Probabilistic Integral Circuits

Gennaro Gala[1] Cassio de Campos[1] Robert Peharz[1,2] Antonio Vergari[3] Erik Quaeghebeur[1]

[1]Eindhoven University of Technology, NL
[2]Graz University of Technology, AT
[3]School of Informatics, University of Edinburgh, UK

Abstract

Continuous latent variables (LVs) are a key ingredient of many generative models, as they allow modelling expressive mixtures with an uncountable number of components. In contrast, probabilistic circuits (PCs) are hierarchical discrete mixtures represented as computational graphs composed of input, sum and product units. Unlike continuous LV models, PCs provide tractable inference but are limited to discrete LVs with categorical (i.e. unordered) states. We bridge these model classes by introducing *probabilistic integral circuits* (PICs), a new language of computational graphs that extends PCs with integral units representing continuous LVs. In the first place, PICs are *symbolic* computational graphs and are fully tractable in simple cases where analytical integration is possible. In practice, we parameterise PICs with lightweight neural nets delivering an intractable hierarchical continuous mixture that can be approximated arbitrarily well with large PCs using numerical quadrature. On several distribution estimation benchmarks, we show that such PIC-approximating PCs systematically outperform PCs commonly learned via expectation-maximization or SGD.

1 INTRODUCTION

Many successful probabilistic models build upon *continuous mixtures* (CMs) (Bond-Taylor et al., 2022), which model the data-generating distribution as:

$$p(\mathbf{X}) = \mathbb{E}_{p(\mathbf{Z})}\left[p(\mathbf{X} \mid \mathbf{Z})\right] = \int p(\mathbf{z})\, p(\mathbf{X} \mid \mathbf{z})\, \mathrm{d}\mathbf{z}, \quad (1)$$

where $p(\mathbf{Z})$ is a mixing distribution over *continuous* latent variables (LVs) $\mathbf{Z}$, $p(\mathbf{X} \mid \mathbf{Z})$ is a conditional distribution defining mixture components, and $p(\mathbf{X})$ is the distribution over observables $\mathbf{X}$ obtained by marginalising out $\mathbf{Z}$ from the joint $p(\mathbf{X}, \mathbf{Z}) = p(\mathbf{X} \mid \mathbf{Z})\, p(\mathbf{Z})$.

CMs underpin the success of deep generative models like variational autoencoders (VAEs) (Kingma and Welling, 2014) and generative adversarial networks (Goodfellow et al., 2014). These models are designed such that (i) $p(\mathbf{Z})$ has a simple parametrisation, e.g., an isotropic Gaussian, and (ii) the mixture components $p(\mathbf{X} \mid \mathbf{Z})$ are parameterised by neural networks taking $\mathbf{Z}$ as input. However, the ability of CMs to support tractable probabilistic inference is generally limited. Specifically, *marginalisation* and *conditioning*—which together form a consistent reasoning process (Ghahramani, 2015; Jaynes, 2003)—are largely intractable in these models, mainly due to the generally high-dimensional integral in Equation 1.

In contrast, *Probabilistic Circuits* (PCs) (Vergari et al., 2019; Choi et al., 2020) represent a unified framework for tractable probabilistic models, providing a wide range of exact and efficient inference routines. PCs are computational graphs composed of input, sum and product units, and they represent hierarchical *discrete* mixtures (DMs). This discrete nature is crucial for their tractability, as it allows tractable marginalization of LVs in a single feedforward pass. However, although PCs are deep and structured models encoding an exponential number of mixture components, even a shallow CM has been recently shown to often outperform them (Correia et al., 2023). It is indeed speculated that DMs—like PCs—are hard to learn, while shallow CMs—like VAEs—are easier to learn or generalise better (Liu et al., 2023a). For instance, applying Gaussian Mixture Models, a special case of PCs, to complex, high-dimensional data presents many challenges, such as dealing with sensitivity to initialization (Bender et al., 2023).

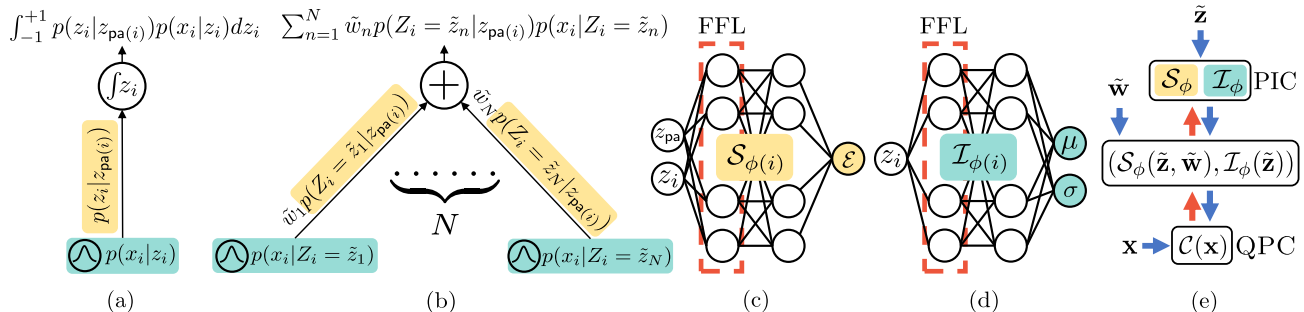In this paper, we show how to incorporate the benefits

Figure 1: **Our contributions.** We extend PCs with symbolic integral units, leading to so-called PICs (Section 3) (a) which we approximate via numerical quadrature as discrete sum units (b). We use neural nets (c) and (d) to parameterise $p(Z_i \mid Z_{\mathsf{pa}(i)})$ and $p(X_i \mid Z_i)$ respectively. In (e), we sketch the training computation graph: We materialise QPC parameters $(\mathcal{S}_\phi(\tilde{\mathbf{z}}, \tilde{\mathbf{w}}), \mathcal{I}_\phi(\tilde{\mathbf{z}}))$ in the forward pass (blue arrows) w.r.t. quadrature points $\tilde{\mathbf{z}}$ and weights $\tilde{\mathbf{w}}$, and propagate gradients in the backward pass (orange arrows) to maximise QPC log-likelihood $\mathcal{C}(\mathbf{x})$.

of CMs into tractable probabilistic models by extending the language of PCs with continuous LVs, which we address with numerical integration. Specifically, we make the following contributions (which are visually summarized in Figure 1):

- We introduce *probabilistic integral circuits* (PICs), a new language of computational graphs that extends PCs with integral units, allowing them to model continuous LVs. Different from PCs, they represent *symbolic* computational graphs and allow tractable inference when analytical integration is possible (Section 3).

- If analytic solutions are not available, e.g., for PICs encoding neural CMs, we design and apply inference routines based on numerical quadrature, allowing to approximate PICs arbitrary well. We call the output of this process QPC (Section 3.1), a PC encoding the **q**uadrature of a deep PIC, thus generalising ideas of Correia et al. (2023).

- We show how to parameterise PICs with lightweight energy-based models, delivering a flexible generative model that we train via maximum-likelihood using quadrature rules (Section 3.2).

- On several distribution estimation benchmarks (Section 4), QPCs systematically outperform traditional PCs commonly learned via EM or SGD.

2 PRELIMINARIES

Before introducing PICs, we recall how to represent hierarchies of LVs as graphical models and review necessary background on circuits. We conclude with a brief overview of numerical quadrature rules, as these will allow approximate inference in PICs.

2.1 Graphical models for CMs

A hierarchy over random variables (RVs) $\mathbf{Y} = \{Y_i\}_{i=1}^D$ can be represented as a Bayesian network (BN), encoded as a DAG with RVs as nodes and parameterized by a set of conditional distributions $p(Y_i \mid \mathsf{pa}(Y_i))$, where $\mathsf{pa}(Y_i)$ is the parent set of Y_i. A BN encodes the joint $p(\mathbf{Y}) = \prod_{i=1}^D p(Y_i \mid \mathsf{pa}(Y_i))$. We will consider some variables $\mathbf{Z} \subset \mathbf{Y}$ to be latent and the remaining variables $\mathbf{X} = \mathbf{Y} \setminus \mathbf{Z}$ observable.

LTMs Marginalizing LVs is a #P-hard task (de Campos et al., 2020), but becomes tractable for discrete BNs when the DAG has bounded treewidth, in particular when it is a tree, yielding the class of *latent tree models* (LTMs, Figure 2) (Choi et al., 2011). However, continuous LTMs allow tractable marginalisation only if the parametric representations of the conditional probabilities allow for analytical integration, like in linear-Gaussian LTMs (Koller and Friedman, 2009).
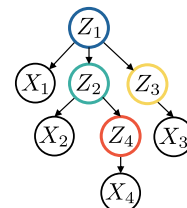

Figure 2: Latent Tree

2.2 Probabilistic Circuits

Circuits A *circuit* $\mathcal{C} \equiv (\mathcal{G}, \theta)$ is a computational graph $\mathcal{G}$ parameterised by θ representing a function $\mathcal{C}(\mathbf{X})$ over its input variables $\mathbf{X}$. $\mathcal{G}$ has three computational unit types: *input* $\bigwedge$, *sum* $\bigoplus$ and *product* $\bigotimes$. An input unit c represents a parameterisable function $f_c(\mathbf{X}_c)$, where $\mathbf{X}_c \subseteq \mathbf{X}$ is its *scope*. Any product or sum unit c receives the outputs of its input units, denoted by the set $\mathrm{IN}(c)$, and has as scope the union of the scopes of its inputs, $\mathbf{X}_c = \cup_{d \in \mathrm{IN}(c)} \mathbf{X}_d$. A product unit c computes the product of its incoming inputs as $\prod_{d \in \mathrm{IN}(c)} d(\mathbf{X}_d)$, whereas a sum unit c com-

putes the weighted sum $\sum_{d\in\text{IN}(c)} \theta_d^c\, d(\mathbf{X}_d)$. Note that $\mathcal{C}(\mathbf{X}) = c(\mathbf{X})$ where c is the circuit root unit. We instantiate a sum or product unit c as $\bigoplus([d]_{d\in\text{IN}(c)}, [\theta_d^c]_{d\in\text{IN}(c)})$ or $\bigotimes([d]_{d\in\text{IN}(c)})$ (cf. Algorithm 1).

