Proceedings of the 7th Symposium on Advances in Approximate Bayesian Inference, 2025 1–21

Are Your Continuous Approximations Really Continuous? Reimagining VI with Bitstring Representations

Aleksanteri Sladek ALEKSANTERI.SLADEK@AALTO.FI
Martin Trapp MARTIN.TRAPP@AALTO.FI
Arno Solin ARNO.SOLIN@AALTO.FI
Aalto University

Abstract

Efficiently performing probabilistic inference in large models is a significant challenge due to the high computational demands and continuous nature of the model parameters. At the same time, the ML community has put effort into quantifying parameters of large-scale models to increase their computational efficiency. We extend this work by proposing a method for learning the probability distributions of quantized parameters via variational inference (VI). This enables effective learning of continuous distributions in a discrete space. We consider both 2D densities and quantized neural networks, where we introduce a tractable learning approach using probabilistic circuits. This method offers a scalable solution to manage complex distributions and provides clear insights into model behavior. We validate our approach in various settings, demonstrating its effectiveness.

1. Introduction

Probabilistic inference is central to modern machine learning, providing a principled framework for reasoning under uncertainty. In Bayesian inference, uncertainty is captured through posterior probability distributions over parameters, however, exact Bayesian inference is often intractable and has to be approximated. Variational inference (VI, Blei et al., 2017; Jordan et al., 1999; Wainwright and Jordan, 2008) is typically employed for this task as a scalable solution. A limitation of VI is that it relies on continuous parameterizations and often restrictive Gaussian assumptions, which can introduce representational and computational inefficiencies, particularly in large-scale settings.

To address computational constraints, the machine learning community has increasingly embraced quantization techniques for model parameters. These methods reduce numerical

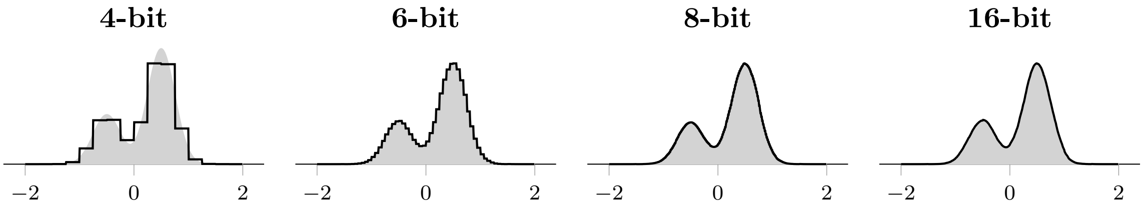


Figure 1: **Capturing a 1D Gaussian mixture** with BitVI with different numbers of bits in the bitstring. Even the 4-bit result serves a practical purpose, while the model saturates around 8 bits when compared to its 16-bit version.

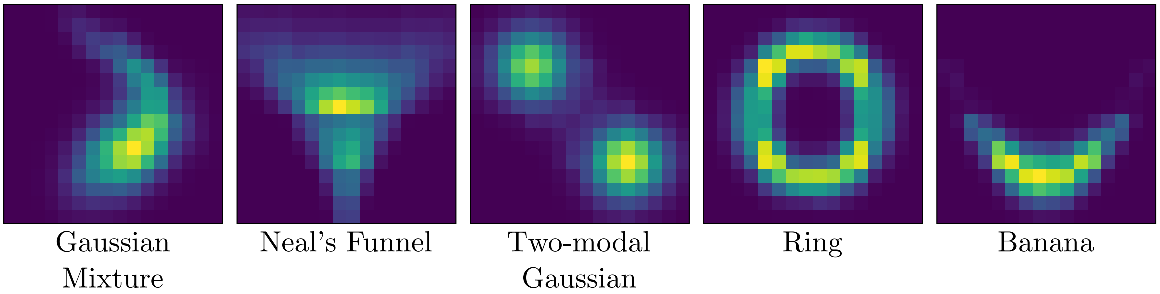


Figure 2: BitVI (4-bit) on non-Gaussian 2D density functions. We capture the overall and cross-densities well despite the low bit precision. We include comparisons to the ground-truth distributions and full-covariance VI in Fig. 8.

