the number of edges in its DAG. These are properties of the variable scope $\phi(n)$ of PC unit n, that is, the variable set comprising all its descendent input nodes.

Definition 2 (Decomposability)**.** A PC is decomposable if for every product unit n, its children have disjoint scopes:

$$\forall c_1, c_2 \in \mathsf{in}(n)\,(c_1 \neq c_2),\ \phi(c_1) \cap \phi(c_2) = \varnothing.$$

Definition 3 (Smoothness)**.** A PC is smooth if for every sum unit n, its children have the same scope:

$$\forall c_1, c_2 \in \mathsf{in}(n),\ \phi(c_1) = \phi(c_2).$$

2.2. Latent Variable Distillation

Despite the recent breakthroughs in developing efficient computational frameworks for PCs (Dang et al., 2021; Molina et al., 2019), exploiting the additional expressive power of large-scale PCs remains extremely challenging. Abundant empirical evidence has attributed this phenomenon to the failure of existing EM-based optimizers to find good local optima in the large and hierarchically nested latent space of PCs (Peharz et al., 2016), which is defined by the hierarchically distributed sum units in their DAGs

Latent Variable Distillation (LVD) overcomes the aforementioned bottleneck by providing extra supervision to

PC optimizers through semantic-aware latent variable (LV) assignments, which are acquired from less tractable yet more expressive deep generative models (Liu et al., 2022). Specifically, LVD operates by first materializing some/all LVs in the PC. That is, transforming the original PC $p(\mathbf{X})$ into $p(\mathbf{X}, \mathbf{Z})$ whose marginal distribution over $\mathbf{X}$ stays unchanged, i.e., $p(\mathbf{X}) = \sum_{\boldsymbol{z}} p(\mathbf{X}, \mathbf{Z}=\boldsymbol{z})$.

Next, deep generative models (DGMs) are used to induce semantic-aware assignments of LVs $\mathbf{Z}$ for every training sample $\boldsymbol{x} \in \mathcal{D}_{\text{train}}$, leading to an augmented dataset $\mathcal{D}_{\text{aug}} := \{(\boldsymbol{x}, \boldsymbol{z}) : \boldsymbol{x} \in \mathcal{D}_{\text{train}}\}$. This LV induction step can be done in various ways and with different DGMs. For example, in Liu et al. (2022), $\mathbf{Z}$ is obtained by clustering the latent features produced by a Masked Autoencoder (He et al., 2022).

Finally, the augmented dataset $\mathcal{D}_{\text{aug}}$ is used to maximize a lower bound of the log-likelihood, as shown on the rightmost term:

$$\begin{aligned} \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)}\right) &:= \sum_{i=1}^{N} \log \sum_{\boldsymbol{z}} p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}\right), \\ &\geq \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}\right). \end{aligned} \tag{1}$$

After training with the augmented dataset, we can obtain the target distribution $p(\mathbf{X})$ by marginalizing out $\mathbf{Z}$, which can be done in linear (w.r.t. size of the PC) time (Choi et al., 2020). The PC can then be finetuned with the original dataset to improve performance further.

The success of LVD is primarily attributed to its ability to simplify the size and depth of PCs' deeply nested latent variable spaces (Peharz et al., 2016). Specifically, after LV materialization, supervision of the LVs can be provided by DGMs, and EM-based PC parameter learners are only responsible for inferring the values of the remaining implicitly defined LVs. Since the DGMs guide PC learning through their provided LV assignments, we refer to them as teacher models and the PCs as student models.

3. Characterizing Performance Gaps in LVD

Although LVD has demonstrated its potential to boost the performance of large PCs, its effectiveness depends strongly on the design choice of materialized LVs and how they are induced from external sources. Specifically, as shown in Figure 1, a bad design choice will lead to significantly worse performance, while a good one can further close the performance gap with SoTA intractable DGMs. Therefore, a crucial yet unanswered question concerning LVD is: *what are the design principles for the LV induction process to achieve good modeling performance?*

We provide a preliminary answer to this question by characterizing the performance differences between the teacher DGM and the student PC via variational inference (VI), which is the mathematical foundation of various DGMs such as VAEs (Kingma & Welling, 2013) and Diffusion models (Ho et al., 2020). Consider a latent variable model $p_\theta(\boldsymbol{x}) := \sum_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z})$. Instead of directly maximizing the log-likelihood $\log p_\theta(\boldsymbol{x})$, which could be infeasible, VI proposes to also learn a variational posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and maximize the following evidence lower bound (ELBO) of the log-likelihood:

$$\mathbb{E}_{\boldsymbol{z} \sim q_\phi(\cdot|\boldsymbol{x})} \left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\text{KL}}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z})\right). \tag{2}$$

Consider a PC $p_{\text{pc}}(\boldsymbol{x}) := \sum_{\boldsymbol{z}} p_{\text{pc}}(\boldsymbol{x}|\boldsymbol{z}) p_{\text{pc}}(\boldsymbol{z})$ defined on the same $\mathbf{X}$ and $\mathbf{Z}$ as above. The ultimate goal of LVD is to distill knowledge from $p_\theta(\boldsymbol{x})$ to $p_{\text{pc}}(\boldsymbol{x})$ to maximize the PC's log-likelihood.

A natural way to achieve this is to use $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ as the variational posterior for the PC. This leads to the following ELBO objective:

$$\mathbb{E}_{\boldsymbol{z} \sim q_\phi(\cdot|\boldsymbol{x})} [\log p_{\text{pc}}(\boldsymbol{x}|\boldsymbol{z})] - D_{\text{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p_{\text{pc}}(\boldsymbol{z})). \tag{3}$$

Although written in different forms, this ELBO objective is equivalent to Equation (1) up to a constant factor independent of the PC, (see Appx. A for a rigorous elaboration). Intuitively, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is treated as the external model to induce LV assignments $\boldsymbol{z}$ for every training sample $\boldsymbol{x}$. Therefore, we call Equation (3) the LVD objective.