Probabilistic Circuits (PCs) A circuit $\mathcal{C}$ is *probabilistic* when $\mathcal{C}(\mathbf{x}) \geq 0$ for any $\mathbf{x}$, i.e. $\mathcal{C}(\mathbf{X})$ is an unnormalized distribution. For efficient renormalization it must satisfy two structural properties: *smoothness* and *decomposability*. A PC is smooth when every sum unit has input units with the same scope, whereas is decomposable when every product unit has input units with disjoint scopes. Furthermore, a PC is *structured-decomposable*—a stricter form of decomposability—when any pair of product units sharing the same scope decomposes the same way, as in Figure 3(b) (Pipatsrisawat and Darwiche, 2008; Vergari et al., 2021). PCs are hierarchical discrete mixtures (Peharz et al., 2017; Zhao et al., 2016; Trapp et al., 2019; Yang et al., 2023), i.e. $\mathcal{C}(\mathbf{X})$ can be written as

$$p(\mathbf{X}) = \sum_{\mathbf{z}} P(\mathbf{z})\, p(\mathbf{X}\mid\mathbf{z}), \qquad (2)$$

where here $\mathbf{Z}$ is a **discrete** latent vector, but otherwise the form is similar to the continuous mixture in Equation 1. The number of states of $\mathbf{Z}$, and thus the number of mixture components $p(\mathbf{X}\mid\mathbf{z})$, grows exponentially in the depth of the PC (Peharz, 2015; Zhao et al., 2016).

Discrete LV interpretation PCs can be compiled from discrete graphical models (Darwiche, 2009) or directly learned from data (Mari et al., 2023; Peharz et al., 2020). In either case sum units can be interpreted as discrete (categorical) LVs (Peharz et al., 2017). As an example, consider the LTM in Figure 2. If we assume its LVs to be categorical, it can be compiled into a smooth and structured-decomposable PC (Butz et al., 2020). Specifically, if we assume *all* its LVs to have $N = 3$ states, the compiled circuit would look like Figure 3(b), where *regions* of $N = 3$ units are arranged hierarchically and sum parameters encode conditional probability tables.

2.3 Numerical Quadrature

As shown by Correia et al. (2023), *numerical quadrature rules* can be used to reduce inference in shallow CMs (Eq. (1)) to shallow DMs (Eq. (2)). In this paper, we deal with *deep* CMs via *recursive* numerical quadrature.

A numerical quadrature rule is an approximation of the definite integral of a function as a weighted sum of function evaluations at specified points within the domain of integration (Davis and Rabinowitz, 1984). Specifically, given some integrand $f : \mathbb{R} \to \mathbb{R}$ and interval $[a, b]$, a quadrature rule consists of a set of N integration points $\tilde{\mathbf{z}} = (\tilde{z}_n)_{n=1}^N \in [a, b]^N$ and weights

$\tilde{\mathbf{w}} = (\tilde{w}_n)_{n=1}^N \in \mathbb{R}^N$ which minimize the integration error $\varepsilon_N = \left| \int_a^b f(z)\,\mathrm{d}z - \sum_{n=1}^N \tilde{w}_n f(\tilde{z}_n)\right|$. Such integration points define sub-intervals over which the integrand is approximated by polynomials. Note that the error ε_N goes to zero as $N \to \infty$ and that $\sum_{n=1}^N \tilde{w}_n = b - a$.

Types of quadratures *Static* quadrature rules like midpoint, trapezoidal and Simpson's achieve error bounds of $\mathcal{O}(N^{-2})$, $\mathcal{O}(N^{-2})$ and $\mathcal{O}(N^{-4})$, respectively. Their choice of $\tilde{\mathbf{z}}$ *exclusively* depends on the finite integration domain. *Gaussian* quadrature rules go a step further as they allow (i) exact integration of any polynomial of degree up to $2N-1$ and (ii) handling infinite integration domains. In particular, Gauss-Hermite quadrature allows to approximate expectations under a Gaussian-distributed variable, but the choice of $\tilde{\mathbf{z}}$ depends on its μ and σ, i.e. it is integrand-dependent. Finally, *adaptive* quadratures approximate the integral of a function using static methods on adaptively refined sub-intervals, until an error tolerance is achieved. The choice of $\tilde{\mathbf{z}}$ is again integrand-dependent, as the refined sub-intervals are those where it is hard to approximate.

3 PROBABILISTIC INTEGRAL CIRCUITS

In this section, we introduce *integral circuits* (ICs), symbolic computational graphs leading to our main contribution, *probabilistic integral circuits* (PICs). The inputs and outputs of every IC unit are considered functions. This is different from PCs, where marginalization of discrete LVs is explicitly and exhaustively represented in the discrete circuit structure (cf. Fig. 1(a-b)).

Integral Circuits (ICs) An *integral circuit* $\mathcal{C} \equiv (\mathcal{G}, \Theta)$ is a *symbolic* computational DAG $\mathcal{G}$ parameterised by Θ that represents a function $\mathcal{C}(\mathbf{X}) = \int \mathcal{C}(\mathbf{X}, \mathbf{z})\mathrm{d}\mathbf{z}$ over input variables $\mathbf{X} = \{X_i\}$ and auxiliary *continuous* variables $\mathbf{Z} = \{Z_i\}$, where $\mathbf{Z} \cap \mathbf{X} = \varnothing$. ICs have four unit types: input $\bigwedge$, integral $\int$, sum $\bigoplus$, and product $\bigotimes$ units. Specifically:

- An input unit c represents a function $f_c(\mathbf{X}_c, \mathbf{Z}_c) \to \mathbb{R}$, where $\mathbf{X}_c \subseteq \mathbf{X}$ and $\mathbf{Z}_c \subseteq \mathbf{Z}$.

- An integral unit c has a function $f_c(\widehat{\mathbf{Z}}_c, \check{\mathbf{Z}}_c) \to \mathbb{R}$ attached, where $(\widehat{\mathbf{Z}}_c, \check{\mathbf{Z}}_c) \subseteq \mathbf{Z}$ and $\check{\mathbf{Z}}_c$ are its integration variables. The unit receives a function $f(\mathbf{X}_d, \widehat{\mathbf{Z}}_d, \check{\mathbf{Z}}_d)$ from its only input unit d, where $\mathbf{X}_d \subseteq \mathbf{X}$, $(\widehat{\mathbf{Z}}_d, \check{\mathbf{Z}}_d) \subseteq \mathbf{Z}$, and $\check{\mathbf{Z}}_d = \check{\mathbf{Z}}_c$, and outputs the function $\int f_c(\widehat{\mathbf{Z}}_c, \check{\mathbf{z}}_c)\, f(\mathbf{X}_d, \widehat{\mathbf{Z}}_d, \check{\mathbf{z}}_d)\,\mathrm{d}\check{\mathbf{z}}_c$.

- A sum (resp. product) unit c outputs a weighted sum (resp. product) of its incoming functions, and therefore a function in turn.

Algorithm 1 PICs → QPCs via *static* quadrature

Input A tree-shaped PIC $\mathcal{C} \equiv (\mathcal{G}, \Theta)$ parameterised by $\{p(X_i \,|\, Z_i)\} \cup \{p(Z_i \,|\, Z_{\mathsf{pa}(i)})\}$, and N_i quadrature points $\tilde{\mathbf{z}}^{(i)}$ with weights $\tilde{\mathbf{w}}^{(i)}$ for each Z_i
Complexity $\mathcal{O}(N^2)$, where $N = \max N_i$

1: $S \leftarrow$ EmptyStackOfRegions()
2: **for** c **in** PostOrderTraversal$(\mathcal{G})$ **do**
3:    **if** c is $\bigotimes(p(X_i \,|\, Z_i))$ **then**
4:        $R \leftarrow \big[\bigotimes \big(p(X_i \,|\, \tilde{z}_j^{(i)})\big)\big]_{j=1}^{N_i}$
5:    **else if** c is $\int z_i$ **then**
6:        $R \leftarrow$ pop(S)
7:        $R \leftarrow \big[\bigoplus \big(R, \big[\tilde{w}_k^{(i)} \, p(\tilde{z}_k^{(i)} \,|\, \tilde{z}_j^{(\mathsf{pa}(i))})\big]_{k=1}^{N_i}\big)\big]_{j=1}^{N_{\mathsf{pa}(i)}}$
8:    **else**
9:        $Q \leftarrow$ pop $|\text{in}(c)|$ regions from S
10:       $R \leftarrow \big[\bigotimes \big([Q[i][j]]_{i=1}^{\text{len}(Q)}\big)\big]_{j=1}^{\text{len}(Q[0])}$
11:   **end if**
12:   push(S, R)
13: **end for**
14: **return** pop$(S)[0]$ ▷ QPC root unit

IC parameters Θ comprise sum unit parameters, and input and integral unit functions f_c. The scope of an input unit c is $\mathbf{X}_c$, and the scope of an inner unit is the union of the scopes of its input units. As such, the structural properties apply to ICs as they do to PCs.

Probabilistic Integral Circuits (PICs) A PIC is a smooth and decomposable IC whose input and integral unit functions f_c are (conditional) probability density functions. As such, PIC auxiliary variables $\mathbf{Z}$ are continuous LVs of the hierarchical mixture the circuit represents.

We will assume that each integral unit has attached (i) a single *distinct* LV Z_i as variable of integration, and (ii) a (conditional) density of the form $p(Z_i \,|\, \mathbf{Z}')$, where $\mathbf{Z}' \subseteq \mathbf{Z} \setminus Z_i$. Using univariate LVs as integration variables allows for straightforward application of standard quadrature rules. We represent an integral unit having integration variable Z_i as $\int z_i$, and we label its incoming edge with $p(Z_i \,|\, \mathbf{Z}')$. An example PIC is shown in Figure 3(a), where input units define conditional probabilities $p(X_i \,|\, Z_i)$, and each integral unit is parameterized by the conditional $p(Z_i \,|\, Z_{\mathsf{pa}(i)})$.

