LOSSLESS COMPRESSION WITH PROBABILISTIC CIRCUITS

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite extensive progress on image generation, deep generative models are suboptimal when applied to lossless compression. For example, models such as VAEs suffer from a compression cost overhead due to their latent variables that can only be partially eliminated with elaborated schemes such as bits-back coding, resulting in oftentimes poor single-sample compression rates. To overcome such problems, we establish a new class of tractable lossless compression models that permit efficient encoding and decoding: Probabilistic Circuits (PCs). These are a class of neural networks involving $|p|$ computational units that support efficient marginalization over arbitrary subsets of the D feature dimensions, enabling efficient arithmetic coding. We derive efficient encoding and decoding schemes that both have time complexity $\mathcal{O}(\log(D) \cdot |p|)$, where a naive scheme would have linear costs in D and $|p|$, making the approach highly scalable. Empirically, our PC-based (de)compression algorithm runs 5-20x faster than state-of-the-art neural compression algorithms while achieving similar bitrates. By scaling up the traditional PC structure learning pipeline, we achieved state-of-the-art results on image datasets such as MNIST. Furthermore, PCs can be naturally integrated with existing neural compression algorithms to achieve state-of-the-art performance on natural image datasets. Our results highlight the potential impact that non-standard learning architectures may have on neural data compression.

1 INTRODUCTION

Thanks to their expressiveness, modern Deep Generative Models (DGMs) such as Flow-based models (Dinh et al., 2014), Variational Autoencoders (VAEs) (Kingma & Welling, 2013), and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) achieved state-of-the-art results on generative tasks such as creating high-quality samples (Vahdat & Kautz, 2020) and learning low-dimensional representation of data (Zheng & Sun, 2019). However, these successes have not been fully transferred into lossless compression. Specifically, GANs cannot be used for lossless compression due to their inability to assign likelihoods to observations. Latent variable models such as VAEs rely on rate estimates obtained by lower-bounding the marginal likelihood of the data, i.e., the quantity which is theoretically optimal for lossless compression; they furthermore rely on sophisticated schemes such as bits-back coding (Hinton & Van Camp, 1993) to realize these rates, oftentimes resulting in poor single-sample compression ratios (Kingma et al., 2019).

Since good generative performance does not imply good compression performance, one may wonder if there exists a class of models currently overlooked by the compression community that ensures (i) good compression rates and (ii) tractable algorithms to efficiently encode and decode close to the model's theoretical rate estimate. This paper answers this question affirmatively and for the first time proposes Probabilistic Circuits (PCs) (Choi et al., 2020) for lossless compression. This class of *tractable* models has a special structure that allows efficient marginalization of its random variables–a property that, as we show, enables efficient conditional entropy coding. Therefore, we introduce PCs as backbone models and develop (de)compression algorithms that achieve high compression ratios and high computational efficiency.

Similar to other neural compression methods, the proposed lossless compression approach operates in two main phases — (i) learn good PC models that approximate the data distribution, and (ii) compress and decompress samples $\boldsymbol{x}$ with computationally efficient algorithms. To provide a first impression

of PCs' performance, we review our empirical and theoretical results before digging into details on the (de)compression algorithm and model learning in the following sections.

First, we propose efficient compression and decompression algorithms that not only achieve close-to-optimal bitrates given a learned PC model (Shannon, 1948), but also run 5-20x faster than existing neural compression methods while realizing similar or better compression rates (Table 2). We prove that compression and decompression take $\mathcal{O}(\log(D) \cdot |p|)$ time, where D is the number of feature dimensions and $|p|$ is the size of the PC model (Thm. 1).

Second, by pushing further recent breakthroughs in large-scale training and model design of PCs (Dang et al., 2021; Peharz et al., 2020a; Liu & Van den Broeck, 2021), we achieved competitive likelihoods and compression rates on various datasets. Specifically, as shown in Table 3, PCs out-perform baselines such as IDF (Hoogeboom et al., 2019), BitSwap (Kingma et al., 2019), and JPEG2000 (Christopoulos et al., 2000) on image datasets such as MNIST. Furthermore, PCs can be naturally integrated with Flow- and VAE-based generative models, and significantly improve their performance on natural image datasets such as sub-sampled ImageNet (Table 4).

Notation We denote random variables by uppercase letters (e.g., X) and their assignments by lowercase letters (e.g., x). Analogously, we use bold uppercase (e.g., $\mathbf{X}$) and lowercase (e.g., $\boldsymbol{x}$) letters to denote sets of variables and their joint assignments, respectively. The set of all possible assignments of $\mathbf{X}$ is denoted $\mathsf{val}(\mathbf{X})$.

2 Tractability Matters in Lossless Compression

The goal of lossless compression algorithms is to map every input sample to an output codeword such that (i) the original input can be reconstructed from the codeword, and (ii) the expected length of the codewords is minimized. Practical (neural) lossless compression algorithms operate in two main phases — learning and compression. In the learning phase, a generative model $p(\mathbf{X})$ is learned from a dataset $\mathcal{D} := \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$. According to Shannon's source coding theorem (Shannon, 1948), the expected codeword length is lower-bounded by the negative cross-entropy between the data distribution $\mathcal{D}$ and the model distribution p (i.e., $-\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[\log p(\boldsymbol{x})]$), rendering it a natural and widely used objective to optimize the model (Hoogeboom et al., 2019; Mentzer et al., 2019).

In the compression phase, compression algorithms take the learned model p and samples $\boldsymbol{x}$ as input and generate codewords whose expected length approaches the theoretical limit (i.e., the negative cross-entropy between $\mathcal{D}$ and p). Although there exist various close-to-optimal compression schemes (e.g., Huffman Coding (Huffman, 1952) and Arithmetic Coding (Rissanen, 1976)), a natural question to ask is *what are the requirements on the model p such that compression algorithms can utilize it for encoding/decoding in a computationally efficient manner?* In this paper, we highlight the advantages of *tractable* probabilistic models for lossless compression by introducing a concrete class of models that are expressive and support efficient encoding and decoding.