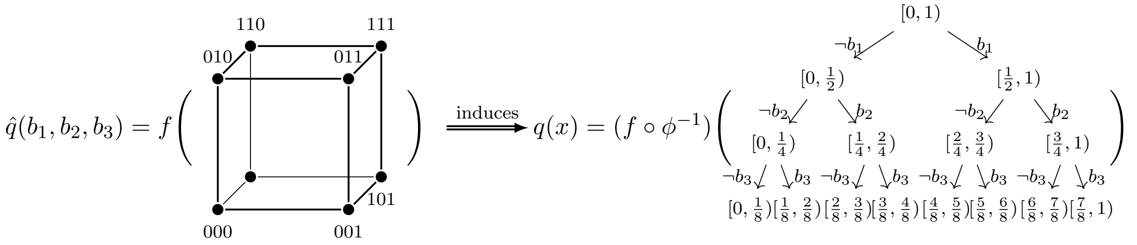
precision to improve efficiency, leveraging low-bit representations for storage and computation. Recent works utilizing large-scale mixed-precision FP8 (*e.g.*, Liu et al., 2024), FP4 (Wang et al., 2025), or even 1-bit neural architectures (Ma et al., 2024) show surprisingly good performance with low-precision parameters. An interesting question arises: *Could we also perform probabilistic inference directly in this discrete representation space of quantized parameters?*

As a preliminary thought experiment, consider Fig. 1, which illustrates how a Gaussian mixture model, typically represented in high-precision floating point, can be equivalently expressed using a low-precision bitstring representation.

This work introduces **BitVI**, a novel approach for approximate probabilistic inference in bitstring models. BitVI exploits the inherent discrete nature of number representations to approximate continuous distributions directly in the space of bitstrings. By leveraging probabilistic circuits (PCs, Choi et al., 2020), our method provides a tractable way to learn and perform inference over complex distributions without requiring high-precision representations. Fig. 2 demonstrates how BitVI can model complex distribution with only 4-bit precision. We validate BitVI across *(i)* standard benchmark densities, demonstrating its ability to approximate known distributions; and *(ii)* Bayesian deep learning in neural network models in *Bayesian Benchmarks*, where BitVI enables scalable and direct uncertainty quantification. Our results highlight the efficiency and accuracy of BitVI, making it a compelling alternative to traditional inference methods.

2. Methods

When performing computations on a computer, parameters of interest will inevitably be represented in a discretized form. Every real-valued number is represented by a series of bitstrings within the computer's hardware and then mapped to the real line by a mapping function $\phi\colon \{0,1\}^B \to \mathbb{R}$ given by the chosen number system (*e.g.* fixed-point, see Fig. 5). Consequently, any '*continuous*' distribution p or q represented on a computer can be expressed in terms of a *discrete* distribution over bitstrings. In the following, we outline how continuous distributions can be represented as a distribution over bitstrings with a tractable and flexible variational family.


$\hat{q}(b_1, b_2, b_3) = f($
$)$
110
111
010
011
101
000
001
induces
$q(x) = (f \circ \phi^{-1})($
$)$
$[0, 1)$
$\neg b_1$
b_1
$[0, \frac{1}{2})$
$[\frac{1}{2}, 1)$
$\neg b_2$
b_2
$\neg b_2$
b_2
$[0, \frac{1}{4})$
$[\frac{1}{4}, \frac{2}{4})$
$[\frac{2}{4}, \frac{3}{4})$
$[\frac{3}{4}, 1)$
$\neg b_3$ b_3 $\neg b_3$ b_3 $\neg b_3$ b_3 $\neg b_3$ b_3
$[0, \frac{1}{8})[\frac{1}{8}, \frac{2}{8})[\frac{2}{8}, \frac{3}{8})[\frac{3}{8}, \frac{4}{8})[\frac{4}{8}, \frac{5}{8})[\frac{5}{8}, \frac{6}{8})[\frac{6}{8}, \frac{7}{8})[\frac{7}{8}, 1)$