Under these assumptions, a PIC is therefore a *symbolic* computational graph representing a multivariate integral defined by nested univariate integrations. As such, inference involves iteratively integrating out one LV while treating others as constants, which is akin to vari-
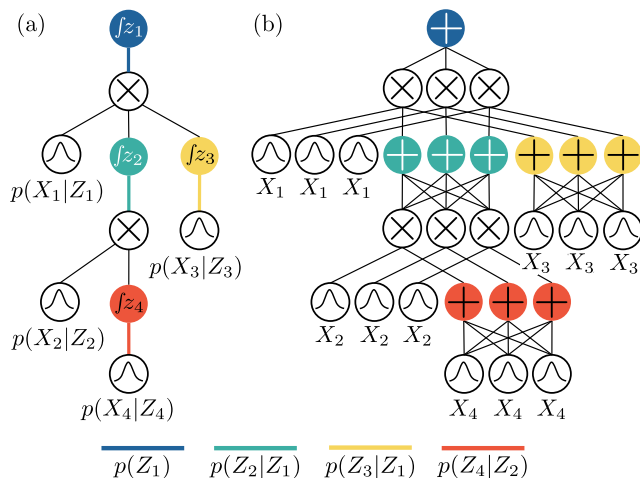

Figure 3: **PICs → QPCs** A tree-shaped PIC in (a) and one of its possible QPC obtained via Algorithm 1 using $N_i = 3$ in (b). Following color-coding, every integral unit in (a) has its incoming edge labelled with one of the conditional densities reported at the bottom. Both circuits are smooth and structured-decomposable.

able elimination (Darwiche, 2009).[1] PICs can in fact describe the operations needed to perform inference with continuous BNs, without *necessarily* specifying how to solve the integrals. Algorithm A.1 shows a possible way to convert an arbitrary BN to a smooth and structured-decomposable PIC via variable elimination, where the resulting PIC is always tree-shaped, regardless the BN structure. For example, converting the LTM in Figure 2 delivers the PIC in Figure 3(a). Such conversion is important, as we can use PIC structures for parameter learning and approximate inference (cf. Section 3.2). Note that, unlike BNs, the conditioning variables of the density attached to an integral unit *cannot* be deduced from the PIC structure.

Tractable PICs Inference in PICs can be tractable, and therefore proceed in an efficient *feedforward* fashion, similarly to standard PCs. However, smoothness and decomposability are not sufficient anymore for tractability as we also need analytical integration to *pass through* integral units. Example of tractable PICs are those representing the computation of Linear Gaussian BNs, i.e. BNs where all RVs are Gaussians and relationships among them are all linear (Bishop, 2006; Koller and Friedman, 2009). The output function of such PICs would then be a multivariate Gaussian, as we show with a step-by-step PIC computation in Appendix C.

---

[1]Note that it may be possible to integrate out more LVs in parallel, like Z_2 and Z_3 in Figure 3(a).

3.1 QPCs approximate intractable PICs

When the computation in integral units cannot be done in analytical form, PICs become intractable. We propose numerical integration as a solution for such intractable scenarios. Specifically, the application of **Q**uadrature rules to approximate intractable PICs, results in standard PCs which we call **Q**PCs. This way, we can *materialise* the symbolic computational graph of a PIC with a QPC that numerically approximates it. Despite QPCs being standard PCs, their LV interpretation is not categorical anymore but rather *ordinal*, as the circuit represents the application of a quadrature rule on a *continuous* intractable model.

Numerically approximating an intractable PIC begins with the application of a quadrature rule over the outermost integral unit (e.g., the root unit for PIC in Figure 3(a)) and then proceeds iteratively with the lower ones. Specifically, for each integration point of an outer integral we would have many for the inner ones, and so on iteratively. Therefore, the selection of integration points plays a crucial role in our numerical approximation.

On the one hand, if for every integration problem involving a specific LV Z_i we *always* use the same static integration points, we can materialise PICs into QPCs with a high re-use of computational units, since same integrand evaluations will be required multiple times. This is the case for QPCs resulting from the application of a *static* quadrature rule. This is detailed in Algorithm 1, where we show how a specific type of tree-shaped PIC can be converted into a QPC compiled in a bottom-up fashion. For instance, if we were to materialise the PIC in Figure 3(a) following Algorithm 1 and using $N = 3$ static integration points for each integral unit, we would obtain the structure in Figure 3(b). Note that applying Algorithm 1 on tree-shaped, smooth and (structured-)decomposable PICs, results in non-tree QPCs with the same structural properties.

On the other hand, if we do not maintain uniformity in our integration point selection, unit re-usage is not possible, and we may need an exponentially big QPC to represent the computation of the resulting numerical approximation. For instance, Gauss-Hermite quadrature chooses the integration points w.r.t. characteristics of the integrand itself, therefore prohibiting re-usage of computational units. Even more dramatically, in cases where the selection of integration points is adaptive, a dependency emerges with the data point **x**. That is because adaptive quadratures can iteratively require more integration points in certain sub-intervals where the density for **X** is relatively difficult to approximate. Consequently, this prevents us from materializing a

circuit that can be arbitrarily good for any **x**.

PICs & LTMs Numerically approximating PICs is particularly effective when the conditional distributions attached to integral units are not conditioned on a large number of LVs. That is especially the case for PICs representing inference of continuous LTMs. Therefore, all the latent conditional distributions of such PIC type are only conditioned on a single LV, except for the root integral unit, which defines a prior. We show how LTMs (cf. Figure 2) can be converted to PICs (cf. Figure 3(a)) in Algorithm A.1. Furthermore, Algorithm A.2 shows how we can generally apply nested univariate quadratures to a tree-shaped PIC.

3.2 Learning neural PICs via quadrature

We here present a methodology to (i) parameterise PICs using light-weight neural energy-based models and (ii) learn the resulting model via numerical integration. It can be applied to every PIC structure, but is particularly suitable for those where LV interactions follow a common pattern. Specifically, we will describe and show its application to PICs compiled from LTMs, since *all* their LVs are in a pair-wise relationship and therefore only involve univariate integrations. So we are interested in approximating integrals of the form $\int_a^b q(x_i, z_i, z_{\mathsf{pa}(i)}) dz_i$, with $q(x, \breve{z}, \widehat{z}) \equiv p(\breve{z} \mid \widehat{z}) p(x \mid \breve{z})$, such that the integration error

$$\varepsilon_N(x, \widehat{z}) = \left| \int_a^b q(x, \breve{z}, \widehat{z}) \mathrm{d}\breve{z} - \sum_{i=1}^N \tilde{w}_n q(x, \tilde{z}_n, \widehat{z}) \right|,$$

goes to zero as $N \to \infty$.

Parameterising integral units Every latent distribution $p(Z_i \mid Z_{\mathsf{pa}(i)})$ attached to integral unit $\left(\int z_i\right)$ is parameterised by a light-weight energy-based model (EBM) (LeCun et al., 2007; Song et al., 2021) $\mathcal{S}_{\phi(i)}$ outputting an energy value for a *conditional* input state $(z_i, z_{\mathsf{pa}(i)})$ of its continuous LVs (cf. Figure 1(c)). To promote the application of static quadrature rules, we will assume that every LV has support $[-1, 1]$, such that pairs $(z_i, z_{\mathsf{pa}(i)})$ belong to $[-1, 1]^2$. Every neural net $\mathcal{S}_{\phi(i)}$ parametrizes a conditional univariate energy-based model:

$$p(z_i \mid z_{\mathsf{pa}(i)}) = \frac{\exp(-\mathcal{S}_{\phi(i)}(z_i, z_{\mathsf{pa}(i)}))}{\int_{-1}^{+1} \exp(-\mathcal{S}_{\phi(i)}(z, z_{\mathsf{pa}(i)})) dz}, \quad (3)$$

where $\mathcal{S}_{\phi(i)} : [-1, 1]^2 \to \mathbb{R}_{\geq 0}$ and the denominator is the normalisation constant for $p(Z_i \mid z_{\mathsf{pa}(i)})$. We constrain the output of $\mathcal{S}_{\phi(i)}$ to be non-negative using a softplus non-linearity, so that the unnormalized density, i.e. the numerator, is bounded by one, as that improves training stability (Nash and Durkan, 2019).

Parameterising input units   We use a similar modelling for input units. Every conditional input distribution $p(X_i \,|\, z_i)$ is parameterised by a neural net $\mathcal{I}_{\phi(i)}$ taking as input a state for its parent, $z_i \in [-1, 1]$, and returning a valid parametrisation of a parametric univariate distribution (cf. Figure 1(d)), similarly to a small VAE decoder (Kingma and Welling, 2014). For instance, if X_i were a Gaussian RV, then:

$$p(X_i \,|\, z) = \mathcal{N}(X_i \,|\, \mu_z, \sigma_z), \qquad (4)$$

where $(\mu_z, \sigma_z) = \mathcal{I}_{\phi(i)}(z)$. Note that, in contrast to standard flow models, we can easily handle heterogeneous features—i.e. both discrete and continuous—by simply designing a specific neural net $\mathcal{I}_{\phi(i)}$ to output a valid parameterisation for the desired X_i input distribution.

Fourier Features   Fourier Feature Layers (FFLs) are an important ingredient for the neural nets $\mathcal{S}_\phi$ and $\mathcal{I}_\phi$. FFLs (Tancik et al., 2020) enable multi-layer perceptrons (MLPs) to learn high-frequency functions in low-dimensional problem domains and are usually used as first layers of coordinate-based MLPs as in Figure 1(c-d). FFLs transform input $\mathbf{x} \in \mathbb{R}^n$ to

$$[\cos(2\pi\mathbf{f}_1^\mathsf{T}\mathbf{x}), \sin(2\pi\mathbf{f}_1^\mathsf{T}\mathbf{x}), \ldots, \cos(2\pi\mathbf{f}_k^\mathsf{T}\mathbf{x}), \sin(2\pi\mathbf{f}_k^\mathsf{T}\mathbf{x})],$$

where k is a hyperparameter and vectors $\mathbf{f}_i \in \mathbb{R}^n$ are non-learnable, randomly initialized parameters.

FFLs have two main benefits: (i) They allow learning more expressive distributions by avoiding oversmoothing, and (ii) they significantly reduce the total count of optimizable parameters when used instead of conventional linear layers as the initial layers in MLPs. We show the effect of using FFLs in Appendix B.2.

Learning via numerical integration   We train neural PICs via *static* numerical integration. Specifically, we optimise neural net parameters ϕ by materialising neural PICs into QPCs via Algorithm 1 and then minimizing the negative log-likelihood under the materialised model at each training step. However, the algorithm cannot be directly applied, as the densities attached to integral units are EBMs $\mathcal{S}_\phi$, for which we need the normalisation constants. Nevertheless, by our design, they can be approximated arbitrarily well via quadrature due to the univariate nature of the density in Equation 3. Therefore, there are two distinct approximations involved, (i) one concerning the normalisation constants for $\mathcal{S}_\phi$, and (ii) another for the integral units. While these approximations can be approached independently, it is computationally cheaper if we use the same quadrature rule for both, especially during training, as the required function evaluations would then be

the same for both. Therefore, this materialisation-by-approximation of QPCs is central for learning PICs, and below we describe how this can be done *efficiently*.