To encode a sample $\boldsymbol{x}$, a standard streaming code operates by sequentially encoding every symbol x_i into a single bitstream b, such that x_i occupies approximately $-\log p(x_i|x_1, \ldots, x_{i-1})$ bits in b. As a result, the length of b is approximately $-\log p(\boldsymbol{x})$. For example, Arithmetic Coding (AC) encodes the symbols $\{x_i\}_{i=1}^{D}$ (define $D := |\mathbf{X}|$ as the number of features) sequentially by successively refining an interval that represents the sample, starting from the initial interval $[0, 1)$. To encode x_i, the algorithm partitions the current interval $[a, b)$ using the left and right side cumulative probability of x_i:

$$l_i(x_i) := p(X_i < x_i \mid x_1, \ldots, x_{i-1}), \qquad h_i(x_i) := p(X_i \leq x_i \mid x_1, \ldots, x_{i-1}). \tag{1}$$

Specifically, the algorithm updates $[a, b)$ to the following: $[a + (b-a) \cdot l_i(x_i), a + (b-a) \cdot h_i(x_i))$, which is a sub-interval of $[a, b)$. Finally, AC picks a number within the final interval that has the shortest binary representation. This number is encoded as a bitstream representing the codeword of $\boldsymbol{x}$. Upon decoding, the symbols $\{x_i\}_{i=1}^{D}$ are decoded sequentially: at iteration i, we decode variable X_i by looking up its value x such that its cumulative probability (i.e., $l_i(x)$) matches the subinterval specified by the codeword and $x_1, \ldots, x_{i-1}$ (Rissanen, 1976); the decoded symbol x_i is then used to compute the following conditional probabilities (i.e., $l_j(x)$ for $j > i$). Despite implementation differences, computing the cumulative probabilities $l_i(x)$ and $h_i(x)$ are required for many other streaming codes (e.g., rANS). For these streaming codes, the main computation cost of both the encoding and decoding process comes from calculating $l_i(x)$ and $h_i(x)$.

Table 1: An (incomplete) summary of our empirical results. "Comp." stands for compression.

Method	MNIST (10,000 test images)		
	Theoretical bpd	Comp. bpd	En- & decoding time
PC (small)	1.26	1.30	**59**
PC (large)	**1.20**	**1.24**	199
IDF	1.90	1.96	880
BitSwap	1.27	1.31	904

Method	ImageNet32	ImageNet64
	Theoretical bpd	Theoretical bpd
PC+IDF	**3.99**	**3.71**
IDF	4.15	3.90
RealNVP	4.28	3.98
Glow	4.09	3.81

The main challenge for the above (de)compression algorithm is to balance the expressiveness of p and the computation cost of $\{l_i(x), h_i(x)\}_{i=1}^D$. On one hand, highly expressive probability models such as Autoregressive neural networks (Salimans et al., 2017; Papamakarios et al., 2017) enjoy high compression ratios at the cost of slow runtime — decoding a sample requires evaluating the model D times, where D is oftentimes several hundred in practical datasets.[1] On the other hand, adding strong independence assumptions in p (e.g., $\mathbf{X}$ are mutually independent) makes $l_i(x)$ and $h_i(x)$ highly tractable. However, model expressiveness is greatly sacrificed.[2]

This paper explores the middle-ground between the above two extremes. Specifically, we ask: *are there probabilistic models that are both expressive and permit efficient computation of the conditional probabilities in Eq.* (1)*?* This question can be answered in its affirmative by establishing a new class of tractable lossless compression algorithms using Probabilistic Circuits (PCs) (Choi et al., 2020), which are neural networks that can compute various probabilistic queries efficiently. In the following, we overview the empirical and theoretical results of the proposed (de)compression algorithm.

We start with theoretical findings: the proposed encoding and decoding algorithms enjoy time complexity $\mathcal{O}(\log(D) \cdot |p|)$, where $|p| \geq D$ is the PC model size. The backbone of both algorithms, formally introduced in Sec. 3, is an algorithm that computes the $2D$ conditional probabilities $\{l_i(x), h_i(x)\}_{i=1}^D$ given any $\boldsymbol{x}$ in $\mathcal{O}(\log(D) \cdot |p|)$ time, as justified by the following theorem.

Theorem 1 (informal). *Let $\boldsymbol{x}$ be a D-dimensional sample, and let p be a PC model of size $|p|$, as proposed in this paper. We then have that computing all quantities $\{l_i(x_i), h_i(x_i)\}_{i=1}^D$ takes $\mathcal{O}(\log(D) \cdot |p|)$ time. Therefore, en- or decoding $\boldsymbol{x}$ with a streaming code (e.g., Arithmetic Coding) takes $\mathcal{O}(\log(D) \cdot |p| + D) = \mathcal{O}(\log(D) \cdot |p|)$ time.*

The properties of PCs that enable this efficient lossless compression algorithm will be described in Sec. 3.1, and the backbone inference algorithm that achieves $\mathcal{O}(\log(D) \cdot |p|)$ time complexity will later be shown as Alg. 2.

Table 1 provides an (incomplete) summary of our empirical results. First, the PC-based lossless compression algorithm is fast and competitive. As shown in Table 1, the small PC model achieved 1.30 compression bits-per-dimension (bpd). Compared to BitSwap (Kingma et al., 2019), which achieved a similar bitrate (1.31), our approach is $\sim$15x faster in terms of en- and decoding speed. Furthermore, a larger PC model achieved a much better bitrate (1.24). Next, PCs can be integrated with Flow- and VAE-based compression methods by replacing their simple latent variables' distributions. Take IDF (Hoogeboom et al., 2019) as an example, as illustrated in Table 1, the integrated model achieved new state-of-the-art results on complex natural image datasets such as sub-sampled ImageNet.