The LVD objective provides a bridge to characterize the difference between the performance of the teacher DGM (Eq. (2)) and the log-likelihood of the student PC. Specifically, as shown in the Figure 2, the performance gap between the teacher DGM and the LVD objective, termed the LVD gap, characterizes the performance loss of LVD. However, the final performance difference between the teacher DGM and the student PC can be much less than the LVD gap. Specifically, thanks to the tractability of PC, $p_{\text{pc}}(\boldsymbol{z}|\boldsymbol{x})$ can be obtained in closed form. Therefore, the variational gap between the LVD objective and the PC's log-likelihood can be closed "for free" right after $p_{\text{pc}}(\boldsymbol{x}|\boldsymbol{z})$ and $p_{\text{pc}}(\boldsymbol{z})$ are trained by the LVD objective. That is, as demonstrated

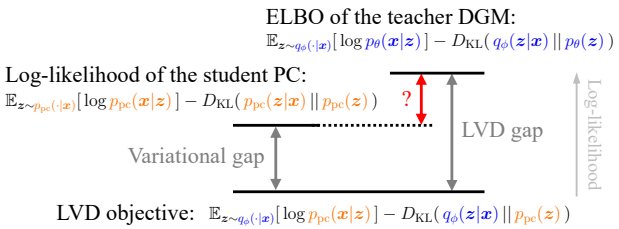

Figure 2. Performance difference of the teacher DGM (top-right) and the student PC (top-left) is characterized by the relative significance between the variational gap and the LVD gap.

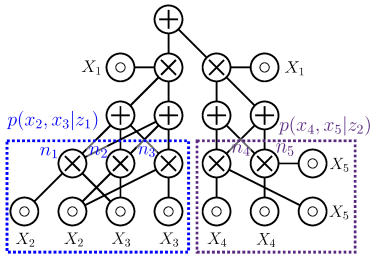

Figure 3. Example cluster-conditioned distributions represented by a PC.

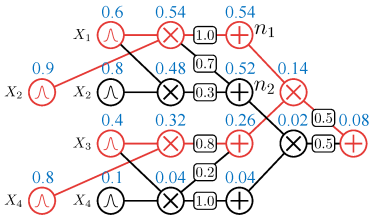

Figure 4. An example PC with likelihood of every node w.r.t. $\boldsymbol{x}$ labeled blue. Red nodes are "important" for modeling $\boldsymbol{x}$.

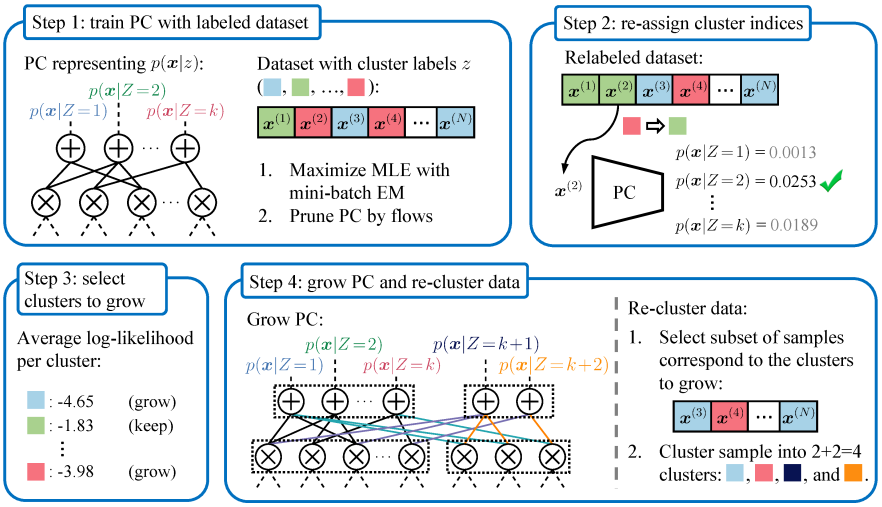

Figure 5. Illustration of the proposed progressive growing algorithm. The algorithm takes as input a dataset of pairs $(\boldsymbol{x}, \boldsymbol{h})$, where $\boldsymbol{h} = g_\phi(\boldsymbol{x})$ is a continuous neural representation of $\boldsymbol{x}$, and a PC representing $p(\boldsymbol{x}|z)$. The algorithm iterates through the four steps shown above to gradually expand cluster size (i.e., $|\mathbf{Z}|$) and grow the PC.

in Figure 2, after LVD training, which gives the ELBO shown at the bottom, we can directly obtain a PC with log-likelihood shown at the top-left side.

Perhaps surprisingly, the above analysis suggests that the log-likelihood of the student PC is not necessarily upper bounded by the ELBO of the teacher DGM. Specifically, as illustrated in Figure 2, this happens whenever the variational gap is larger than the LVD gap. In the extreme case where the student perfectly simulates the teacher (i.e., $p_\theta(\boldsymbol{x}\ \boldsymbol{z}) = p_{\text{pc}}(\boldsymbol{x}\ \boldsymbol{z})$ and $p_\theta(\boldsymbol{z}) = p_{\text{pc}}(\boldsymbol{z})$), the PC becomes a tractable instantiation of $p_\theta(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p_\theta(\boldsymbol{x}\ \boldsymbol{z}) p_\theta(\boldsymbol{z})$ that can compute various probabilistic queries. Towards achieving this ideal case, we need to minimize the LVD gap.