Since the support of $\mathcal{S}_{\phi(i)}$ and $\mathcal{I}_{\phi(i)}$ is $[-1, 1]$, we use the same *static* integration points $\tilde{\mathbf{z}} \in [-1, 1]^N$ and weights $\tilde{\mathbf{w}} \in \mathbb{R}^N$ for every integral unit. Leveraging GPU acceleration, we start by generating the tensor of energy values $\mathcal{E} \in \mathbb{R}^{D \times N \times N}$ *in one shot* as

$$\mathcal{E}_{ijk} = \mathcal{S}_{\phi(i)}(\tilde{z}_k, \tilde{z}_j), \qquad (5)$$

where $i \in [D]$ and $j, k \in [N]$. Then, we use quadrature to approximate the normalisation constants as

$$\mathcal{N}_{ij} = \sum_{k=1}^N \tilde{w}_k \exp(-\mathcal{E}_{ijk}), \qquad (6)$$

which we then use to move from $\mathcal{E}$ to a tensor of approximated probability density values $\mathcal{P}$ as

$$\mathcal{P}_{ijk} = \frac{\exp(-\mathcal{E}_{ijk})}{\mathcal{N}_{ij}} \approx p(Z_i = \tilde{z}_k \,|\, Z_{\mathsf{pa}(i)} = \tilde{z}_j). \quad (7)$$

Note that sums over the third dimension of $\mathcal{P}$ do *not* return 1. Finally, we approximate the integral units w.r.t. the same quadrature rule used in Equation 6. Specifically, we move from $\mathcal{P}$ to a tensor of sum unit parameters as

$$\mathcal{S}_{ijk} = \tilde{w}_k \mathcal{P}_{ijk}, \qquad (8)$$

where $k \in [N]$. Note that sums over the third dimension of $\mathcal{S}$ now return 1. We perform such computation in the log space to ensure numerical stability. Concisely, we denote the materialisation of *all* sum unit parameters $\mathcal{S}$ w.r.t. integration points $\tilde{\mathbf{z}}$ and weights $\tilde{\mathbf{w}}$ as $\mathcal{S}_\phi(\tilde{\mathbf{z}}, \tilde{\mathbf{w}})$. Similarly, we materialise *all* input unit parameters in a tensor $\mathcal{I} \in \mathbb{R}^{D \times N \times I}$ as

$$\mathcal{I}_{ij} = \mathcal{I}_{\phi(i)}(\tilde{z}_j) \in \mathbb{R}^I, \qquad (9)$$

where I is the number of parameters per input unit.[2] Concisely, we denote the materialisation of input unit parameters w.r.t. integration points $\tilde{\mathbf{z}}$ as $\mathcal{I}_\phi(\tilde{\mathbf{z}})$. The training flow is sketched in Figure 1(e).

Under a static quadrature regime, both $\mathcal{S}_{\phi(i)}$ and $\mathcal{I}_{\phi(i)}$ can be run in parallel for every i, delivering a fast materialisation of QPCs. Importantly, the circuit can be materialized for any N, delivering arbitrarily wide QPCs. Furthermore, the neural nets $(\mathcal{S}_\phi, \mathcal{I}_\phi)$ represent a very compact representation for the materialised QPC when a large N is used. That is because the size of $\mathcal{S}_\phi$ and $\mathcal{I}_\phi$ is independent from the N we choose. Moreover, similar to Shih et al. (2021), not only can we be memory-efficient during storage, but we can also be memory-efficient during evaluation. In fact, we materialize QPC regions on the fly, and erase them from

---

[2]We assume, for ease of presentation, that *all* input units of a circuit have the same parametric distribution.
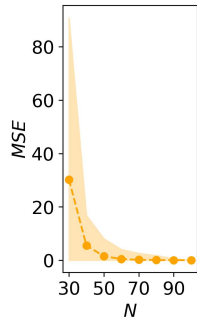
Figure 4: **Increasing the number of integration points reduces the approximation error.** MSE (y-axis) between linear Gaussian LTM log-likelihoods and QPC approximations at increasing number of integration points N (x-axis) averaged over 50 different LTMs with 256 nodes.

memory when they become unnecessary. We conjecture that training via numerical integration sufficiently regularises the neural nets $\mathcal{S}_\phi$ and $\mathcal{I}_\phi$ so that the overall model remains amenable to numerical integration.

4 EXPERIMENTS

4.1 A tractable sanity check

We start off our experiments showing empirically that QPCs can be used to approximate (tractable) continuous LV models. We do this by using linear Gaussian LTMs as tractable ground-truth models and show, as a sanity check, that these can be approximated arbitrarily well with QPCs. Specifically, we convert these LTMs to PICs using Algorithm A.1 and apply Algorithm 1 to approximate them by QPCs. To measure the goodness of our approximation, we generate 1,000 samples from the ground-truth model and then use the MSE between their true log-likelihoods and their QPC approximations. We provide more detail in subsection C.2. Results are sketched in Figure 4.

4.2 Distribution estimation

In the following experiments, we use the latent tree structure provided by the Hidden Chow-Liu Tree algorithm (Liu and Van den Broeck, 2021). These structures are heuristically derived by introducing LVs in classical Chow-Liu Trees (CLTs). Specifically, the structure is built in three steps: (i) first, a tree over input variables $\mathbf{X} = \{X_i\}_{i=1}^D$ is learned via the CLT algorithm (Chow and Liu, 1968), (ii) then, every node X_i in the tree is replaced with LV Z_i, and (iii) finally, every node X_i is reintroduced in the tree as child of Z_i. Assuming LVs to be categorical, these LTMs can be compiled into smooth and structured-decomposable PCs—precisely resembling the one in Figure 3(b)—which are called HCLTs. They demonstrated competitive performance across many different tasks in recent literature (Liu et al., 2022; Liu and Van den Broeck, 2021; Liu et al., 2023a; Dang et al., 2022b; Liu et al., 2023b). Our code is available at github.com/gengala/pic.

4.2.1 MNIST Family

Following Dang et al. (2022a), we evaluate PICs on the MNIST-family image datasets which includes MNIST (LeCun et al., 2010), EMNIST (and its 4 splits) (Cohen et al., 2017), and FashionMNIST (Xiao et al., 2017). We use the average test-set bits-per-dimension $\mathsf{bdp}(\mathcal{C}(\mathbf{x})) = -\log_2(\mathcal{C}(\mathbf{x}))/D$ as the evaluation metric.

Setting Our main reference is the HCLT model (Liu and Van den Broeck, 2021) as applying a static quadrature on PICs results in QPCs having *exactly* the same structure. We ablate using $N = \{16, 32, 64, 128\}$ and batch size $B = \{64, 128, 256\}$. The hyperparameter N refers to the number of integration points when associated to QPCs and to the number of latent categorical states when associated to HCLTs. We train HCLTs using stochastic mini-batch Expectation-Maximization. Both the EM step size (used for HCLTs) and learning rate (used for PICs) are annealed using the Cosine Annealing with Warm Restarts scheduler (Loshchilov and Hutter, 2017) to avoid sensitivity to initialization which often leads to convergence issues in PCs. For a fair comparison, we re-implement HCLTs in pytorch and achieve *better results* than those reported by Liu et al. (2022) and Dang et al. (2022a). We report more training details and experiments in Appendix B.

Results Figure 5 reports the best average test-set bpd for the MNIST-family. Remarkably, we outperform *all* the baselines except SparsePC (Dang et al., 2022a), which, however, uses a more sophisticated structure learner, which iteratively prunes and grows the computational graph of an HCLT.

Basing our comparison on standard HCLTs, our main reference, not only QPCs outperform them in terms of best bpd, but they do that consistently for every B-N configuration as we show in the scatter plot in Figure 5. The same trend occurs when comparing QPCs with HCLTs trained with Adam (Kingma and Ba, 2015) and using binomial input units (cf. Appendix B). QPCs also outperform shallow CMs of PCs (Correia et al., 2023) as they only score 1.28 (resp. 3.55) on MNIST (resp. FASHION-MNIST). Note that when using $N = 128$, PICs have roughly half the parameters of their materialised QPCs.

4.2.2 Penn Tree Bank

Again following Dang et al. (2022a), we use the Penn Tree Bank dataset with standard processing from Mikolov et al. (2012), which contains around 5M characters and a character-level vocabulary size of 50. The data is split into sentences with a maximum sequence length of 288. We compare with four competitive baselines: SparsePC (Dang et al., 2022a), Bipartite flow

	QPC	HCLT	Sp-PC	RAT	IDF	BitS	BBans	McB
MNIST	**1.18**	1.21	<u>1.14</u>	1.67	1.90	1.27	1.39	1.98
F-MNIST	**3.27**	3.34	<u>3.27</u>	4.29	3.47	3.28	3.66	3.72
EMN-MN	**1.66**	1.70	<u>1.52</u>	2.56	2.07	1.88	2.04	2.19
EMN-LE	**1.70**	1.75	<u>1.58</u>	2.73	1.95	1.84	2.26	3.12
EMN-BA	**1.73**	1.78	<u>1.60</u>	2.78	2.15	1.96	2.23	2.88
EMN-BY	**1.67**	1.73	<u>1.54</u>	2.72	1.98	1.87	2.23	3.14
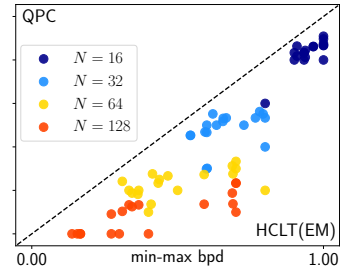
Figure 5: **QPCs systematically outperform PCs trained via EM or SGD.** Table (left): Best average test-set bpd for the MNIST-family datasets. We compare against HCLT (Liu and Van den Broeck, 2021), SparsePC (Dang et al., 2022a) RAT-SPN (Peharz et al., 2020), IDF (Hoogeboom et al., 2019), BitSwap (Kingma et al., 2019), BBans (Townsend et al., 2019) and McBits (Ruan et al., 2021). QPC results are in bold if better than HCLTs, whereas global best results are underlined. QPC and HCLT results are averaged over 5 different runs; the other results are taken from Dang et al. (2022a). Scatter plot (right): bpd results for QPCs (y-axis) and HCLTs (x-axis) paired by B-N hyperparameter configuration and (min-max) normalized for every MNIST-family dataset. A similar trend occurs for binomial input units (cf. Appendix B).

Table 1: Character-level language modeling results on Penn Tree Bank in test-set bpd. Competitor results are taken from Dang et al. (2022a)

QPC	Sp-PC	B-flow	AF/SCF	IAF/SCF
1.22	1.35	1.38	1.46	1.63

(Tran et al., 2019) and latent flows (Ziegler and Rush, 2019) including AF/SCF and IAF/SCF, since they are the only comparable work with non-autoregressive language modeling. Table 1 show that QPCs remarkably outperform all baselines by a considerable margin.