3 Computationally Efficient (De)compression with PCs

In the previous section, we have boiled down the task of lossless compression into calculating conditional probabilities $\{l_i(x), h_i(x)\}_{i=1}^D$ given p and x. This section takes PCs into consideration and demonstrates how these queries can be computed efficiently. In the following, we first introduce relevant background on PCs (Sec. 3.1), and then proceed to introduce an efficient algorithm to compute the conditional probabilities (Sec. 3.2). Finally, we empirically evaluate the optimality and speed of the proposed compressor and decompressor (Sec. 3.3).

[1] Note that high-resolution images can be generally compressed in patches.

[2] Flow-model-based neural compression algorithms adopt p defined on mutually independent latent variables (denoted $\mathbf{Z}$), and improve expressiveness by learning bijection functions between $\mathbf{Z}$ and $\mathbf{X}$ (i.e., the input space). This is orthogonal to our approach of directly learn better p. Furthermore, we can naturally integrate proposed expressive p with the bijection functions and achieve better performance as demonstrated in Sec. 5.

3.1 BACKGROUND: PROBABILISTIC CIRCUITS

Probabilistic Circuits (PCs) are an umbrella term of a wide variety of Tractable Probabilistic Models (TPMs). They provide a set of succinct definitions for popular TPMs such as Sum-Product Networks (Poon & Domingos, 2011), Arithmetic Circuits (Shen et al., 2016), and Probabilistic Sentential Decision Diagrams (Kisa et al., 2014). The syntax and semantics of a PC are defined as follows.

Definition 1 (Probabilistic Circuits)**.** A PC $p(\mathbf{X})$ represents a probability distribution over $\mathbf{X}$ via a parametrized directed acyclic graph (DAG) with a single root node n_r. Similar to neural networks, every node of the DAG defines a computational unit. Specifically, each leaf node corresponds to an *input unit*; each inner node n represents either a *sum* or a *product unit* that receives inputs from its children, denoted $\mathsf{in}(n)$. Each node n encodes a probability distribution p_n, defined as follows:

$$p_n(\boldsymbol{x}) := \begin{cases} f_n(\boldsymbol{x}) & \text{if } n \text{ is an input unit,} \\ \sum_{c \in \mathsf{in}(n)} \theta_{n,c} \cdot p_c(\boldsymbol{x}) & \text{if } n \text{ is a sum unit,} \\ \prod_{c \in \mathsf{in}(n)} p_c(\boldsymbol{x}) & \text{if } n \text{ is a product unit,} \end{cases} \tag{2}$$

where $f_n(\cdot)$ is an univariate input distribution (e.g., Gaussian, Categorical), and $\theta_{n,c}$ denotes the parameter that corresponds to edge (n, c). Intuitively, sum and product units encode weighted mixtures and factorized distributions of their children's distributions, respectively. To ensure a PC models a valid distribution, we assume the parameters associated with any sum unit n are normalized: $\forall n, \sum_{c \in \mathsf{in}(n)} \theta_{n,c} = 1$. We further assume w.l.o.g. that a PC alternates between sum and product units before reaching an input unit. The size of a PC p, denoted $|p|$, is the number of edges in its DAG.

This paper focuses on PCs that can compute arbitrary marginal queries in time linear in their size, since this is necessary to unlock the efficient (de)compression algorithm. In order to support efficient marginalization, PCs need to satisfy two *structural properties* — smoothness (Def. 2) and decomposability (Def. 3).[3] Both are properties of the (variable) *scope* $\phi(n)$ of PC units n, that is, the collection of variables defined by all its descendent input units.

Definition 2 (Smoothness)**.** A PC is smooth if for every sum unit n, its children have the same scope: $\forall c_1, c_2 \in \mathsf{in}(n), \phi(c_1) = \phi(c_2)$.

Definition 3 (Decomposability)**.** A PC is decomposable if for every product unit n, its children have disjoint scopes: $\forall c_1, c_2 \in \mathsf{in}(n)\ (c_1 \neq c_2), \phi(c_1) \cap \phi(c_2) = \varnothing$.

In addition to Def. 2 and 3, we introduce another structural property, *structured decomposability*, which is the key to guaranteeing computational efficiency of the proposed (de)compression algorithm.

Definition 4 (Structured decomposability)**.** A PC is structured-decomposable (abbreviated as SD) if (i) it is decomposable and (ii) for every pair of product units (m, n) with identical scope (i.e., $\phi(m) = \phi(n)$), we have that $|\mathsf{in}(m)| = |\mathsf{in}(n)|$ and the scopes of their children are pairwise identical: $\forall i \in \{1, ..., |\mathsf{in}(m)|\}, \phi(cm_i) = \phi(cn_i)$, where cm_i and cn_i are the ith child unit of m and n.

As a key sub-routine in the proposed algorithm, we describe how to compute marginal queries given a smooth and (structured-)decomposable PC in $\mathcal{O}(|p|)$ time. First, we assign probabilities to every input unit: for an input unit n defined on variable X, if evidence is provided for X in the query (e.g., $X = x, X < x$), we assign to n the corresponding probability (e.g., $p(X = x), p(X < x)$) according to f_n in Eq. (2); if evidence of X is not given, probability 1 is assigned to n. Next, we do a bottom-up (children before parents) traverse of inner PC units and compute their probabilities following Eq. (2). The probability assigned to the root unit is the final answer of the marginal query.

3.2 EFFICIENT MARGINALIZATION WITH PCS

As we will proceed to show, efficient computation of the conditional probabilities (Eq. (1)) relies on an optimal variable order over $\mathbf{X}$. To reflect this idea, we generalize $l_i(x)$ and $h_i(x)$ to

$$l_{\pi,i}(x) := p(X_{\pi_i} < x \mid x_{\pi_1}, \ldots, x_{\pi_{i-1}}), \qquad h_{\pi,i}(x) := p(X_{\pi_i} \leq x \mid x_{\pi_1}, \ldots, x_{\pi_{i-1}}),$$

where π defines a random variable order over $\mathbf{X}$, i.e., the ith variable in the order defined by π is X_{π_i}. Again, both the encoding and decoding algorithm require computing $\{l_{\pi,i}(x), h_{\pi,i}(x)\}_{i=1}^{D}$ w.r.t. p, π and various $\boldsymbol{x}$. Since every conditional probability can be represented as the quotient

[3] Sufficiency of the conditions are proved by Poon & Domingos (2011).