Distribution over bitstrings $\boldsymbol{b}$

Distribution over fixed-point numbers x

Figure 3: **Illustration of our method:** The bitstring can be visualized as a hypercube, and the PC induces a distribution over the fixed-point numbers represented by the bitstring. The corresponding PC structure is shown in Fig. 6.

2.1. BitVI: Variational Distributions over Bitstring Representations

Let $\hat{q}$ be a distribution over binary strings with probability measure $\hat{Q}$ defined on the measurable space of binary strings $(\mathcal{Y}, \mathcal{A})$ with corresponding σ-algebra $\mathcal{A}$. Further, let $(\mathbb{R}, \mathcal{B})$ be the measurable space of real numbers with Borel σ-algebra $\mathcal{B}$. Then, define a measurable mapping $\phi \colon \mathcal{Y} \to \mathbb{R}$ that assigns to each binary string a real number according to a specified number system, for example, the fixed point representation (see Fig. 5). The induced probability measure Q on $(\mathbb{R}, \mathcal{B})$ is defined as the pushforward measure of $\hat{Q}$ through ϕ. Specifically, for any Borel set $B \in \mathcal{B}$ we have $Q(B) = \hat{Q}(\phi^{-1}(B))$ where $\phi^{-1}(B)$ is the pre-image of B under ϕ. Finally, we represent the density q of Q using a (deterministic) probabilistic circuit (PC) (see App. C). The resulting construction is illustrated in Fig. 3 for the case of fixed-point numbers. Note that for fixed-point representations with infinite precision, this construction is equivalent to probability measures generated by Pólya trees (Ferguson, 1974; Trapp and Solin, 2022).

Consequently, by specifying a $\hat{q}$ over bitstrings and a respective number system, we obtain an induced variational distribution q on the real line. Next, our goal is to find a parameterization $\boldsymbol{\theta}$ of our variational distribution that minimizes its KL-divergence (see Eq. (5)) from some true distribution being approximated. When representing q using a deterministic PC, the parameters $\boldsymbol{\theta}$ correspond to the collection of weights $\{w_i\}_i$ of the circuit. Note that by construction, the leaf nodes of our circuit model are continuous uniform distributions and, therefore, do not have any additional parameters. The resulting deterministic PC is a tree with depth proportional to the number of bits used in the bitstring representation. Each sum node in the PC represents the decision of a bit, and its weight corresponds to the conditional probability of the respective decision. For example, the probability of 0.5 in a 3-bit fixed-point number system with one integer bit and no sign-bit, which corresponds to the bitstring 010, is computed by obtaining the bit decisions, *i.e.*, $b_0 = 0$, $b_1 = 1$, and $b_2 = 0$, and evaluating the circuit along the respective path, *i.e.*, $p(x = 0.5) = w_0 w_{01} w_{010} \frac{1}{2^{B_{\text{frac}}}}$ where $B_{\text{frac}} = 2$ is the number of fraction bits. Fig. 6 illustrates the decision process reflected by the circuit.