5 DISCUSSION & CONCLUSION

Related Work Like HyperSPN (Shih et al., 2021) and ConditionalSPN (Shao et al., 2020), we materialise circuit parameters using neural nets. However, different from HyperSPN, we can materialise an *arbitrary big* QPCs whose size is independent of the size of the neural nets $(\mathcal{S}_\phi, \mathcal{I}_\phi)$. That is because HyperSPN still works under the categorical LV interpretation, and therefore is constrained by design to output a fixed number of circuit parameters.

Closer to ours, Correia et al. (2023) work under the continuous LV interpretation as we do, and propose a simple CM of PCs, where the parameters of each PC are dependent on a continuous multivariate LV via a learnable function parametrised by a neural net, which resembles a VAE decoder. These models—like PICs—are inherently intractable, but for small latent dimensions, can be accurately approximated via numerical integration methods, which allows learning via regular backpropagation. The numerical approximation eventually yields a discrete mixture of PCs, which

is a PC in itself. However, the application of CMs by Correia et al. (2023) is designed and limited for (large) root units only, and their usage as internal units is not explored.

Model interpretation One can interpret our model in two distinct ways (cf. Correia et al., 2023). The first is to interpret it as a *factory method*, whereby each finite set of LV values (integration points) yields a PC. The second is to take it as an intractable continuous LV model, but one that is amenable to numerical integration. At test time, we are free to change the integration method and the number of integration points if more or less precision is needed. However, do note that increasing the number of integration points at test-time does *not* necessarily results in better log-likelihoods.

Broader Impact The use of continuous LVs in PICs can unlock new learning schemes for PCs, such as variational inference, adversarial and self-supervised learning. These learning schemes are not applicable to PCs directly, as they do *not* provide differentiable sampling. This is because their backward pass is not differentiable, as sum units involve categorical sampling. Recently, Lang et al. (2022) used the Gumbel-Softmax trick (Jang et al., 2017), combined with a smooth children-interpolation to bypass the issue, but with very little practical success. Instead, PICs are more directly compatible with differentiable sampling and therefore amenable to the previously mentioned learning schemes. That is possible if the neural nets $\mathcal{S}_{\phi(i)}$ allow for differentiable sampling. For this work, the neural nets $\mathcal{S}_\phi$ are EBMs, which are incapable of differentiable sampling. However, for instance, replacing EBMs with Normalizing Flows would already be sufficient to allow for differentiable sampling in PICs.

Limitations & Future Work Training PICs requires a little overhead both in space and time (cf. Appendix B.1). For future work, we plan to investigate higher latent dimensionalities, arbitrary PIC structures, differentiable sampling and pruning methods. We also plan to investigate more tractable model like Squared Neural Families (Tsuchida et al., 2023) instead of EBMs. This work shows new means to perform inference with continuous LVs with a possibly complex association structure. It is left open the possibility to learn the better structures, beyond HCLTs. This work may also open research on inference algorithms for models such as continuous Bayesian nets and to extend models such as variational auto-encoders to represent latent structures that are amenable to marginalization via quadrature rules.

Acknowledgments

References

Christopher M. Bender, Yifeng Shi, Marc Niethammer, and Junier Oliva. Continuously parameterized mixture models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 2050–2062, 2023. URL https://proceedings.mlr.press/v202/bender23a.html.

Christopher M. Bishop. *Pattern recognition and machine learning*. Information Science and Statistics. Springer, 2006.

Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, 2022. doi:10.1109/TPAMI.2021.3116668.

Cory Butz, Jhonatan S. Oliveira, and Robert Peharz. Sum-product network decompilation. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 53–64, 2020. URL https://proceedings.mlr.press/v138/butz20a.html.

Myung Jin Choi, Vincent Y. F. Tan, Animashree Anandkumar, and Alan S Willsky. Learning latent tree graphical models. *Journal of Machine Learning Research*, 12(49):1771–1812, 2011. URL https://www.jmlr.org/papers/v12/choi11b.html.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, 2020. URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968. doi:10.1109/TIT.1968.1054142.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: Extending MNIST to handwritten letters. In *IJCNN 2017*, pages 2921–2926, 2017. doi:10.1109/IJCNN.2017.7966217.

Alvaro H. C. Correia, Gennaro Gala, Erik Quaeghebeur, Cassio de Campos, and Robert Peharz. Continuous mixtures of tractable probabilistic models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):7244–7252, 2023. doi:10.1609/aaai.v37i6.25883.

Meihua Dang, Anji Liu, and Guy Van den Broeck. Sparse probabilistic circuits via pruning and growing. In *NeurIPS 2022*, volume 35 of *Advances in Neural Information Processing Systems*, 2022a. URL https://openreview.net/forum?id=KieCChVB6mN.

Meihua Dang, Anji Liu, Xinzhu Wei, Sriram Sankararaman, and Guy Van den Broeck. Tractable and expressive generative models of genetic variation data. In *Research in Computational Molecular Biology*, pages 356–357, 2022b. doi:10.1007/978-3-031-04749-7_26.

Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009. doi:10.1017/CBO9780511811357.

Philip J. Davis and Philip Rabinowitz. *Methods of numerical integration*. Academic Press, 1984. doi:10.1016/C2013-0-10566-1.

Cassio P. de Campos, Georgios Stamoulis, and Dennis Weyland. A structured view on weighted counting with relations to counting, quantum computation and applications. *Information and Computation*, 275:104627, 2020. doi:10.1016/j.ic.2020.104627.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015. doi:10.1038/nature14541.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS 2014*, volume 27 of *Advances in Neural Information Processing Systems*, 2014. doi:10.48550/arXiv.1406.2661.

Emiel Hoogeboom, Jorn Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *NeurIPS 2019*, volume 32 of *Advances in Neural Information Processing Systems*, 2019. doi:10.48550/arXiv.1905.07376.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *ICLR 2017*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Edwin T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003. doi:10.1017/CBO9780511790423.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR 2015*, 2015. doi:10.48550/arXiv.1412.6980.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR 2014*, 2014. doi:10.48550/arXiv.1312.6114.

Friso Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3408–3417, 2019. URL https://proceedings.mlr.press/v97/kingma19a.html.

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

Steven Lang, Martin Mundt, Fabrizio Ventola, Robert Peharz, and Kristian Kersting. Elevating perceptual sample quality in pcs through differentiable sampling. In *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*, volume 181 of *Proceedings of Machine Learning Research*, pages 1–25, 2022. URL https://proceedings.mlr.press/v181/lang22a.html.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. In *Predicting structured data*, Neural Information Processing. MIT Press, 2007. URL http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf.

Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST database of handwritten digits, 2010. URL http://yann.lecun.com/exdb/mnist.

Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. In *NeurIPS 2021*, volume 34 of *Advances in Neural Information Processing Systems*, pages 3558–3570, 2021. doi:10.48550/arXiv.2106.02264.

Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. In *ICLR 2022*, 2022. URL https://openreview.net/forum?id=X_hByk2-5je.

Anji Liu, Honghua Zhang, and Guy Van den Broeck. Scaling up probabilistic circuits by latent variable distillation. In *ICLR 2023*, 2023a. URL https://openreview.net/forum?id=067CGykiZTS.

Xuejie Liu, Anji Liu, Guy Van den Broeck, and Yitao Liang. Understanding the distillation process from deep generative models to tractable probabilistic circuits. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21825–21838, 2023b. URL https://proceedings.mlr.press/v202/liu23x.html.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *ICLR 2017*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Antonio Mari, Gennaro Vessio, and Antonio Vergari. Unifying and understanding overparameterized circuit representations via low-rank tensor decompositions. In *The 6th Workshop on Tractable Probabilistic Modeling*, 2023. URL https://openreview.net/forum?id=1btutFdIya.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Černocký. Subword language modeling with neural networks, 2012. URL http://www.fit.vutbr.cz/~imikolov/rnnlm/char.pdf.

Charlie Nash and Conor Durkan. Autoregressive energy machines. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1735–1744, 2019. URL https://proceedings.mlr.press/v97/durkan19a.html.

Robert Peharz. *Foundations of sum-product networks for probabilistic modeling*. 2015. doi:10.3217/rz004-x6g87. PhD thesis, Graz University of Technology.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(10):2030–2044, 2017. doi:10.1109/TPAMI.2016.2618381.

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 *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344, 2020. URL https://proceedings.mlr.press/v115/peharz20a.html.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08)*, volume 1, pages 517–522, 2008.

Yangjun Ruan, Karen Ullrich, Daniel S. Severo, James Townsend, Ashish Khisti, Arnaud Doucet, Alireza Makhzani, and Chris Maddison. Improving lossless compression rates via monte carlo bits-back coding. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9136–9147, 2021. URL https://proceedings.mlr.press/v139/ruan21a.html.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 401–412. PMLR, 2020. URL https://proceedings.mlr.press/v138/shao20a.html.

Andy Shih, Dorsa Sadigh, and Stefano Ermon. Hyperspns: Compact and expressive probabilistic circuits. In *NeurIPS 2021*, volume 34 of *Advances in Neural Information Processing Systems*, pages 8571–8582, 2021. doi:10.48550/arXiv.2112.00914.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR 2021*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2nd edition, 2023. doi:10.1017/9781108917407.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *NeurIPS 2020*, volume 33 of *Advances in Neural Information Processing Systems*, pages 7537–7547, 2020. doi:10.48550/arXiv.2006.10739.

James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *ICLR 2019*, 2019. URL https://openreview.net/forum?id=ryE98iR5tm.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. Discrete flows: Invertible generative models of discrete data. In *NeurIPS 2019*, volume 32 of *Advances in Neural Information Processing Systems*, 2019. doi:10.48550/arXiv.1905.10347.

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *NeurIPS 2019*, volume 32 of *Advances in Neural Information Processing Systems*, 2019. doi:10.48550/arXiv.1905.10884.

Russell Tsuchida, Cheng Soon Ong, and Dino Sejdinovic. Squared neural families: A new class of tractable density models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=FDzQQTPqEJ.

Antonio Vergari, Nicola Di Mauro, and Guy Van den Broeck. Tractable probabilistic models: Representations, algorithms, learning, and applications, 2019. URL http://web.cs.ucla.edu/~guyvdb/slides/TPMTutorialUAI19.pdf. Tutorial at the 35th Conference on Uncertainty in Artificial Intelligence (UAI 2019).

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *NeurIPS 2021*, volume 36 of *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=9SD2Rb3NiWu.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, art. 1708.07747, 2017. doi:10.48550/arXiv.1708.07747.