A key insight towards closing the LVD gap is to ensure $p_\theta(\boldsymbol{x}\ \boldsymbol{z})$ (resp. $p_\theta(\boldsymbol{z})$) has similar modeling assumptions to $p_{\text{pc}}(\boldsymbol{x}\ \boldsymbol{z})$ (resp. $p_{\text{pc}}(\boldsymbol{z})$). This works well in both directions: on the one hand, by mimicking the inductive biases of the DGM (i.e., $p_\theta(\boldsymbol{x}\ \boldsymbol{z})$ and $p_\theta(\boldsymbol{z})$), we can learn PCs that have better performance as well as fewer parameters; on the other hand, it is often beneficial to remove modeling assumptions in the DGM that cannot be fully adopted by PCs due to their structural constraints, though it might lead to worse performance of the DGM. In the following, we first identify a general source of modeling assumption mismatch, and propose an algorithm to mitigate this problem (Sec. 4). We then demonstrate how these design principles specialize to image data and improve modeling performance (Sec. 5).

4. Latent Variable Distillation from Continuous Neural Representations

A major challenge in minimizing the LVD gap is the mismatch between expressive *continuous* neural embeddings and the *discrete* nature of LVs materialized from PCs (since sum units represent discrete mixtures). Therefore, to obtain discrete LV assignments, either a post hoc discretization step (e.g., K-means) is used, or the DGM needs to learn discrete representations, which often leads to worse performance compared to learning continuous features. Such one-shot discretization strategies result in a relatively large LVD gap, which degrades PC modeling performance significantly.

Formally, the variational posterior of the DGM can be decomposed as

$$q_\phi(\boldsymbol{z}\ \boldsymbol{x}) := q(\boldsymbol{z}\ \boldsymbol{h}) \quad \text{where } \boldsymbol{h} = g_\phi(\boldsymbol{x})$$

Here g_ϕ is a neural network, and $\boldsymbol{h}$ is a continuous neural representation. Although we still have to discretize $\boldsymbol{h}$ to obtain LV assignments for the PC, this discretization procedure need not be one-shot: we can leverage *feedback* from the PC to adjust and re-assign cluster indices to narrow the LVD gap. Specifically, we want the cluster indices (i.e., $\boldsymbol{z}$) to be assigned in a way that both respect the neural representation $\boldsymbol{h}$ (i.e., samples with similar $\boldsymbol{h}$ are assigned to the same cluster) and are easily learned by the cluster-conditioned PCs $p_{\text{pc}}(\boldsymbol{x}\ \boldsymbol{z})$. Intuitively, while the latter condition ensures $p_{\text{pc}}(\boldsymbol{x}\ \boldsymbol{z})$ is properly learned, the former guarantees that $\boldsymbol{z}$ preserves information from $\boldsymbol{h}$, which empirically leads to a better $p_{\text{pc}}(\boldsymbol{z})$.

Before delving into the details of our solution, we briefly review the LV materialization process and illustrate the struc-

ture of cluster-conditioned PCs $p(\boldsymbol{x}\ \boldsymbol{z})$. Instead of assigning every sum unit an LV (as they represent mixture distributions), we group them according to their variable scopes (cf. Sec. 2.1), and assign every LV to a particular scope. Specifically, since the children of the sum units with every scope ϕ are all product units with the same scope, we can assign each child product unit a unique discrete value. Take the PC in Figure 3 as an example, we choose to materialize two LVs Z_1 and Z_2 w.r.t. the scopes $\phi_1 := \ X_2\ X_3\ $ and $\phi_2 := \ X_4\ X_5\ $, respectively. After LV materialization, we obtain two sub-PCs representing the cluster-conditioned distributions $p(x_2\ x_3\ z_1)$ and $p(x_4\ x_5\ z_2)$, respectively. Specifically, the child product node n_i (i 1 2 3) with scope ϕ_1 represents $p(x_2\ x_3\ Z_1 = i)$. Therefore, $p(x_2\ x_3\ z_1)$ is represented by the three-headed PC highlighted in the dashed blue box.

We proceed to describe the proposed *progressive growing* algorithm that overcomes the suboptimality of the aforementioned one-shot discretization method. The algorithm takes as input a dataset $\mathcal{D}_{\text{train}}$ accompanied with continuous neural embeddings, defined as $\mathcal{D} := \ (\boldsymbol{x}\ \boldsymbol{h}) : \boldsymbol{x}\ \mathcal{D}_{\text{train}}\ \boldsymbol{h} = g_\phi(\boldsymbol{x})\ $. Having materialized a LV Z that corresponds to scope $\mathbf{X}$, the algorithm also takes an initial cluster-conditioned PC $p(\boldsymbol{x}\ z)$ as input. We assume Z initially takes a single value (i.e., all samples in $\mathcal{D}_{\text{train}}$ belong to the same cluster), and thus $p(\boldsymbol{x}\ z)$ is represented by a single-headed PC.

Given a predefined number of clusters, denoted K, progressive growing aims to learn both a discretization function that maps every $\boldsymbol{h}$ into a cluster index $i\ \ [K]$, and a K-headed PC representing $p(\boldsymbol{x}\ Z = i)$ ($i\ \ [K]$). This is done by iteratively dividing $\mathcal{D}$ into more clusters and correspondingly learning the structure and parameters of the cluster-conditioned PCs. Specifically, as illustrated in Figure 5, progressive growing operates by repeating four main steps, which are detailed in the following.

Step 1: Training PC with Labeled Dataset. In this stage, we have access to a clustering function λ_k that maps every $\boldsymbol{h}$ to an index in $[k]$, where $1 \leq k \leq K$ is the current number of clusters, and a k-headed PC with the ith head encoding $p(\boldsymbol{x}\ Z = i)$. We train the PC by maximizing the conditional log-likelihood specified by $\mathcal{D}$ and λ_m:

$$\underset{\boldsymbol{\varphi}}{\text{maximize}} \sum_{(\boldsymbol{x},\boldsymbol{h})\in\mathcal{D}} \log p_{\boldsymbol{\varphi}}(\boldsymbol{x}\ Z = \lambda_k(\boldsymbol{h})) \tag{4}$$

where are the parameters of the PC. We optimize Equation (4) with the standard mini-batch EM algorithm (Peharz et al., 2020a; Choi et al., 2021). Hyperparameters are detailed in Appx. B. To learn a compact yet expressive PC, we apply the pruning algorithm proposed by Dang et al. (2022) after the parameter learning phase. This results in significantly smaller PCs with negligible performance loss.