Computation of the ELBO A convenient property of deterministic PCs is that their entropy can be computed in linear time w.r.t. the number of edges of the circuit (Vergari et al., 2021), see App. D for details. Consequently, for computing the ELBO, we only need to approximate the expected log probability in Eq. (6) using Monte Carlo (MC) integration. To do so, we first use a reparameterization via the inverse CDF transform, which is also available analytically for deterministic PCs. In particular, we use the following reparameterization of the ELBO,

$$\mathcal{L}(q_{\boldsymbol{\theta}}, p) = \mathbb{E}_{u \sim \mathsf{Unif}(0,1)} \left[\log p(F_{q_{\boldsymbol{\theta}}}^{-1}(u)) \right] + \mathcal{H}\left(q_{\boldsymbol{\theta}}\right), \tag{1}$$

where $F_{q_{\boldsymbol{\theta}}}^{-1}(\cdot)$ is the inverse CDF transform of $q_{\boldsymbol{\theta}}$. We then generate T samples from a uniform distribution $u^t \sim \mathsf{Unif}(0, 1)$ and compute a MC estimate of Eq. (1), *i.e.*,

$$\mathcal{L}(q_{\boldsymbol{\theta}}, p) \approx \frac{1}{T} \sum_{t=1}^{T} \log p(F_{q_{\boldsymbol{\theta}}}^{-1}(u_t)) + \mathcal{H}\left(q_{\boldsymbol{\theta}}\right). \tag{2}$$

Note that Eq. (2) can be computed efficiently.

Remark 1 *The inverse CDF transform of $q_{\boldsymbol{\theta}}$ can be computed in time linear in the depth of the circuit.*

For a given input y, we can compute the inverse CDF transform of y under $q_{\boldsymbol{\theta}}$ using a series of linear transformations. In particular, for sum nodes S we compute

$$F_{\mathsf{S}_\epsilon}^{-1}(y) = \begin{cases} F_{\mathsf{C}_{\epsilon 1}}^{-1}\left(\frac{y - w_{\epsilon 0}}{w_{\epsilon 1}}\right) & \text{if } y > w_{\epsilon 0} \\ F_{\mathsf{C}_{\epsilon 0}}^{-1}\left(\frac{y}{w_{\epsilon 0}}\right) & \text{otherwise} \end{cases}, \tag{3}$$

where C denotes a child node of S, *i.e.* a sum or leaf node, and $\epsilon \in \bigcup_{j=0}^{B} \{0, 1\}^j$ is a binary string. If C is a leaf node, we compute the inverse CDF according to the respective leaf distribution, *i.e.*, $F_{\psi}^{-1}(y) = y(b-a)+a$ in case of a continuous uniform distribution $\mathsf{Unif}(a, b)$.

Note that the resulting value still requires discretization and, in the case of fixed-point numbers, needs to be rounded to the nearest fixed-point value. Fortunately, the bitstring ϵ generated by traversing the circuit in order to compute its inverse CDF already encodes the nearest fixed-point value for y. However, as the discretization operation does not have a well-defined gradient, we resort to the application of the straight-through estimator (STE) (Bengio et al., 2013). In particular, we compute:

$$x = (\phi(\epsilon) + F_{q_{\boldsymbol{\theta}}}^{-1}(y)) - F_{q_{\boldsymbol{\theta}}}^{-1}(y), \tag{4}$$

where $\phi(\epsilon)$ is the mapping function defined by the number system and the bitstring ϵ is a function of $F_{q_{\boldsymbol{\theta}}}^{-1}$ and indicates the decision taken in Eq. (7).

3. Experiments

We begin with a 2D density estimation task in Sec. 3.1 to demonstrate the effectiveness of our method in capturing complex non-Gaussian distributions. Next, Sec. 3.2 explores learning higher-dimensional posterior densities in the Bayesian deep learning setting by applying BitVI to Bayesian neural networks (NNs), showcasing its ability to perform effective uncertainty quantification in predictive modeling. We also conduct a series of ablation studies (See App. F.1) to assess the trade-offs between numerical precision and model expressiveness, investigating the effect of bitstring depth on performance and the role of hierarchical structure in NNs.