Yang Yang, Gennaro Gala, and Robert Peharz. Bayesian structure scores for probabilistic circuits. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 563–575, 2023. URL https://proceedings.mlr.press/v206/yang23a.html.

Han Zhao, Pascal Poupart, and Geoffrey J Gordon. A unified approach for learning the parameters of sum-product networks. In *NIPS 2016*, volume 29 of *Advances in Neural Information Processing Systems*, 2016. doi:10.48550/arXiv.1601.00318.

Zachary Ziegler and Alexander Rush. Latent normalizing flows for discrete sequences. In *Proceedings of the 36th International Conference on Machine Learning*,

volume 97, pages 7673–7682, 2019. URL https://proceedings.mlr.press/v97/ziegler19a.html.

Checklist

1. For all models and algorithms presented, check if you include:

   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]

   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]

   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:

   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]

   (b) Complete proofs of all theoretical results. [Not Applicable]

   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:

   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]

   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]

   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]

   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

   (a) Citations of the creator If your work uses existing assets. [Yes]

   (b) The license information of the assets, if applicable. [Yes]

   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

   (d) Information about consent from data providers/curators. [Not Applicable]

   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

   (a) The full text of instructions given to participants and screenshots. [Not Applicable]

   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A   ALGORITHMS

Algorithm A.1 shows how to convert an arbitrary Bayesian network with continuous LVs $\mathbf{Z}$ into a tree-shaped, smooth and structured-decomposable PIC via variable elimination (Darwiche, 2009). Figure C.5 shows the result of such algorithm when applied to a latent tree structure.

Algorithm A.1 BN2PIC via latent variable elimination

Input: A Bayesian Network over observables $\mathbf{X} = \{X_i\}_{i=1}^{D}$ and continuous latents $\mathbf{Z} = \{Z_i\}_{i=1}^{D'}$
Output: A tree-shaped, smooth and structured-decomposable PIC

1: $C \leftarrow \text{SETOFUNITS}\big(\bigotimes\!\big(\big[p(X_i \mid \mathsf{pa}(X_i))\big]\big)_{i=1}^{D}\big)$ ▷ Initialise set of units with input units
2: **for** Z_i **in** $\text{TOLIST}(\mathbf{Z})$ **do**   ▷ Any order of $\mathbf{Z}$ is fine
3:      $D \leftarrow$ retrieve all units in C whose output functions include Z_i
4:      $C \leftarrow$ replace units D with integral unit $\widecircled{\int z_i}\big(\big[\bigotimes(D)\big]\big)$ having $p(Z_i \mid \mathsf{pa}(Z_i))$ as attached density
5: **end for**
6: $c \leftarrow \bigotimes(C)$
7: $c \leftarrow \text{OPTIONALPRUNING}(c)$
8: **return** c ▷ PIC root unit

Algorithm A.2 shows how to apply arbitrary quadrature rules (e.g. integrand-dependent) to approximate a PIC into a QPC whose size is exponential in the PIC depth. Figure C.6 shows the QPC resulting from the application of Algorithm A.2 on the PIC in Figure C.5. Note that our neural PICs (cf. Section 3.2) are always materialised via Algorithm 1, therefore always delivering QPCs whose sizes are quadratic in the number of integration points N.

Algorithm A.2 PIC $\rightarrow$ QPC via arbitrary nested univariate quadratures

---

Input: A tree-shaped PIC $\mathcal{C}$ parameterised by $\{p(X_i \mid Z_i)\} \cup \{p(Z_i \mid Z_{\mathsf{pa}(i)})\}$ with no sum units
Output: A QPC representing an approximation of the input PIC via nested univariate quadratures

1: $c = \text{QUAD}(\text{GETROOT}(\mathcal{C}), \text{NULL})$ ▷ PIC integral root unit has no conditioning RVs, thus $\tilde{z} = \text{NULL}$
2: **return** c ▷ return QPC sum root unit
3: **function** $\text{QUAD}(\widecircled{\int z_i}, \tilde{z})$
4:      $\tilde{\mathbf{z}}, \tilde{\mathbf{w}} \leftarrow \text{QUADPOINTSANDWEIGHTS}\big(p(Z_i \mid Z_{\mathsf{pa}(i)} = \tilde{z})\big)$ ▷ $(\tilde{\mathbf{z}}, \tilde{\mathbf{w}})$ *can* depend on the integrand
5:      $R \leftarrow \big[\bigotimes\big(p(X_i \mid Z_i = \tilde{z}_n)\big)\big]_{n=1}^{\text{LEN}(\tilde{\mathbf{z}})}$
6:      **if** $\widecircled{\int z_i}$ has a product input unit **then**
7:          $C \leftarrow \text{GETINPUTINTEGRALUNITS}(\text{GETPRODINPUTUNIT}(\widecircled{\int z_i}))$
8:          $R \leftarrow \big[\bigotimes\big([R[n]] \cup [\text{QUAD}(c, \tilde{z}_n)]_{c \in C}\big)\big]_{n=1}^{\text{LEN}(\tilde{\mathbf{z}})}$ ▷ *all* $\text{QUAD}(\cdot)$ calls can run in parallel
9:      **end if**
10:      **return** $\bigoplus\big(R, \big[\tilde{w}_n\, p(Z_i = \tilde{z}_n \mid Z_{\mathsf{pa}(i)} = \tilde{z})\big]_{n=1}^{\text{LEN}(\tilde{\mathbf{z}})}\big)$
11: **end function**

B   EXPERIMENTS

Training   We train PICs (resp. HCLTs) for a maximum of 30K training steps using a batch size of $B \in \{64, 128, 256\}$ and number of integration points (resp. categorical latent dimension) of $N \in \{16, 32, 64, 128\}$. Adam learning rate and EM step size are both annealed from 10^{-2} to 10^{-4} using the *cosine annealing with warm restarts* scheduler over a range of 500 training steps (Loshchilov and Hutter, 2017). We use early stopping: Training stops when the average validation log-likelihood does not improve after 1250 training steps. We train PICs under the trapezoidal quadrature rule. We use a single NVIDIA A100-SXM4-40GiB for all our experiments.
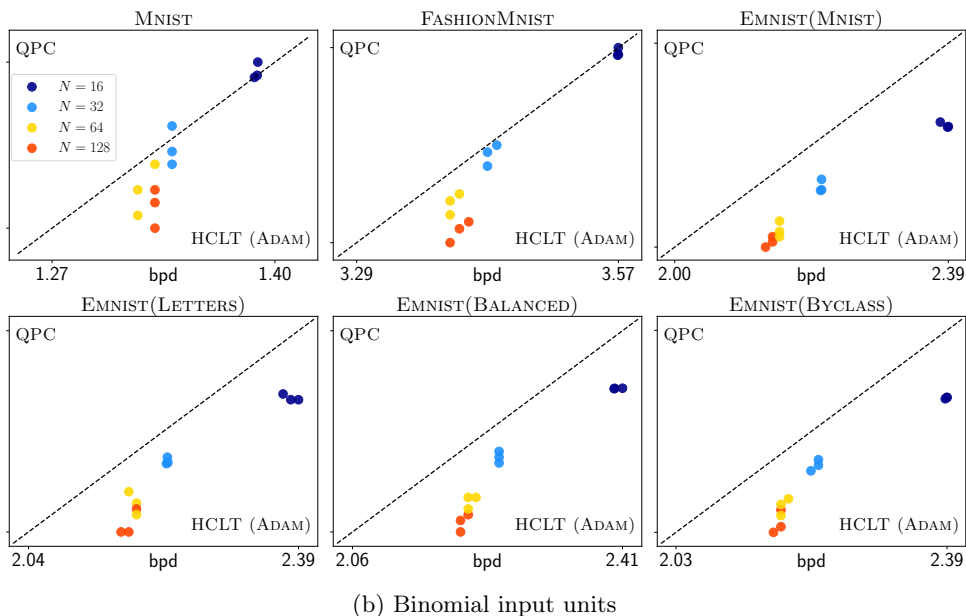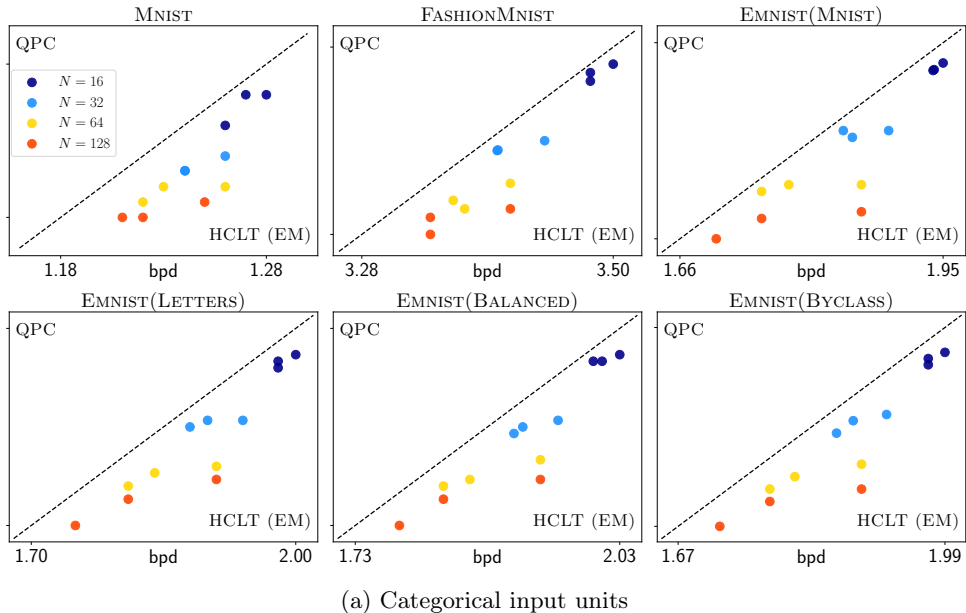

(a) Categorical input units

(b) Binomial input units

Figure B.1: **QPCs outperform HCLTS.** Average test-set bits-per-dimension (bpd) of QPCs (y-axis) and HCLTs (x-axis) paired by $N \in \{16, 32, 64, 128\}$ and batch size $B \in \{64, 128, 256\}$, for categorical (a) and binomial (b) input units. In (a) HCLTs are trained via Expectation-Maximisation, whereas in (b) using Adam. QPCs outperfom HCLTs in almost all hyperparameter configurations.