Step 2: Re-assigning Cluster Indices. As hinted by the suboptimality of the one-step discretization method, cluster indices assigned by λ_k may not fully respect the PC $p(\boldsymbol{x}\ z)$. That is, since λ_k is obtained by clustering neural representation $\boldsymbol{h}$, some samples $\boldsymbol{x}$ assigned to cluster i could be better modeled by $p_{\boldsymbol{\varphi}}(\boldsymbol{x}\ Z = j)$ ($j = i$) trained in the previous step. To mitigate this problem, we leverage feedback from the PC to re-assign cluster indices. Specifically, as demonstrated in Figure 5, the cluster index of sample $\boldsymbol{x}$ is re-labeled as $z := \text{argmax}_{i\in[k]}\, p(\boldsymbol{x}\ Z = i)$. Function λ_k is modified correspondingly to reflect this change.

As we will elaborate more in the following steps, this relabeling process allows us to escape from poorly assigned clusters in past iterations, and is crucial to the effectiveness of progressive growing.

Step 3: Selecting Clusters to Grow. As suggested by its name, progressive growing operates by iteratively expanding the number of clusters in Z. To improve the overall performance of the cluster-conditioned PC (i.e., Eq. 4), we select clusters with low average log-likelihood to be further divided. Specifically, as illustrated in Figure 5, we first compute the average log-likelihood for each cluster $i\ \ [k]$:

$$\texttt{LL}_i := \frac{1}{\mathcal{D}_i} \sum_{\boldsymbol{x}\in\mathcal{D}_i} \log p(\boldsymbol{x}\ Z = i)$$

where $\mathcal{D}_i := \ \boldsymbol{x} : (\boldsymbol{x}\ \boldsymbol{h})\ \ \mathcal{D}\ \lambda_k(\boldsymbol{h}) = i\ $. We then select a subset of clusters based on $\texttt{LL}_i\ _{i=1}^{k}$ and the number of samples belonging to every cluster. See Appx. B for detailed design choices.

Step 4: Growing PC and Re-clustering Data. Suppose the previous step selects a set of cluster indices $\mathcal{I}$ for growing. The goal of this step is to expand these $\mathcal{I}$ clusters into M new clusters ($M > \mathcal{I}$). Under the hood, we need to re-cluster the corresponding subset of samples as well as apply structure modifications to the PC to fit the new clusters. Both procedures are described in the following.

To ensure that the structure and parameters of the multi-headed PC are still relevant to the cluster assignments λ_k after the reclustering step, a natural approach is to perform hierarchical growing and clustering to the PC and the dataset, respectively. Specifically, for each selected cluster $i\ \ \mathcal{I}$, we use K-means to cluster the samples belonging to the ith cluster into n clusters, and create $n-1$ new PC root units for the added clusters based on $p_{\boldsymbol{\varphi}}(\boldsymbol{x}\ Z = i)$. We use a slightly modified approach to re-cluster training samples for all $\mathcal{I}$ clusters simultaneously. Specifically, we first select the subset of samples $(\boldsymbol{x}\ \boldsymbol{h})$ belonging to clusters in $\mathcal{I}$. We then run K-means to cluster the neural representations $\boldsymbol{h}$ into K clusters, with the first $\mathcal{I}$ cluster centers initialized to be the centers of the clusters in $\mathcal{I}$. λ_k is then updated to reflect the new cluster assignments. In this way, the first $\mathcal{I}$ new clusters are still relevant to the corresponding PC

Algorithm 1 Grow Multi-Headed PCs

1: **Input:** A dataset $\mathcal{D}=\{(\boldsymbol{x}^{(i)},z^{(i)})\}_{i=1}^{N}$, where $z^{(i)}\in[k]$ is the cluster index of $\boldsymbol{x}^{(i)}$; a k-headed PC p
2: **Output:** A new multi-headed PC p'
3: Compute $F_n(\mathcal{D})$ for every PC unit n
4: $\texttt{G}\leftarrow\{n: F_n(\mathcal{D})\geq\epsilon\}$, where ϵ is a predefined threshold
5: $\texttt{old2new}\leftarrow\texttt{dict}()$ ▷ Maps n to a pair of (new) nodes
6: **foreach** n traversed in postorder **do** ▷ Child before parent
7: $\quad\texttt{ch}_1,\texttt{ch}_2\leftarrow\{\texttt{old2new}[c][0]\}_{c\in\texttt{in}(n)},\{\texttt{old2new}[c][1]\}_{c\in\texttt{in}(n)}$
8: $\quad$**if** n **isa** input unit **then**
9: $\quad\quad\texttt{old2new}[n]\leftarrow(n,\texttt{copy}(n))$ if $n\in\texttt{G}$ else (n,n)
10: $\quad$**elif** n **isa** product unit **then**
11: $\quad\quad\texttt{old2new}[n]\leftarrow(\bigotimes(\texttt{ch}_1),\bigotimes(\texttt{ch}_2))$
12: $\quad$**elif** $n\in\texttt{G}$ **isa** sum unit **then**
13: $\quad\quad\texttt{old2new}[n]\leftarrow(\bigoplus(\texttt{ch}_1,\texttt{ch}_2),\bigoplus(\texttt{ch}_1,\texttt{ch}_2))$
14: $\quad$**elif** $n\notin\texttt{G}$ **isa** sum unit **then**
15: $\quad\quad\texttt{old2new}[n]\leftarrow(\bigoplus(\texttt{ch}_1,\texttt{ch}_2),\texttt{None})$
16: **return** A multi-head PC with root nodes $\{\texttt{old2new}[n]: n \text{ is a root node in } p\}$

$p_{\boldsymbol{\varphi}}(\boldsymbol{x}\ Z=i)$ $(i\quad\mathcal{I})$.