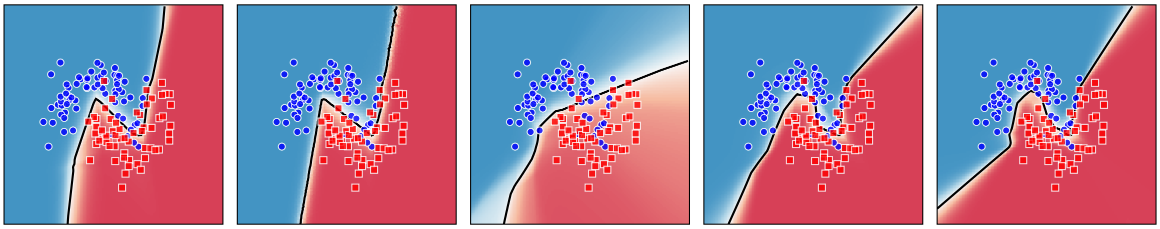


Figure 4: **Uncertainty quantification in neural networks:** We consider the two moons binary ■● classification problem with a Bayesian neural network (two hidden layers). The predictive density (▬) shows that BitVI provides both uncertainties in the decision boundary (white regions) and good decision boundaries compared to the deterministic and MFVI baselines.

3.1. 2D Densities

First, we demonstrate the flexibility of our proposed approach in 2D non-Gaussian target distributions. In Fig. 2, we include results of typical benchmark target densities (mixture, Neal's funnel, two-modal Gaussian, ring, and banana) that we approximate with 4-bit and 8-bit BitVI. Moreover, Fig. 8 illustrates a comparison for two densities, indicating that BitVI captures the overall density and cross-dependencies well, with approximation quality increasing with the number of bits. For reference, we also include the exact target densities and the approximation by full-covariance Gaussian VI (FCGVI) in Fig. 8.

3.2. Higher-dimensional densities: Bayesian Neural Networks

Next, we experiment with approximating the posterior density of the parameters of a Bayesian NN model using BitVI. For simplicity, we use similar NN architectures in all the NN experiments. We use two hidden layers in all experiments and only vary the number of units. Additionally, we use the layer norm (Ba et al., 2016) to limit weight scaling.

Fig. 4 shows an uncertainty quantification example. We consider the two moons binary classification problem with an NN ([8,8] hidden units). The predictive density shows that BitVI provides both representative uncertainties and good decision boundaries compared to the deterministic and mean-field Gaussian VI baselines.

To give a more quantitative treatment to the NN modeling task, we use the *Bayesian Benchmarks*[1] community suite meant for benchmarking Bayesian methods in machine learning. Bayesian benchmarks include common evaluation data sets (typically from UCI (Kelly et al., 2025)) and make it possible to run many comparisons under a fixed evaluation setup. We evaluate our approach in binary classification, and for an interesting probabilistic treatment, we include small-data binary classification tasks with $100 \leq n \leq 1000$ data samples (25 data sets). We follow the standard setup of input point normalization and splits in the evaluation suite. Additional details on the NN architectures and evaluation setup can be found in App. E.2.

[1] github.com/secondmind-labs/bayesian_benchmarks; originally by Salimbeni *et al.*

Table 1: **Bayesian benchmarks:** Negative log predictive density (NLPD±std, smaller better) results on the *Bayesian Benchmarks* UCI tasks (5-fold CV). We comapre BitVI to Gaussian MFVI and Full-covariance Gaussian VI (FCGVI). The best-performing method for each task is bolded, and multiple methods bolded based on a paired t-test ($p = 5\%$).