B.1 Time & Space

In Figure B.2, we report the time (in seconds) and space (in GiB) required to process MNIST mini-batches when *training* under different learning schemes. Although PIC number of parameters can be significantly smaller than the number of materialised QPC parameters, training PICs via numerical integration (cf. Section 3.2) adds little time-space overhead due to the materialisation of QPC parameters at each optimisation step. Instead, at test-time, being QPCs standard PCs, they require the same inference time and space of standard HCLTs, if materialised using Algorithm 1.
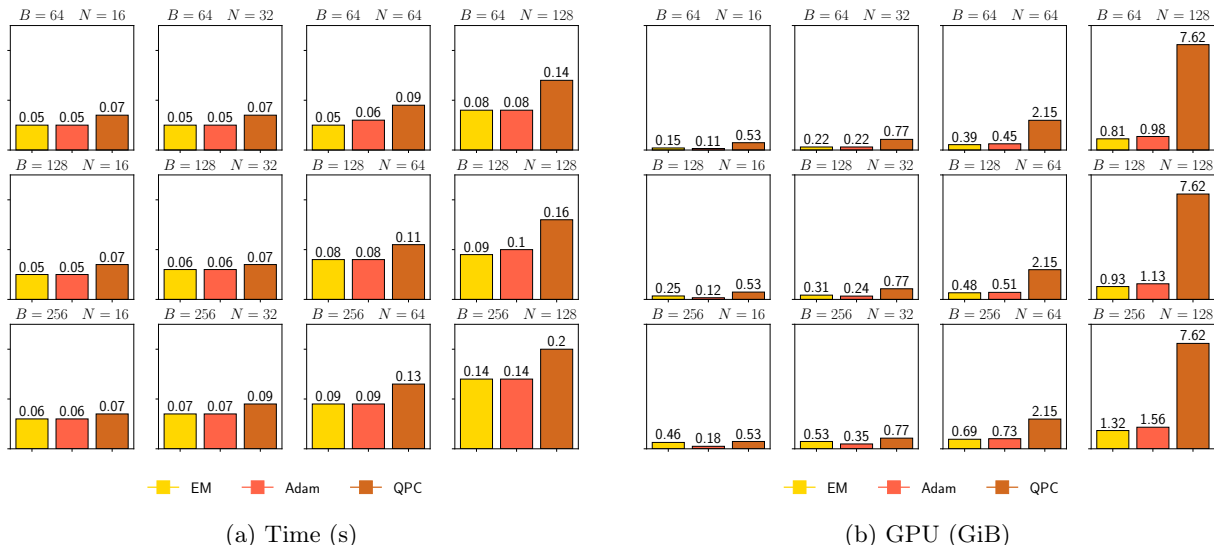

(a) Time (s)   (b) GPU (GiB)

Figure B.2: **Training PICs via numerical integration adds little time-space overhead.** We report time (a) and allocated GPU-memory peak (b) needed to process MNIST mini-batches of size $B \in \{64, 128, 256\}$ at different $N \in \{16, 32, 64, 128\}$ for HCLTs trained via EM (yellow) and Adam (orange), and QPCs (brown). All PC models have categorical input units.

B.2 Fourier Features vs No Fourier Features

Fourier Feature Layers (Tancik et al., 2020) are essential to get competitive performance with PICs. In fact, when using simple multi-layer perceptrons (MLPs) without FFLs, materialised QPCs with categorical input units could not score better than 1.27 bpd on MNIST. We argue that this is because FFLs allow modelling expressive distribution in contrast to possible oversmoothing behaviour of simple MLPs, as show in figure Figure B.3.
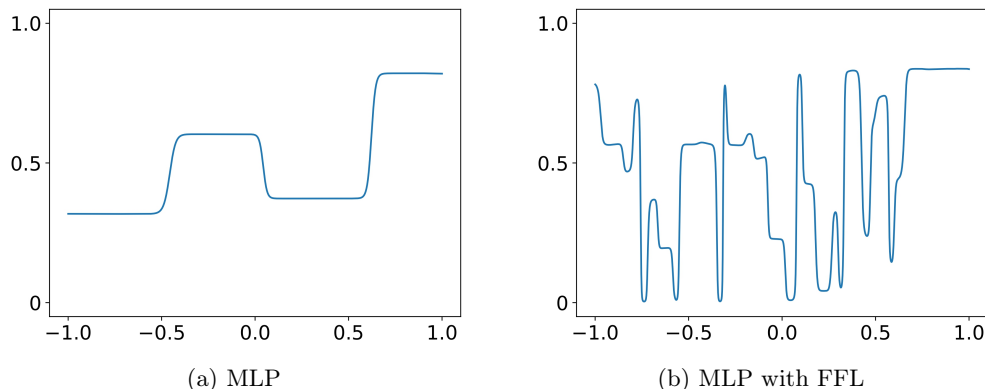

(a) MLP   (b) MLP with FFL

Figure B.3: **Fourier Features Layers allow modelling expressive distributions.** PIC root unit density learned using a standard MLP (a) vs. an MLP with FFL in (b).

B.3   Sum region heatmaps

In Figure B.4, we show the significant (but expected) difference between HCLT and QPC sum regions. We do this by visualising sum region parameters $\mathcal{S} \in [0, 1]^{N \times N}$ as a heatmap with child and parent states reported on the x and y-axis respectively. Sum regions of standard HCLTs look like the one in Figure B.4a, which resembles pure noise with no structure in it. This reflects the fact that standard sum units represent discrete LVs of unordered categorical states. Instead, QPC sum regions show interesting smooth patterns, as the one showed in Figure B.4b.

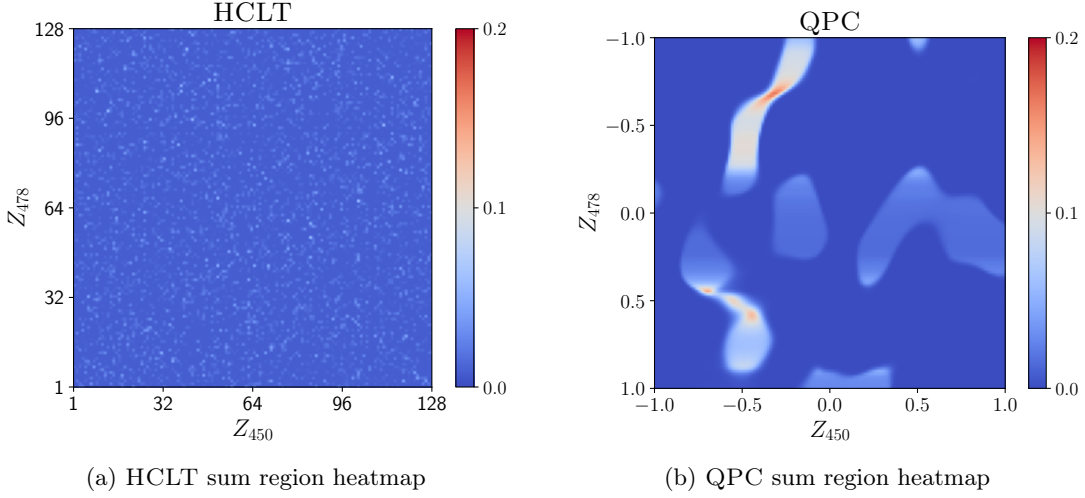(a) HCLT sum region heatmap

(b) QPC sum region heatmap

Figure B.4: **QPC sum regions present structure, unlike PC ones.** Z_{450} sum region heatmap of a standard HCLT(a) vs. a QPC (b) both with $N = 128$.

C   PICs AND QPCs FOR LINEAR GAUSSIAN MODELS

C.1   Tractable Gaussian PICs

In this section, we will show the symbolic computation of a tractable PIC representing a linear Gaussian LTM. Specifically, we will solve every integration problem by making use of the following result (Särkkä and Svensson, 2023): If $\mathbf{Z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{X} \mid \mathbf{Z} \sim \mathcal{N}(\mathbf{AZ} + \mathbf{b}, \boldsymbol{\Omega})$ then $\mathbf{X} \sim \mathcal{N}(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top + \boldsymbol{\Omega})$.
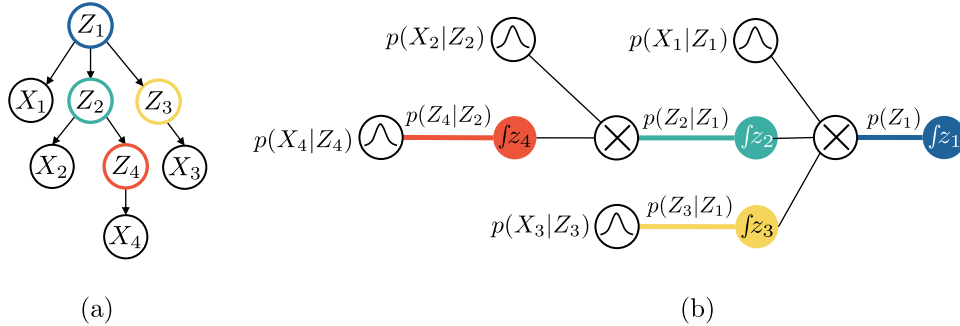

(a)

(b)

Figure C.5: An LTM (a) and its corresponding PIC (b) compiled via Alg. A.1 using $[Z_4, Z_2, Z_3, Z_1]$ as ordering.