In order to represent the newly-added clusters, the structure of the PC needs to be modified to contain $M-\mathcal{I}$ additional root/head units to represent $p(\boldsymbol{x}\ Z=i)$ $(i\quad k{+}1\quad k{+}M-\mathcal{I}\ )$. A simple strategy would be to directly copy all descendent units of $M-\mathcal{I}$ existing root units for the new clusters. However, this will significantly increase the size of the cluster-conditioned PC, rendering the progressive growing algorithm highly inefficient. Moreover, it rules out the possibility to reuse sub-circuits that are useful for modeling $\boldsymbol{x}$ conditioned on different z, seriously limiting the PC's expressive power at any particular size.

To mitigate this problem, we propose a structure growing operator that only copies the most important substructure for describing a distribution. By introducing additional edges between the original and copied sub-circuit, the PC can learn to share structures that can be used to describe $p(\boldsymbol{x}\ z)$ for various z. At the heart of the growing algorithm is a statistic termed *flow* that measures the generative significance of a node/edge w.r.t. a sample $\boldsymbol{x}$ (Dang et al., 2022; Liu & Van den Broeck, 2021), defined as follows.

Definition 4 (Circuit flow)**.** For a PC $p(\mathbf{X})$ and a sample $\boldsymbol{x}$, the circuit flow for every PC unit n, denoted $F_n(\boldsymbol{x})$, is defined recursively as follows ($\texttt{out}(n)$ denotes the set of parent units of n): first, $F_n(\boldsymbol{x})=1$ if n is the root unit; next, if n is a product unit, we have

$$F_n(\boldsymbol{x}):=\sum\nolimits_{m\in\texttt{out}(n)}\frac{\theta_{m,n}\cdot p_n(\boldsymbol{x})}{p_m(\boldsymbol{x})}\cdot F_m(\boldsymbol{x});$$

otherwise (n is a sum or input unit), the flow is defined by

$$F_n(\boldsymbol{x}):=\sum\nolimits_{m\in\texttt{out}(n)}F_m(\boldsymbol{x})$$

Intuitively, flow $F_n(\boldsymbol{x})$ quantifies the "contribution" of unit n to the log-likelihood of $\boldsymbol{x}$. Figure 4 demonstrates an example PC-sample pair with likelihoods labeled on top of every node. Nodes and edges with relatively high flows are labeled red. Note that high node likelihood does not guarantee high flow, which is illustrated by n_1 and n_2: they both have high likelihoods, but only n_1 has high flow. For a dataset $\mathcal{D}$, $F_n(\mathcal{D}):=\sum_{\boldsymbol{x}\in\mathcal{D}}F_n(\boldsymbol{x})$ measures the total contribution of n to the samples in $\mathcal{D}$.

Recall that our goal is to expand the current k-headed PC to have $M-\mathcal{I}$ additional root units to encode $p(\boldsymbol{x}\ Z=i)$ $(i\quad k{+}1\quad k{+}M-\mathcal{I}\ )$, respectively. To achieve this, we first extend Definition 4 for multi-headed PC. Specifically, while the recurrent definition of the inner nodes remain unchanged, for the ith root node, we set the flow to 1 if $\boldsymbol{x}$ is assigned to cluster i by λ_m and 0 otherwise.

The proposed growing operator is shown in Algorithm 1. It consists of two main parts: in lines 3-4, circuit flow is used to choose a subset of "important" (i.e., nodes with flow higher than a predefined threshold) nodes to be grown; in lines 5-15, the PC is modified in a way that only the selected nodes are duplicated, while other parts are kept unchanged. In our use case, since we want to modify the sub-circuit corresponds to the $\mathcal{I}$ chosen clusters, we invoke Algorithm 1 with the subset of samples whose cluster indices are in $\mathcal{I}$. According to the definition of flows, the returned PC will have $k+\mathcal{I}$ heads since the $\mathcal{I}$ chosen root nodes will be duplicated by the algorithm, while all other nodes will not.

Progressive growing alternates between the four steps described above until we have expanded the number of clusters to a predefined value K. Therefore, the parameters of the multi-headed PC grown by step #4 will be updated in step #1 of the algorithm's next iteration.

In summary, the data re-clustering process in step 4 ensures that the cluster assignments respect the neural representation, and the cluster assigning process in step 2 leads to well-fitted cluster-conditioned PCs.

5. Closing the LVD Gap for Image Data

Using image data as an example, this section demonstrates how the general guidelines for narrowing the LVD gap introduced in Section 3 can be specialized to practical design choices. Throughout this paper, we adopt Vector Quantized Variational Autoencoders (VQ-VAEs) (van den Oord et al., 2017; Razavi et al., 2019) as the teacher model. In the following, we first briefly introduce VQ-VAE. We then proceed to describe the design choices we make to better align the modeling assumptions of $p_\theta(\boldsymbol{x}\ \boldsymbol{z})$ (resp. $p_\theta(\boldsymbol{z})$) and $p_{\text{pc}}(\boldsymbol{x}\ \boldsymbol{z})$ (resp. $p_{\text{pc}}(\boldsymbol{z})$).

As shown in Figure 6, VQ-VAE consists of an encoder that produces a feature map, and a decoder that reconstructs the

Table 1. Density estimation performance of Tractable Probabilistic Models (TPMs) and Deep Generative Models (DGMs) on three natural image datasets. Reported numbers are test set bit-per-dimension (bpd), specifically, we report mean bpds and respective standard deviations over three runs. Bold indicates the best bpd (smaller is better) among all three TPMs. Best TPM w/o LVD represents the best performance over three TPMs: HCLT, RAT-SPN, and EiNet.