Dataset	(n, d)	MFVI	FCGVI	2-BitVI	4-BitVI	8-BitVI
fertility	(100,10)	**0.379**±0.107	0.406±0.111	0.728±0.139	**0.407**±0.109	**0.406**±0.142
pittsburg-bridges-T-OR-D	(102,8)	**0.345**±0.168	**0.347**±0.078	**0.301**±0.064	**0.352**±0.082	**0.391**±0.068
acute-inflammation	(120,7)	**0.004**±0.001	0.021±0.009	**0.006**±0.002	**0.006**±0.002	0.684±0.031
acute-nephritis	(120,7)	**0.003**±0.001	0.014±0.003	**0.002**±0.000	**0.002**±0.002	0.051±0.016
echocardiogram	(131,11)	**0.446**±0.167	**0.515**±0.151	**0.524**±0.200	**0.435**±0.095	0.660±0.132
hepatitis	(155,20)	**0.438**±0.081	**0.447**±0.116	**0.620**±0.246	**0.694**±0.279	**0.427**±0.085
parkinsons	(195,23)	**0.322**±0.151	**0.284**±0.109	**0.253**±0.098	**0.261**±0.064	**0.289**±0.061
breast-cancer-wisc-prog	(198,34)	**0.540**±0.106	**0.522**±0.128	0.699±0.087	**0.584**±0.073	**0.548**±0.087
spect	(265,23)	**0.614**±0.067	**0.624**±0.053	0.801±0.108	0.807±0.148	**0.670**±0.125
statlog-heart	(270,14)	**0.478**±0.133	**0.488**±0.156	**0.550**±0.207	**0.606**±0.270	**0.478**±0.147
haberman-survival	(306,4)	**0.535**±0.062	**0.523**±0.054	**0.531**±0.042	**0.525**±0.044	**0.530**±0.036
ionosphere	(351,34)	**0.288**±0.094	**0.276**±0.092	**0.335**±0.126	**0.459**±0.217	**0.323**±0.127
horse-colic	(368,26)	**0.611**±0.159	**0.595**±0.163	0.618±0.119	0.690±0.143	**0.576**±0.103
congressional-voting	(435,17)	**0.670**±0.093	**0.700**±0.126	**0.699**±0.105	**0.704**±0.108	**0.644**±0.048
cylinder-bands	(512,36)	**0.602**±0.107	**0.633**±0.050	0.835±0.222	**0.955**±0.361	**0.678**±0.019
breast-cancer-wisc-diag	(569,31)	**0.078**±0.050	**0.108**±0.029	**0.148**±0.080	**0.172**±0.152	**0.155**±0.097
ilpd-indian-liver	(583,10)	**0.547**±0.059	**0.547**±0.033	**0.535**±0.053	**0.518**±0.032	**0.567**±0.025
monks-2	(601,7)	**0.083**±0.121	0.607±0.082	0.563±0.060	0.656±0.073	0.666±0.030
credit-approval	(690,16)	**0.357**±0.025	**0.417**±0.096	0.405±0.041	**0.358**±0.026	**0.343**±0.009
statlog-australian-credit	(690,15)	**0.662**±0.035	0.650±0.029	0.764±0.075	**0.629**±0.019	**0.626**±0.019
breast-cancer-wisc	(699,10)	**0.091**±0.042	**0.105**±0.041	**0.171**±0.113	0.168±0.055	**0.122**±0.059
blood	(748,5)	**0.483**±0.058	**0.483**±0.036	**0.486**±0.057	**0.478**±0.043	**0.486**±0.039
pima	(768,9)	0.516±0.045	**0.507**±0.042	**0.512**±0.039	**0.492**±0.031	**0.492**±0.042
mammographic	(961,6)	**0.428**±0.039	0.468±0.044	**0.430**±0.053	**0.417**±0.039	**0.423**±0.049
statlog-german-credit	(1000,25)	**0.547**±0.066	**0.557**±0.086	0.651±0.092	**0.646**±0.101	0.894±0.249

Table 1 shows the results for BitVI (with 2, 4, and 8 bits), mean-field Gaussian VI (MFVI), and full-covariance Gaussian VI (FCGVI). Our approach performs competitively with the standard VI baselines, even in the low-bit regime. BitVI with 4-bit and 8-bit representations achieves comparable performance to MFVI and FCGVI, demonstrating that probabilistic inference can be effectively conducted over bitstring representations without significant loss in predictive power. Even 2-bit precision remains viable in several cases.