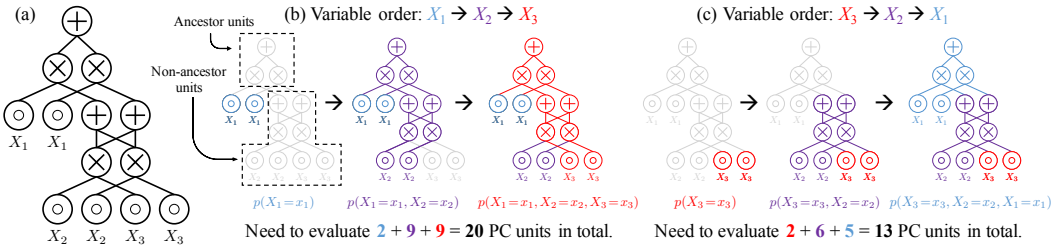

Figure 1: Good variable orders lead to more efficient computation of $F_\pi(\boldsymbol{x})$. Consider the PC p shown in (a). (b): If variable order X_1, X_2, X_3 is used, we need to evaluate 20 PC units in total. (c): The optimal variable order X_3, X_2, X_1 allows us to compute $F_\pi(\boldsymbol{x})$ by only evaluating 13 PC units.

of two marginals, it is equivalent to compute the following two sets of marginal probabilities: $F_\pi(\boldsymbol{x}) := \{p(x_{\pi_1}, \ldots, x_{\pi_i})\}_{i=1}^D$ and $G_\pi(\boldsymbol{x}) := \{p(x_{\pi_1}, \ldots, x_{\pi_{i-1}}, X_{\pi_i} < x_{\pi_i})\}_{i=1}^D$.

As a direct application of the marginal algorithm described in Sec. 3.1, for every $\boldsymbol{x} \in \mathsf{val}(\mathbf{X})$ and π, computing the $2D$ marginals $\{F_\pi(\boldsymbol{x}), G_\pi(\boldsymbol{x})\}$ take $\mathcal{O}(D \cdot |p|)$ time. However, the linear dependency on D could render compression and decompression extremely time-consuming.

Structured-decomposability is the key to overcome this problem. In the following, we introduce an algorithm that computes $F_\pi(\boldsymbol{x})$ and $G_\pi(\boldsymbol{x})$ in $\mathcal{O}(\log(D) \cdot |p|)$ time given a smooth and structured-decomposable PC p. For ease of presentation, we only discuss how to compute $F_\pi(\boldsymbol{x})$, and $G_\pi(\boldsymbol{x})$ can be computed analogously. This is because according to the marginal algorithm introduced in Sec. 3.1, the only difference between computing the ith term in $F_\pi(\boldsymbol{x})$ and $G_\pi(\boldsymbol{x})$ is that different probabilities (i.e., $p_n(X_i = x)$ and $p_n(X_i < x)$) should be assigned to leaf units n correspond to X_i.

The key to speeding up the naive marginalization algorithm is the observation that we only need to evaluate a small fraction of PC units to compute each of the D marginals in $F_\pi(\boldsymbol{x})$. Suppose we want to compute $F_\pi(\boldsymbol{x})$ given the structured-decomposable PC shown in Fig. 1(a), where $\oplus$, $\otimes$, and $\odot$ denote sum, product, and input units, respectively. Model parameters are omitted for simplicity. Consider using the variable order $\pi = (X_1, X_2, X_3)$ (Fig. 1(b)). We ask the following question: what is the minimum set of PC units that need to be evaluated in order to compute $p(X_1 = x_1)$ (the first term in $F_\pi(\boldsymbol{x})$)? First, every PC unit with scope $\{X_1\}$ (i.e., the two nodes colored blue) has to be evaluated. Next, every PC unit n that is not an ancestor of the two blue units (i.e., "non-ancestor units" in Fig. 1(b)) must have probability 1 since (i) leaf units correspond to X_2 and X_3 have probability 1 while computing $p(X_1 = x_1)$, and (ii) for any sum or product unit, if all its children have probability 1, it also has probability 1 following Eq. (2). Therefore, we do not need to evaluate these non-ancestor units. Finally, following all ancestors of the two blue units (i.e., "ancestor units" in Fig. 1(b)), we can compute the probability of the root unit, which is the target quantity $p(X_1 = x_1)$. At a first glance, this seems to suggest that we need to evaluate these ancestor units explicitly. Fortunately, as we will proceed to show, the root unit's probability can be equivalently computed using the blue units' probabilities weighted by a set of cached *top-down probabilities*.

For ease of presentation, denote the two blue input units as n_1 and n_2, respectively. A key observation is that the probability of every ancestor unit of $\{n_1, n_2\}$ (including the root unit) can be represented as a weighted mixture over $p_{n_1}(\boldsymbol{x})$ and $p_{n_2}(\boldsymbol{x})$, the probabilities assigned to n_1 and n_2, respectively. The reason is that since the PC is decomposable, for every ancestor product unit m, we have that n_1 and n_2 should be descendants of a same child unit of m, and thus we shall never multiply $p_{n_1}(\boldsymbol{x})$ and $p_{n_2}(\boldsymbol{x})$. Following the above intuition, the top-down probability $p_{\text{down}}(n)$ of PC unit n is designed to represent the "weight" of n w.r.t. the probability of the root unit. Formally, $p_{\text{down}}(n)$ is defined as the sum of the probabilities of every path from n to the root unit n_r, where the probability of a path is the product of all edge parameters traversed by it. Back to our example, using the top-down probabilities, we can compute $p(X_1 = x_1) = \sum_{i=1}^2 p_{\text{down}}(n_i) \cdot p_{n_i}(x_1)$ without explicitly evaluating the ancestors of n_1 and n_2. The quantity $p_{\text{down}}(n)$ of all PC units n can be computed by Alg. 1 in $\mathcal{O}(|p|)$ time. Specifically, the algorithm performs a top-down traversal over all PC units n, and updates the top-down probabilities of their children $\mathsf{in}(n)$ along the process.