Dataset	TPMs			DGMs			
	LVD-PG (ours)	LVD	Best TPM w/o LVD	Glow	RealNVP	BIVA	VDM
ImageNet32	**4.06**±0.01	4.38	4.82	4.09	4.28	3.96	3.72
ImageNet64	**3.80**±0.07	4.12	4.67	3.81	3.98	-	3.40
CIFAR	**3.87**±0.00	4.37	4.61	3.35	3.49	3.08	2.65

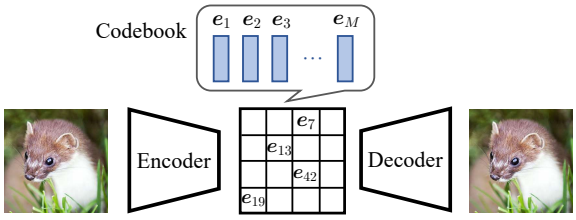

Figure 6. Illustration of the VQ-VAE model.

input image using the feature map. Different from many other DGMs, the latent feature map of VQ-VAEs is constructed by a codebook with M vectors representing M codes. Specifically, the latent embedding at each position must be a vector from the codebook. Since every latent code in the feature map corresponds to a patch of the input image, we materialize an LV Z_i for each position in the latent feature map, and define the corresponding image patch as $\mathbf{X}_i$. Denote $\mathbf{Z} := \ Z_i \ _i$, likelihood of an image $\boldsymbol{x}$ can be computed as $p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z}) \prod_i p(\boldsymbol{x}_i \ z_i)$. For every Z_i, we can learn $p(\boldsymbol{x}_i \ z_i)$ from the patches $\boldsymbol{x}_i$ and the corresponding continuous feature vectors $\boldsymbol{h}_i$ produced by VQ-VAE using the progressive growing algorithm detailed in the previous section. The generated discretization function can then be used to generate $\boldsymbol{z} = \ \boldsymbol{z}_i \ _i$ for every training sample, and is used to train $p(\boldsymbol{z})$.

However, in the above treatment, there are mismatches between the modeling assumptions made by VQ-VAE and the PC. First, since the latent feature map produced by VQ-VAE uses the same codebook at all locations, the cluster-conditioned distributions for different patches should be homogeneous. That is, for every discretization function λ and sample pair $(\boldsymbol{x} \ \boldsymbol{h})$, we have $\ i \ j \quad [\ \mathbf{Z}\ ]$,

$$p(\mathbf{X}_i = \boldsymbol{x} \ Z_i = \lambda(\boldsymbol{h})) \approx p(\mathbf{X}_j = \boldsymbol{x} \ Z_j = \lambda(\boldsymbol{h}))$$

To reflect this inductive bias, instead of learning $p(\boldsymbol{x} \ z_i)$ for every $i \quad [\ \mathbf{Z}\ ]$ independently, we aggregate their respective training samples and learn a single cluster-conditioned distribution, which is then applied to every image patch. That is, we do parameter tying between different cluster-conditioned distributions. This not only decreases the number of parameters of the PC, but also allows us to use much more data to train better cluster-conditioned distributions.

Another modeling assumption mismatch comes from the conditional independence between $\mathbf{X}_i$ and $\mathbf{X} \ \mathbf{X}_i$ given Z_i assumed by the PC. The convolutional decoder of a VQ-VAE breaks this assumption as $\boldsymbol{x}_i$ can correlate to other patches given $\boldsymbol{z}_i$. To mitigate this mismatch, we use an independent decoder where z_i is the only source of information used to generate $\boldsymbol{x}_i$. Although this will degrade the performance of VQ-VAE, as demonstrated in Section 3, the performance of LVD-learned PC can surpass the teacher model. And the primary goal of LVD is to find an initial set of parameters that can be optimized by the EM algorithm to good local optima. We will proceed to show this phenomenon in Section 6.1.

6. Experiments

This section first empirically verify the theoretical findings in Section 3 (Sec. 6.1). We then move on to evaluate our method on image modeling benchmarks (Sec. 6.2).

6.1. Analyzing Performance Gaps in LVD

We empirically investigate the finding in Section 3 that the log-likelihood of the student PC can surpass the ELBO of the teacher DGM. Specifically, we consider an instantiation of the LVD pipeline and empirically compute the three ELBOs shown in Figure 2. For ease of computation, we use VQ-VAE as the teacher model and one-shot K-means discretization strategy to train a PC on Imagenet32. The resulting ELBO of the teacher DGM is -2493 (Fig. 2 top-right), while the LVD objective is -2499 (Fig. 2 bottom). Therefore, the LVD gap is 5, which matches the extreme case mentioned in Section 4 (i.e., the student almost perfectly simulates the teacher). Hence the PC becomes a tractable instantiation of the teacher DGM. Thanks to PC's traceability, we are able to close the variational gap for free and obtain a PC with log-likelihood -2317, leading to a student PC better than the teacher DGM.

6.2. Image Modeling Benchmarks

We evaluate the proposed algorithmic improvements to the LVD pipeline on three natural image benchmarks: CIFAR

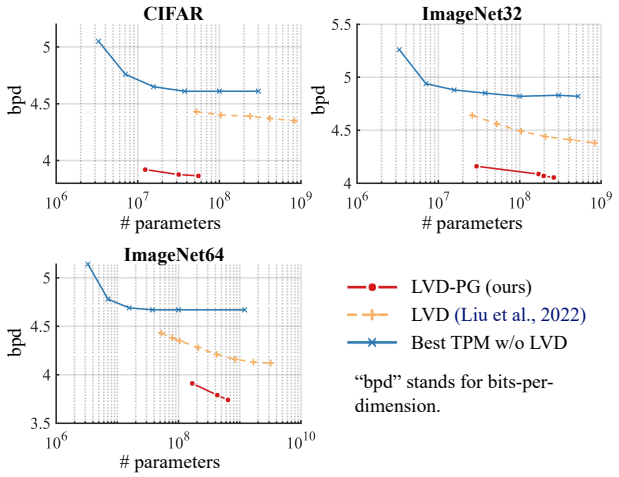

Figure 7. Comparison of image modeling performance. LVD-PG outperforms all baselines with similar sizes by a large margin.