Suppose that the LTM structure in Figure C.5(a) is a linear-Gaussian LTM whose marginal over $\mathbf{X}$ is

$$p(\mathbf{X}) = \int \psi_1(X_1, z_1) \int \psi_2(X_2, z_2, z_1) \int \psi_4(X_4, z_4, z_2) dz_4 dz_2 \int \psi_3(X_3, z_3, z_1) dz_3 dz_1,$$

where $\psi_i \equiv p(X_i \mid Z_i) p(Z_i \mid Z_{\mathsf{pa}(i)})$. Since RVs are in linear relationships between them, we have that:

- $Z_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

- $Z_i \sim \mathcal{N}(a_i Z_{\mathsf{pa}(i)} + b_i, \sigma_i^2)$, where $i \neq 1$

- $X_i \sim \mathcal{N}(c_i Z_i + d_i, \tau_i^2)$

where a_i, b_i, c_i and d_i belong to $\mathbb{R}$. Following the bottom-up computation of the PIC in Figure C.5(b), we could evaluate *in parallel* the integral unit $\fbox{$\int z_4$}$ and $\fbox{$\int z_3$}$ as:

$$\int \mathcal{N}(z_4 \,|\, a_4 z_2 + b_4, \sigma_4^2) \mathcal{N}(X_4 \,|\, c_4 z_4 + d_4, \tau_4^2) dz_4 = \mathcal{N}(X_4 \,|\, c_4(a_4 z_2 + b_4) + d_4, c_4^2 \sigma_4^2 + \tau_4^2)$$

$$\int \mathcal{N}(z_3 \,|\, a_3 z_1 + b_3, \sigma_3^2) \mathcal{N}(X_3 \,|\, c_3 z_3 + d_3, \tau_3^2) dz_3 = \mathcal{N}(X_3 \,|\, c_3(a_3 z_1 + b_3) + d_3, c_3^2 \sigma_3^2 + \tau_3^2)$$

Setting $\tilde{c}_4 = c_4 a_4$, $\tilde{d}_4 = c_4 b_4 + d_4$, $\tilde{\tau}_4^2 = c_4^2 \sigma_4^2 + \tau_4^2$, $\tilde{c}_3 = c_3 a_3$, $\tilde{d}_3 = c_3 b_3 + d_3$ and $\tilde{\tau}_3^2 = c_3^2 \sigma_3^2 + \tau_3^2$, we rewrite the integrals above as

$$\int \mathcal{N}(z_4 \,|\, a_4 z_2 + b_4, \sigma_4^2) \mathcal{N}(X_4 \,|\, c_4 z_4 + d_4, \tau_4^2) dz_4 = \mathcal{N}(X_4 \,|\, \tilde{c}_4 z_2 + \tilde{d}_4, \tilde{\tau}_4^2)$$

$$\int \mathcal{N}(z_3 \,|\, a_3 z_1 + b_3, \sigma_3^2) \mathcal{N}(X_3 \,|\, c_3 z_3 + d_3, \tau_3^2) dz_3 = \mathcal{N}(X_3 \,|\, \tilde{c}_3 z_1 + \tilde{d}_3, \tilde{\tau}_3^2)$$

Then, we evaluate the integral unit $\fbox{$\int z_2$}$, using the result of unit $\fbox{$\int z_4$}$ as

$$\int \mathcal{N}(z_2 \,|\, a_2 z_1 + b_2, \sigma_2^2) \mathcal{N}(X_2 \,|\, c_2 z_2 + d_2, \tau_2^2) \mathcal{N}(X_4 \,|\, \tilde{c}_4 z_2 + \tilde{d}_4, \tilde{\tau}_4^2) dz_2 = \mathcal{N}((X_2, X_4)^\top \,|\, \boldsymbol{\mu}', \boldsymbol{\Sigma}').$$

where

$$\boldsymbol{\mu}' = \begin{pmatrix} c_2(a_2 z_1 + b_2) + d_2 \\ \tilde{c}_4(a_2 z_1 + b_2) + \tilde{d}_4 \end{pmatrix}, \quad \boldsymbol{\Sigma}' = \begin{pmatrix} c_2^2 \sigma_2^2 + \tau_2^2, & \\ c_2^2 \tilde{c}_4^2 \sigma_2^2, & \tilde{c}_4^2 \sigma_2^2 + \tilde{\tau}_4^2 \end{pmatrix}.$$

Finally, we evaluate $\fbox{$\int z_1$}$—using the output functions from $\fbox{$\int z_2$}$ and $\fbox{$\int z_3$}$—as

$$\int \mathcal{N}(z_1 \,|\, \mu_1, \sigma_1^2) \mathcal{N}(X_1 \,|\, c_1 z_1 + d_1, \tau_1^2) \mathcal{N}((X_2, X_4)^\top \,|\, \boldsymbol{\mu}', \boldsymbol{\Sigma}') \mathcal{N}(X_3 \,|\, \tilde{c}_3 z_1 + \tilde{d}_3, \tilde{\tau}_3^2) dz_1 = \mathcal{N}(\mathbf{X} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where

$$\boldsymbol{\mu} = \begin{pmatrix} c_1 \mu_1 + d_1 \\ c_2(a_2 \mu_1 + b_2) + d_2 \\ \tilde{c}_3 \mu_1 + \tilde{d}_3 \\ \tilde{c}_4(a_2 \mu_1 + b_2) + \tilde{d}_4 \end{pmatrix},$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} c_1^2 \sigma_1^2 + \tau_1^2, & & & \\ a_1 a_2 c_2 \sigma_1^2, & a_2^2 c_2^2 \sigma_1^2 + c_2^2 \sigma_2^2 + \tau_2^2 & & \\ c_1 \tilde{c}_3 \sigma_1^2, & a_2 c_2 \tilde{c}_3 \sigma_1^2, & \tilde{c}_3^2 \sigma_1^2 + \tilde{\tau}_3^2 & \\ a_2 c_1 \tilde{c}_4 \sigma_1^2, & a_2^2 c_2 \tilde{c}_4 \sigma_1^2 + c_2 \tilde{c}_4 \sigma_2^2, & a_2 \tilde{c}_3 \tilde{c}_4 \sigma_1^2, & a_2^2 \tilde{c}_4^2 \sigma_1^2 + \tilde{c}_4 \sigma_2^2 + \tilde{\tau}_4^2 \end{pmatrix}.$$

C.2 Approximating linear-Gaussian models with QPCs

In this section, we discuss details and insights behind our sanity check in Section 4.1. Specifically, consider again the scenario in Appendix C.1 and suppose we want to numerically approximate the PIC in Figure C.5(b)—despite it being tractable—with a QPC.

Different from our energy-based distributions (cf. Section 3.2), Gaussians have an infinite support, $\mathbb{R}$. Nonetheless, a Gaussian $\mathcal{N}(\mu, \sigma^2)$ has a coverage of 99.7% within 3 standard deviations, and therefore using $[\mu - 3\sigma, \mu + 3\sigma]$

as integration domain could already be sufficient for a good numerical approximation, provided that the number of quadrature points N is large enough. However, the choice of such integration domain is integrand-dependent as the endpoints depend on the μ and σ of the Gaussian we are integrating out. As explained in Section 3.1 this is problematic. Indeed, for every one of the N integration points $\tilde{z}$ used to condition on a parent LV $Z_{\mathsf{pa}(i)}$, we instantiate N distinct Gaussian distributions of the form $\mathcal{N}(a_i\tilde{z}_n + b_i, \sigma_i^2)$, with n ranging from 1 to N. Every sum unit in a QPC is therefore associated with a specific Gaussian distribution, and thus requires specific integration domains. As a consequence, the application of integrand-dependent quadrature rules deliver an exponentially big QPC in the depth of the PIC to approximate.

As an example, applying integrand-dependent quadrature rules for approximating the PIC in Figure C.5(b) could result in the QPC shown in Figure C.6, if using $N = 3$ quadrature points for each integral approximation. Specifically, such QPC is obtained via Algorithm A.2, choosing the integration points in line 4 within the interval $[\mu - 3\sigma, \mu + 3\sigma]$ where $p(Z_i \mid Z_{\mathsf{pa}(i)} = \tilde{z}) = \mathcal{N}(\mu, \sigma^2)$. The circuit has no unit re-usage as the integration points are chosen independently for each sum unit. Note that, such exponential blow-up does *not* arise neither because of the usage of Gaussian distributions nor for the infinite integration domain, but only for using integrand-dependent integration points. As a consequence, we cannot represent such a nested quadrature approximation with a QPC whose size is quadratic in the number of integration points N, as no re-usage of units is possible.

To approximate the PIC in Figure C.5(b) using Algorithm 1 and therefore maintaining a quadratically large QPC, we avoid integrand-dependent quadrature rules in favor of large integration domains, so as ensuring a good coverage for all the Gaussian sum units within a region. Specifically, for each integral unit $\int z_i$ with density $p(Z_i|Z_{\mathsf{pa}(i)})$, we set the integration domain to $[\check{\mu} - 3\sigma_i, \hat{\mu} + 3\sigma_i]$, where $\check{\mu} = \min_{n \in [N]} a_i\tilde{z}_n + b_i$, $\hat{\mu} = \max_{n \in [N]} a_i\tilde{z}_n + b_i$ and $\tilde{\mathbf{z}}$ is the set of integration points used for $Z_{\mathsf{pa}(i)}$. This is exactly what we do in Section 4.1. This quadrature process proves effective, as shown in Figure 4, but only when the coefficients of the linear transformations, a_i, fall within a relatively small range, e.g. $[-2, 2]$. Instead, when arbitrary large coefficients are used, the integration bounds rapidly expand, rendering our static quadrature approach impractical.
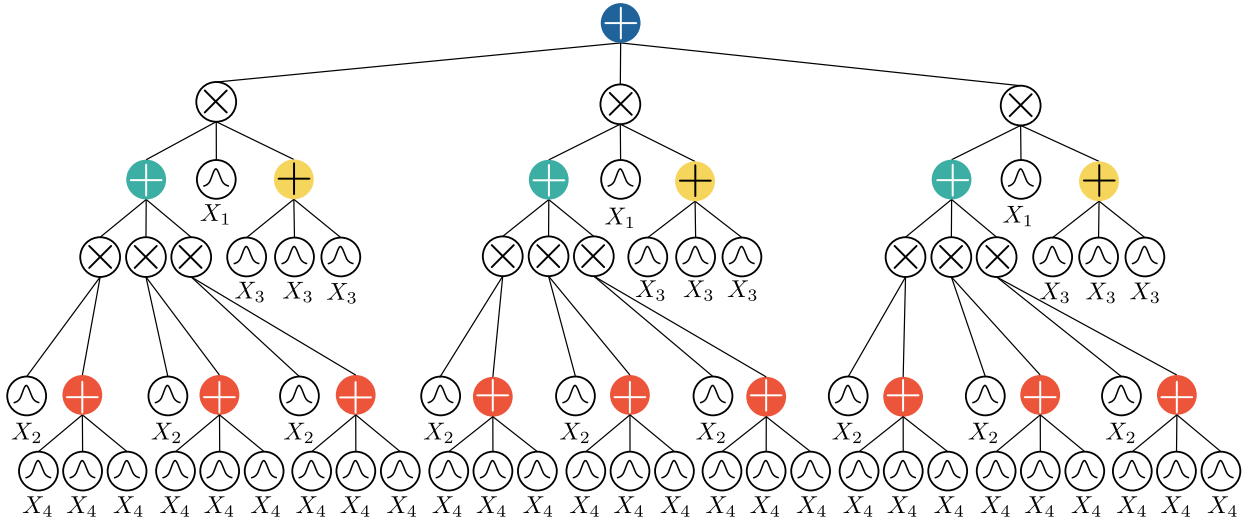

Figure C.6: **Exponentially large QPC via integrand-dependent quadrature rules.** A QPC materialised from the PIC in Figure C.5(b) via Algorithm A.2 using integrand-dependent numerical quadrature rules. The QPC is a tree as unit re-usage is generally not possible when using integrand-dependent rules.