Therefore, we only need to compute the two PC units with scope $\{X_1\}$ in order to calculate $p(X_1 = x_1)$. Next, when computing the second term $p(X_1 = x_1, X_2 = x_2)$, as illustrated in Fig. 1(b), we can reuse the evaluated probabilities of n_1 and n_2, and similarly only need to evaluate the PC units with scope $\{X_2\}$, $\{X_2, X_3\}$, or $\{X_1, X_2, X_3\}$ (i.e., nodes colored purple). The same scheme can be

Algorithm 1 PC Top-down Probabilities

Input: A smooth and structured-decomposable PC p
Output: The top-down probabilities $p_{\text{down}}(n)$ of all PC units n
For every PC unit n in p, initialize $p_{\text{down}}(n) \leftarrow 0$
foreach unit n traversed in preorder (parent before children) **do**
  if n is the root node of p **then** $p_{\text{down}}(n) \leftarrow 1$
  elif n is a sum unit **then foreach** $c \in \mathsf{in}(n)$ **do** $p_{\text{down}}(c) \leftarrow p_{\text{down}}(c) + p_{\text{down}}(n) \cdot \theta_{n,c}$
  elif n is a product unit **then foreach** $c \in \mathsf{in}(n)$ **do** $p_{\text{down}}(c) \leftarrow p_{\text{down}}(c) + p_{\text{down}}(n)$

Algorithm 2 Compute $F_\pi(\boldsymbol{x})$

1: **Input:** A smooth and structured-decomposable PC p, variable order π, variable instantiation $\boldsymbol{x}$
2: **Output:** $F_\pi(\boldsymbol{x}) = \{p(x_{\pi_1}, \ldots, x_{\pi_i})\}_{i=1}^{D}$
3: **Initialize:** The probability $p(n)$ of every unit n is initially set to 1
4: $p_{\text{down}} \leftarrow$ the top-down probability of every PC unit$n$ (i.e., Alg. 1)
5: **for** $i = 1$ **to** D **do** # Compute the ith term in $F_\pi(\boldsymbol{x})$: $p(x_{\pi_1}, \ldots, x_{\pi_i})$
6: $\text{eval}_i \leftarrow$ the set of PC units n with scopes $\phi(n)$ that satisfy at least one of the following conditions:
  (i) $\phi(n) = \{X_{\pi_i}\}$; (ii) n is a sum unit and at least one child c of n needs evaluation, i.e., $c \in \text{eval}_i$;
  (iii) n is a product unit and $X_{\pi_i} \in \phi(n)$ and $\nexists c \in \mathsf{in}(n)$ such that $\{X_{\pi_j}\}_{j=1}^{i} \in \phi(c)$
7: Evaluate PC units in eval_i in a bottom-up manner to compute $\{p_n(\boldsymbol{x}) : n \in \text{eval}_i\}$
8: $\text{head}_i \leftarrow$ the set of PC units in eval_i such that none of their parents are in eval_i
9: $p(x_{\pi_1}, \ldots, x_{\pi_i}) \leftarrow \sum_{n \in \text{head}_i} p_{\text{down}}(n) \cdot p_n(\boldsymbol{x})$

used when computing the third term, and we only evaluate PC units with scope $\{X_3\}$, $\{X_2, X_3\}$, or $\{X_1, X_2, X_3\}$ (i.e., all red nodes). As a result, we only evaluate 20 PC units in total, compared to $3 \cdot |p| = 39$ units required by the naive approach.

This procedure is formalized in Alg. 2. In the main loop (lines 5-9), the D terms in $F_\pi(\boldsymbol{x})$ are computed one-by-one. While computing each term, we first find the PC units that need to be evaluated (line 6).[4]After computing their probabilities in a bottom-up manner (line 7), we additionally use the pre-computed top-down probabilities to obtain the target marginal probability (lines 8-9).

The previous example demonstrates that even without a careful choice of variable order, we can significantly lower the computation cost by only evaluating the necessary PC units. We now show that with an *optimal* choice of variable order (denoted π^*), the cost can be further reduced. Consider using order $\pi^* = (X_3, X_2, X_1)$, as shown in Fig. 1(c), we only need to evaluate $2+6+5 = 13$ PC units in total when running Alg. 2. This optimal variable order is the key to guaranteeing $\mathcal{O}(\log(D) \cdot |p|)$ computation time. In the following, we first give a technical assumption and then proceed to justify the *correctness* and *efficiency* of Alg. 2 when using the optimal variable order π^*.

Definition 5. For a smooth and SD PC p defined on D variables, for any scope ϕ, denote $\text{nodes}(p, \phi)$ as the set of PC units in p whose scope is ϕ. We say p is *balanced* if for every scope ϕ' that is equal to the scope of any unit n in p, we have $|\text{nodes}(p, \phi')| = \mathcal{O}(|p|/D)$.