(Krizhevsky et al., 2009) and two down-sampled ImageNet (ImageNet32 and ImageNet64) (Deng et al., 2009).

Baselines. We compare the proposed method, termed LVD with Progressive Growing (LVD-PG) against four TPM baselines: LVD (Liu & Van den Broeck, 2021), Hidden Chow-Liu Tree (HCLT) (Liu & Van den Broeck, 2021), Random Sum-Product Network (RAT-SPN) (Peharz et al., 2020b), and Einsum Network (EiNet). These baselines cover most of the recent endeavors on scaling up and improving the expressiveness of TPMs. Moreover, to evaluate the performance gap with less tractable DGMs, we further compare LVD-PG with the following flow-based models and variational autoencoders: Glow (Kingma & Dhariwal, 2018), RealNVP (Dinh et al., 2016), BIVA (Maaløe et al., 2019) and Variational Diffusion Models (VDM) (Kingma et al., 2021). Implementation details of the baselines can be found in Appx. C.

Empirical insights. We start by comparing our performance with other TPM models. As shown in Figure 7, for all benchmarks, LVD-PG consistently outperforms the previous approaches by a large margin. In particular, on CIFAR, a $\sim$12M LVD-PG model is much better than a $\sim$ 800M PC trained by LVD; on ImageNet32, a $\sim$20M PC trained by LVD-PG also obtains significant performance gain compared to a $\sim$800M PC trained by original LVD. This indicates that proper design choices can further exploit LVD's potential to train expressive yet compact PCs, thus significantly boosting the performances of large PCs.

Next, we compare the performance achieved by LVD-PG with the three adopted DGM baselines. Notably, as demonstrated in Table 1, our approach enables PCs to outperform all DGMs except the SoTA VDM on ImageNet64, and on ImageNet32, LVD-PG is only inferior to BIVA with a 0.1 bpd gap and VDM with a 0.34 bpd gap.

Ablation studies. To evaluate the effect of the progressive growing algorithm proposed in Section 4 and the image-data-specific modifications (such as using an "independent decoder" in VQ-VAE) elaborated in Section 5, we do an ablation analysis by training two other PCs without either component, respectively. Both PCs have similar model sizes as the SoTA PC trained with LVD-PG on ImageNet32 ($\sim$ 260M parameters). Specifically, compared to the SoTA LVD-learned PC with 4 06 bpd, the LVD-learned PC without progressive growing only achieves 4 12 bpd, while the performance of the LVD-learned PC with convolutional decoder degrades to 4 18 bpd.

7. Related work

There have been significant recent efforts to scale up and improve the expressiveness of PCs. Many works focus on constructing expressive yet compact initial PC structures (Rahman et al., 2014; Adel et al., 2015; Rooshenas & Lowd, 2014), while others aim for an iterative structure learning process that gradually increases model capacity (Di Mauro et al., 2021; Dang et al., 2020; Liang et al., 2017). These methods have led to significant performance gains on various density estimation datasets such as MNIST-family datasets.

However, improving the PC structure alone does not seem to offer too much performance gain on real-world high-dimensional datasets such as natural images and text. Towards solving this problem, there have been many recent endeavors to explore different ways of combining PCs with neural networks (NNs) to obtain tractable while expressive hybrid models. For example, Conditional SPNs (Shao et al., 2022) harness the expressive power of NNs to learn expressive conditional density estimators; HyperSPNs (Shih et al., 2021) use NNs to regularize the parameters of PCs; Correia et al. (2022) learn continuous mixtures of PCs with the help of continuous latent-space models represented by NNs.

A key to the above successes in scaling up PCs is the development of computation frameworks and easy-to-use libraries that make training large-scale PCs highly efficient. Specifically, EiNet (Peharz et al., 2020a) and SPFlow (Molina et al., 2019) leverage well-developed deep learning packages such as PyTorch (Paszke et al., 2019) to implement various inference and parameter learning procedures, and Juice.jl (Dang et al., 2021) implement custom kernels to better handle sparse PCs.

8. Conclusion

This paper aims to demystify the latent variable distillation process from intractable Deep Generative Models (DGMs) to tractable Probabilistic Circuits (PCs). We discover both theoretical and empirical evidence that the performance of the student PC can exceed that of the teacher DGM, where

the performance gain originates from the tractability of PCs that closes a variational gap "for free". Following this variational interpretation of the distillation technique, we further propose algorithmic improvements that lead to significant performance gain over SoTA TPMs. It also outperforms several intractable DGM baselines.

References

Adel, T., Balduzzi, D., and Ghodsi, A. Learning the structure of sum-product networks via an svd-based algorithm. In *UAI*, pp. 32–41, 2015.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai.cs.ucla.edu/papers/ProbCirc20.pdf*, 2020.

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

Correia, A. H., Peharz, R., and de Campos, C. Joints in random forests. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 11404–11415, 2020.

Correia, A. H., Gala, G., Quaeghebeur, E., de Campos, C., and Peharz, R. Continuous mixtures of tractable probabilistic models. *arXiv preprint arXiv:2209.10584*, 2022.

Dang, M., Vergari, A., and Broeck, G. Strudel: Learning structured-decomposable probabilistic circuits. In *International Conference on Probabilistic Graphical Models*, pp. 137–148. PMLR, 2020.

Dang, M., Khosravi, P., Liang, Y., Vergari, A., and Van den Broeck, G. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, Feb 2021.

Dang, M., Liu, A., and Van den Broeck, G. Sparse probabilistic circuits via pruning and growing. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, dec 2022.