Theorem 1. *For a smooth, SD, and balanced PC p defined on D variables $\mathbf{X}$ and a sample $\boldsymbol{x}$, there exists a variable order π^*, s.t. Alg. 2 correctly computes $F_{\pi^*}(\boldsymbol{x})$ in $\mathcal{O}(\log(D) \cdot |p|)$ time.*

Proof. The high-level idea of the proof is to first show how to compute the optimal variable order π^* for any smooth and structured-decomposable PC. Next, we justify the correctness of Alg. 2 by showing (i) we only need to evaluate units that satisfy the criterion in line 6 and (ii) weighting the PC units with the top-down probabilities always give the correct marginal probability. Finally, we use induction (on the number of variables D) to demonstrate Alg. 2 computes $\mathcal{O}(\log(D) \cdot |p|)$ PC units in total if π^* is used. Please refer to Appx. A.1 for details. □

While Def. 5 may seem restrictive at a first glance, we highlight that most existing PC structures such as EiNets (Peharz et al., 2020a), RAT-SPNs (Peharz et al., 2020b) and HCLTs (Sec. 4.1) are balanced (see Appx. A.2 for justifications). Once all marginal probabilities are calculated, samples $\boldsymbol{x}$ can be en- or decoded autoregressively with any streaming codes in time $\mathcal{O}(\log(D) \cdot |p|)$. Specifically, our implementation adopted the widely used streaming code rANS (Duda, 2013).

[4]The three conditions in line 6 formalize the intuitions provided in the example Fig. 1(b); see Appx. A.1.

Table 2: Efficiency and optimality of the (de)compressor. The compression (resp. decompression) time are the total computation time used to compress (resp. decompress) all 10,000 MNIST test samples on a single TITAN RTX GPU. The proposed (de)compressor for SD PCs is 5-20x faster than the (de)compressor of IDF and BitSwap and only leads to negligible increase in the codeword bpd compared to the theoretical bpd. HCLT is a PC model that will be introduced in Sec. 4.1.

Method	# parameters	Theoretical bpd	Codeword bpd	Comp. time (s)	Decomp. time (s)
PC (HCLT, $M=16$)	3.3M	1.26	1.30	15	44
PC (HCLT, $M=24$)	5.1M	1.22	1.26	26	89
PC (HCLT, $M=32$)	7.0M	1.20	1.24	44	155
IDF	24.1M	1.90	1.96	288	592
BitSwap	2.8M	1.27	1.31	578	326

3.3 Empirical Evaluation

We compare the proposed algorithm with competitive Flow-model-based (IDF by Hoogeboom et al. (2019)) and VAE-based (BitSwap by Kingma et al. (2019)) neural compression algorithms using the MNIST dataset. We first evaluate bitrates. As shown in Table 2, the PC (de)compressor achieved compression rates close to its theoretical rate estimate — codeword bpds only have ~ 0.04 loss w.r.t. the corresponding theoretical bpds. We note that PC and IDF have an additional advantage: their reported bitrates were achieved while compressing one sample at a time; however, BitSwap needs to compress sequences of 100 samples to achieve 1.31 codeword bpd (Kingma et al., 2019).

Next, we focus on efficiency. While achieving a better codeword bpd (i.e., 1.30) compared to IDF and BitSwap, a relatively small PC model (i.e., HCLT, $M = 16$) encodes (resp. decodes) images 20x (resp. 10x) faster than both baselines.[5] Furthermore, a bigger PC model ($M = 32$) with 7M parameters achieved codeword bpd 1.24, and is still 5x faster in terms of en- and decoding.

Related work As hinted by Sec. 2, we seek to directly learn probability distributions $p(\mathbf{X})$ that are expressive and support tractable (de)compression. In contrast, existing Flow-based (van den Berg et al., 2020; Zhang et al., 2021) and VAE-based (Townsend et al., 2019; Kingma et al., 2019; Ho et al., 2019) neural lossless compression algorithms are based on an orthogonal idea: they adopt simple (oftentimes fully factorized) distributions over a latent space $\mathbf{Z}$ to ensure the tractability of encoding and decoding latent codes $\boldsymbol{z}$, and learn expressive neural networks that "transmit" probability mass from $\mathbf{Z}$ to the feature space $\mathbf{X}$ to compress samples $\boldsymbol{x}$ indirectly. We note that both ideas can be *integrated* naturally: the simple latent distributions used by existing neural compression algorithms can be replaced by expressive PC models. We will further explore this idea in Sec. 5.

4 Scaling Up Learning and Inference of PCs

Being equipped with an efficient (de)compressor, our next goal is to learn PC models that achieve good generative performance on various datasets. Although recent breakthroughs have led to PCs that can generate CelebA and SVHN images (Peharz et al., 2020a), PCs have not been shown to have competitive (normalized) likelihoods on image datasets, which directly influence compression rates. In this section, we show that Hidden Chow-Liu Tree (HCLT) (Liu & Van den Broeck, 2021), a PC model initially proposed for simple density estimation tasks containing binary features, can be scaled up to achieve state-of-the-art performance on various image datasets. In the following, we first introduce HCLTs and demonstrate how to scale up their learning and inference (for compression) in Sec. 4.1, before providing empirical evidence in Sec. 4.2.

4.1 Hidden Chow-Liu Trees

Hidden Chow-Liu Trees (HCLTs) are smooth and structured-decomposable PCs that combine the ability of Chow-Liu Trees (CLTs) (Chow & Liu, 1968) to capture feature correlations and the extra expressive power provided by hidden variable models. Every HCLT can be equivalently represented as a Probabilistic Graphical Model (PGM) (Koller & Friedman, 2009) with hidden variables. Specifically, Fig. 2(a)-(c) demonstrate how to construct the PGM representation of an

[5] The HCLT architecture will be introduced in Sec. 4.1; see Appx. B.4 for more details about the experiments.

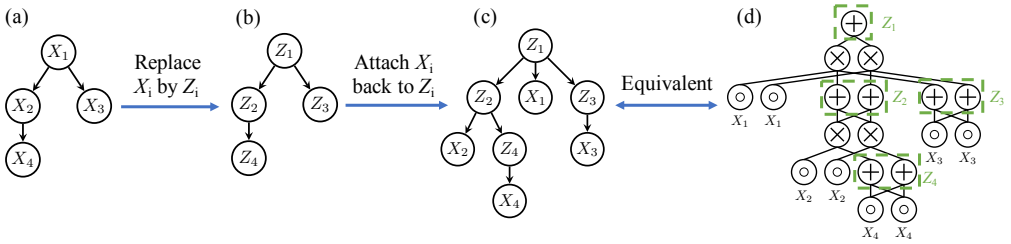

Figure 2: An example of constructing a HCLT PC given a dataset $\mathcal{D}$ with 4 features. (a): Construct the Chow-Liu Tree over variables $X_1, \ldots, X_4$ using $\mathcal{D}$. (b): Replace every variable X_i by its corresponding latent variable Z_i. (c): Attach all X_i back to their respective latent variables Z_i. (d): This PGM representation of HCLT is compiled into an equivalent PC.

Table 3: Compression performance of PCs on MNIST, FashionMNIST, and EMNIST in bits-per-dimension (bpd). For all neural compression algorithms, numbers in parentheses represent the corresponding theoretical bpd (i.e., models' test-set likelihood in bpd).

Dataset	HCLT (ours)	IDF	BitSwap	BB-ANS	JPEG2000	WebP
MNIST	**1.24** (1.20)	1.96 (1.90)	1.31 (1.27)	1.42 (1.39)	3.37	2.09
FashionMNIST	3.37 (3.34)	3.50 (3.47)	**3.35** (3.28)	3.69 (3.66)	3.93	4.62
EMNIST (Letter split)	**1.84** (1.80)	2.02 (1.95)	1.90 (1.84)	2.29 (2.26)	3.62	3.31
EMNIST (ByClass split)	**1.89** (1.85)	2.04 (1.98)	1.91 (1.87)	2.24 (2.23)	3.61	3.34

example HCLT. Given a dataset $\mathcal{D}$ containing 4 features $\mathbf{X} = X_1, \ldots, X_4$, we first learn a CLT w.r.t. $\mathbf{X}$ (Fig. 2(a)). To improve expressiveness, hidden variables are added to the CLT by the two following steps: (i) replace observed variables X_i by their corresponding hidden variables Z_i, which are defined to be categorical variables with M (a hyperparameter) categories (Fig. 2(b)); (ii) connect observed variables X_i with the corresponding hidden variables Z_i by directed edges $Z_i \to X_i$. This leads to the PGM representation of the HCLT shown in Fig. 2(c). Finally, we are left with generating a PC that represents an equivalent distribution w.r.t. the PGM in Fig. 2(c), which is detailed in Appx. B.2. Fig. 2(d) illustrates an HCLT that is equivalent to the PGM shown in Fig. 2(c) (with $M = 2$).

Recent advances in scaling up learning and inference of PCs largely rely on the regularity of the PC architectures they used (Peharz et al., 2020a;b) — the layout of the PCs can be easily *vectorized*, allowing them to use well-developed deep learning packages such as PyTorch (Paszke et al., 2019). However, due to the irregular structure of learned CLTs, HCLTs cannot be easily vectorized. To overcome this problem, we implemented customized GPU kernels for parameter learning and marginal query computation (i.e., Alg. 2) based on Juice.jl (Dang et al., 2021), an open-source Julia package. The kernels automatically segment PC units into layers such that the computation in every layer can be fully parallelized. As a result, we can train PCs with millions of parameters in less than an hour and en- or decode samples very efficiently. Implementation details can be found in Appx. B.3.

Related work Finding good PC architectures has been a central topic in the literature (Choi et al., 2020). A recent trend for learning smooth and (structured-)decomposable PCs is to construct large models with pre-defined architecture, which is mainly determined by the variable ordering strategies. For example, RAT-SPNs (Peharz et al., 2020b) and EiNets (Peharz et al., 2020a) use random variable orders, Gens & Pedro (2013) proposes an effective variable ordering for image data, and other works propose data-dependent orderings based on certain information criterion (Rooshenas & Lowd, 2014) or clustering algorithms (Gens & Pedro, 2013). Alternatively, researchers have focused on methods that iteratively grow PC structures to fit better the data (Dang et al., 2020; Liang et al., 2017). However, this approach has only been successfully applied to PCs that are also deterministic, which are generally less expressive than their non-deterministic counterparts (Choi & Darwiche, 2017).

4.2 Empirical Evaluation

Bringing together expressive PCs (i.e., HCLTs) and our (de)compressor, we proceed to evaluate the compression performance of the proposed PC-based compression algorithm. We compare with 5 competitive lossless compression algorithm: JPEG2000 (Christopoulos et al., 2000); WebP; IDF (Hoogeboom et al., 2019), a Flow-based lossless compression algorithm; BitSwap (Kingma et al.,

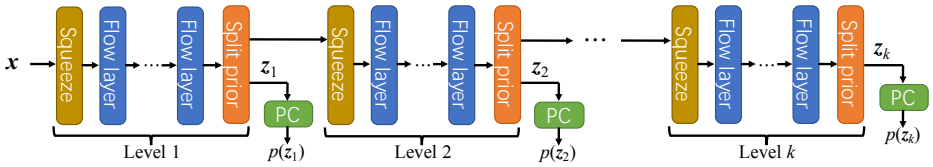

Figure 3: Using PCs as prior distributions of the IDF model (Hoogeboom et al., 2019). PCs are used to represent the k sets of latent variables $\{\boldsymbol{z}_i\}_{i=1}^{k}$.

2019) and BB-ANS (Townsend et al., 2018), two VAE-based lossless compression methods. All 6 methods were tested on four datasets, which include MNIST (Deng, 2012), FashionMNIST (Xiao et al., 2017), and two splits of EMNIST (Cohen et al., 2017). As shown in Table 3, the proposed method out-performed all 5 baselines in 3 out of 4 datasets. On FashionMNIST, where the proposed approach did not achieve state-of-the-art result, it was only 0.02 bpd worse than BitSwap.