Darwiche, A. A logical approach to factoring belief networks. In *Proc. 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-02)*, 2002.

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

Di Mauro, N., Gala, G., Iannotta, M., and Basile, T. M. Random probabilistic circuits. In *Uncertainty in Artificial Intelligence*, pp. 1682–1691. PMLR, 2021.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. In *International Conference on Learning Representations*, 2016.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Kingma, D. P. and Dhariwal, P. Glow: generative flow with invertible $1\times$ 1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 10236–10245, 2018.

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

Kingma, D. P., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. In *Advances in Neural Information Processing Systems*, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Technical report*, 2009.

Liang, Y., Bekker, J., and Van den Broeck, G. Learning the structure of probabilistic sentential decision diagrams. In *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Liu, A. and Van den Broeck, G. Tractable regularization of probabilistic circuits. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, dec 2021.

Liu, A., Mandt, S., and Van den Broeck, G. Lossless compression with probabilistic circuits. In *International Conference on Learning Representations*, 2021.

Liu, A., Zhang, H., and Broeck, G. V. d. Scaling up probabilistic circuits by latent variable distillation. *arXiv preprint arXiv:2210.04398*, 2022.

Maaløe, L., Fraccaro, M., Liévin, V., and Winther, O. Biva: a very deep hierarchy of latent variables for generative modeling. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 6551–6562, 2019.

Marinescu, R. and Dechter, R. And/or branch-and-bound for graphical models. In *IJCAI*, pp. 224–229, 2005.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, jul 2020a.

Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pp. 334–344. PMLR, 2020b.

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

Rahman, T., Kothalkar, P., and Gogate, V. 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*, pp. 630–645. Springer, 2014.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

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

Shao, X., Molina, A., Vergari, A., Stelzner, K., Peharz, R., Liebig, T., and Kersting, K. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140:298–313, 2022.

Shih, A., Sadigh, D., and Ermon, S. Hyperspns: Compact and expressive probabilistic circuits. In *Advances in Neural Information Processing Systems*, 2021.

van den Oord, A., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 6309–6318, 2017.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, dec 2021.

Wang, B. and Kwiatkowska, M. Symbolic causal inference via operations on probabilistic circuits. In *NeurIPS 2022 Workshop on Neuro Causal and Symbolic AI (nCSI)*, 2022.

A. Equivalence Between the Two LVD Formulations

Consider a sample $\boldsymbol{x}$. Suppose its LV assignment $\boldsymbol{z}$ is generated by $q_\phi(\boldsymbol{z}\ \boldsymbol{x})$. Then Equation (1) can be written as:

$$\begin{aligned}\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_{pc}(\boldsymbol{x}\ \ \boldsymbol{z})] &= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(p_{pc}(\boldsymbol{x}\ \boldsymbol{z})p_{pc}(z)\right)\right] \\ &= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_{pc}(\boldsymbol{x}\ \boldsymbol{z})] + \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_{pc}(\boldsymbol{z})}{q_\phi(\boldsymbol{z}\ \boldsymbol{x})}q_\phi(\boldsymbol{z}\ \boldsymbol{x})\right)\right] \\ &= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_{pc}(\boldsymbol{x}\ \boldsymbol{z})] - D_{\mathrm{KL}}\left(q_\phi(\boldsymbol{z}\ \boldsymbol{x})\ \ p_{pc}(\boldsymbol{z})\right) + \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log q_\phi(\boldsymbol{z}\ \boldsymbol{x})]\end{aligned}$$

The first two terms of the last equation are the LVD objective shown in Equation (3). Since the last term is independent with the PC, we conclude that Equations (1) and (3) are equivalent up to a constant factor independent with the PC.

B. Details of the Progressive Growing Algorithm

Training PC with Labeled Dataset. For the cluster-conditioned distribution, we adopt multi-head HCLTs with hidden size 16 and run mini-batch EM optimization with batch size 256. The learning rate anneals linearly from 0.1 to 0.01 for 50 epochs.

Selecting Clusters to Grow. The cluster set $\mathcal{I}$ selected to grow initiates with an empty set, then we choose the cluster with the smallest LL and push all samples belonging to this cluster into $\mathcal{I}$ successively until its capacity reaches a certain threshold. In our experiments, the threshold is fixed to be 40% of the total number of samples.

Additional Details. Given $\mathcal{D} := \ (\boldsymbol{x}\ \boldsymbol{h}) : \boldsymbol{x}\ \ \mathcal{D}_{\text{train}}\ \boldsymbol{h} = g_\phi(\boldsymbol{x})\ $, we first use K-means to pre-cluster the training samples into N_1 outer clusters based on their continuous neural embeddings. Then we apply the progressive growing algorithm to grow each outer cluster up to N_2 inner clusters, which initiates with a single-head HCLT. Specifically, when N_1 equals one, the pipeline is equivalent to growing clusters from scratch, and the smaller total cluster number $N_1 \times N_2$ typically corresponds to smaller PCs. Empirically we vary N_1 from 50 to 400 and adjust N_2 from 20 to 3 accordingly. On the three image benchmarks: Imagenet32, Imagenet64 and CIFAR10, the (N_1,N_2) adopted by our SoTA LVD-learned PC are (400,4), (320,4), and (100,4), respectively.

C. Implementation Details of the Baselines

To ensure a fair comparison, we implement HCLT and RAT-SPN with the Julia package Juice.jl (Dang et al., 2021) and adopt the original PyTorch implementation of EiNet. For all TPMs, we train a series of models with their number of parameters ranging from $\sim$1M to $\sim$100M and tune hyperparameters accordingly. Finally, we choose the best model among these TPM baselines as the Best TPM w/o LVD. We also report the best performance of each TPM in the following table.

Dataset	HCLT	EiNet	RAT-SPN
ImageNet32	4.82	5.63	6.90
ImageNet64	4.67	5.69	6.82
CIFAR	4.61	5.81	6.95