5 PCs as Expressive Prior Distributions of Flow Models

As hinted by previous sections, PCs can be naturally integrated with existing neural compression algorithms: the simple latent variables' distributions used by Flow- and VAE-based lossless compression methods can be replaced by more expressive distributions represented by PCs. In this section, we take IDF (Hoogeboom et al., 2019), a Flow-based lossless compression model, as an example to demonstrate the effectiveness of such model integration. IDF was chosen because their authors provided open-source implementation on GitHub. In theory, PC can be integrated with any VAE- and Flow-based model.

Table 4: Theoretical bpd of 5 generative models on three natural image datasets.

Model	CIFAR10	ImageNet32	ImageNet64
RealNVP	3.49	4.28	3.98
Glow	3.35	4.09	3.81
IDF	3.32	4.15	3.90
IDF++	**3.24**	4.10	3.81
PC+IDF	3.28	**3.99**	**3.71**

The integrated model is illustrated in Fig. 3. Following Hoogeboom et al. (2019), an IDF model contains k levels. Each level contains a squeeze layer (Dinh et al., 2016), followed by several integer flow layers and a prior layer. Each level i outputs a set of latent variables $\boldsymbol{z}_i$, which are originally defined as a set of mutually independent discretized logistic variables (Kingma et al., 2016). Instead, we propose to model every set of latent variables $\boldsymbol{z}_i$ with a PC $p(\boldsymbol{z}_i)$. Specifically, we adopted the EiNet codebase (Peharz et al., 2020a) and used a PC structure similar to the one proposed by Gens & Pedro (2013). We adopted the discretized logistic distribution for all leaf units in the PCs. Given a sample $\boldsymbol{x}$, the log-likelihood of the model is the sum of the k PCs' output log-likelihood: $\log p(\boldsymbol{x}) = \sum_{i=1}^{k} \log p(\boldsymbol{z}_i \mid \boldsymbol{x})$. Since both IDF and the PC models are fully differentiable, the PC+IDF model can be trained end-to-end via gradient descent. Details regarding model architecture and parameter learning are provided in Appx. B.5.

We proceed to evaluate the generative performance of the proposed PC+IDF model on 3 natural image datasets: CIFAR10, ImageNet32, and ImageNet64. Results are shown in Table 4. First, compared to 4 baselines (i.e., IDF, IDF++ (van den Berg et al., 2020), Glow (Kingma & Dhariwal, 2018), and RealNVP (Dinh et al., 2016)), PC+IDF achieved the best bpd on ImageNet32 and ImageNet64. Next, PC+IDF improved over its base model IDF by 0.04, 0.16, and 0.19 bpd on three datasets, respectively. This shows the benefit of integrating PCs with IDFs. Although not tested in our experiments, we conjecture that the performance could be further improved by integrating PCs with better Flow-models (e.g., IDF++).

Compression and decompression with the PC+IDF model can be done easily: we can adopt the high-level compression algorithm of IDF and replace the parts of en- or decoding latent variables $\boldsymbol{z}_i$ with the proposed PC (de)compressor. Improving the compression performance of these hybrid models is left for future work.

6 Conclusions

This paper proposes to use Probabilistic Circuits (PCs) for lossless compression. We develop a theoretically-grounded (de)compression algorithm that efficiently encode and decode close to the model's theoretical rate estimate. By scaling up PC models and integrating them with Flow-models, we achieved state-of-the-art compression/generative modeling performance on various image datasets.

Ethics and Reproducibility Statement We are not aware of any ethical concerns of our research. To facilitate reproducibility, we have uploaded our code to the following anonymous link: `https://github.com/pccompression/pccompression`. In addition, we have provided detailed algorithm tables Alg. 1 and 2 for all proposed algorithms, and elaborated each step in detail in the main text (Sec. 3). Formal proofs of all theorems, and details of all experiments (e.g., hardward specifications, hyperparameters) are provided in the appendix.

REFERENCES

Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In *International Conference on Machine Learning*, pp. 825–833. PMLR, 2017.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020.

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

CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi. The jpeg2000 still image coding system: an overview. *IEEE transactions on consumer electronics*, 46(4):1103–1127, 2000.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

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

Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations*, 2016.

Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

Robert Gens and Domingos Pedro. Learning the structure of sum-product networks. In *International conference on machine learning*, pp. 873–880. PMLR, 2013.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13, 1993.

Jonathan Ho, Evan Lohn, and Pieter Abbeel. Compression with flows via local bits-back coding. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 3879–3888, 2019.

Emiel Hoogeboom, Jorn Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 32:12134–12144, 2019.

David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

Diederik P Kingma and Prafulla Dhariwal. 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.

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

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 4743–4751, 2016.

Friso Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*, pp. 3408–3417. PMLR, 2019.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Proceedings of the 14th international conference on principles of knowledge representation and reasoning (KR)*, pp. 1–10, 2014.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

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

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. *arXiv preprint arXiv:2106.02264*, 2021.

Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10629–10638, 2019.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2335–2344, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: an imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 8026–8037, 2019.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pp. 7563–7574. PMLR, 2020a.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pp. 334–344. PMLR, 2020b.

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

Jorma J Rissanen. Generalized kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20(3):198–203, 1976.

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

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 3943–3951. Citeseer, 2016.

James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *International Conference on Learning Representations*, 2018.

James Townsend, Thomas Bird, Julius Kunze, and David Barber. Hilloc: lossless image compression with hierarchical latent variable models. In *International Conference on Learning Representations*, 2019.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *arXiv preprint arXiv:2007.03898*, 2020.

Rianne van den Berg, Alexey A Gritsenko, Mostafa Dehghani, Casper Kaae Sønderby, and Tim Salimans. Idf++: Analyzing and improving integer discrete flows for lossless compression. In *International Conference on Learning Representations*, 2020.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Shifeng Zhang, Chen Zhang, Ning Kang, and Zhenguo Li. ivpf: Numerical invertible volume preserving flow for efficient lossless compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 620–629, 2021.

Zhilin Zheng and Li Sun. Disentangling latent space for vae by label relevant/irrelevant dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12192–12201, 2019.