4. Conclusion

This work introduced **BitVI**, a novel approach for approximate Bayesian inference that operates directly in the space of discrete bitstring representations. We demonstrated that inference can be performed directly on bitstring representations of number systems while still enabling effective approximate inference and uncertainty quantification. Our experiments showcased the flexibility of BitVI across different settings: In Sec. 3.1, we illustrated its ability to approximate complex non-Gaussian 2D densities; and in Sec. 3.2, we demonstrated its effectiveness in a higher-dimensional Bayesian deep learning posterior inference setting, providing robust uncertainty estimates while maintaining computational efficiency.

While BitVI provides a promising direction for flexible variational inference, several limitations remain. Since the circuit models the bitstring of each parameter, our approach introduces many new parameters to be optimized, which can result in further challenges for high-dimensional settings. In order to scale to high-dimensional settings, our approach also employed a mean-field approximation to the posterior, meaning dependencies between model parameters were not modeled. In practical applications, however, modeling all dependencies is likely unnecessary. Therefore, a promising future direction is to leverage more compact circuit representations such as (Peharz et al., 2020).

Acknowledgments

Arno Solin acknowledges funding from the Research Council of Finland (grant number 339730). Martin Trapp acknowledges funding from the Research Council of Finland (grant number 347279). Aleksanteri Sladek acknowledges funding from the Finnish Doctoral Program Network in Artificial Intelligence, AI-DOC (decision number VN/3137/2024-OKM-6). We acknowledge the computational resources provided by the Aalto Science-IT project. We thank the reviewers for their valuable feedback and suggestions.

References

Naoki Awaya and Li Ma. Unsupervised tree boosting for learning probability distributions. *Journal of Machine Learning Research*, 25(198):1–52, 2024.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, abs/1607.06450, 2016.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

Oliver Broadrick, Honghua Zhang, and Guy Van den Broeck. Polynomial semantics of tractable probabilistic circuits. In *40th Conference on Uncertainty in Artificial Intelligence (UAI)*, Proceedings of Machine Learning Research, pages 418–429. PMLR, 2024.

Ismaël Castillo. Pólya tree posterior distributions on densities. *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, 53(4):2074 – 2102, 2017.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, University of California, Los Angeles (UCLA), 2020.

Thomas S. Ferguson. Prior Distributions on Spaces of Probability Measures. *The Annals of Statistics*, 2(4):615 – 629, 1974.

Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Bit blasting probabilistic programs. *Proc. ACM Program. Lang.*, 8, 2024.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. The UCI machine learning repository. https://archive.ics.uci.edu, 2025.

Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Professional, 1997.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Daniel Lowd and Pedro Domingos. Approximate inference by compilation to arithmetic circuits. pages 1477–1485, 2010.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

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 *37th International Conference on Machine Learning (ICML)*, pages 7563–7574. PMLR, 2020.

Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. Sppl: probabilistic programming with fast exact symbolic inference. In *42nd International Conference on Programming Language Design and Implementation (ACM/SIGPLAN)*, pages 804–819, 2021.

Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*. Curran Associates, 2020.

Pat H. Sterbenz. *Floating-point Computation*. Prentice-Hall Series in Automatic Computation, 1974.

Martin Trapp and Arno Solin. On priors in Bayesian probabilistic circuits and multivariate pólya trees. In *The 5th Workshop on Tractable Probabilistic Modeling*, 2022.

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, pages 6344–6355. Curran Associates, 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 *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 13189–13201. Curran Associates, 2021.

Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Benjie Wang, Denis Deratani Mauá, Guy Van den Broeck, and YooJung Choi. A compositional atlas for algebraic circuits. In *Advances in Neural Information Processing Systems 38 (NeurIPS)*. Curran Associates, 2024.

Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zhengjun Zha, and Peng Cheng. Optimizing large language model training using FP4 quantization. *arXiv preprint arXiv:2501.17116*, 2025.

Zhongjie Yu, Martin Trapp, and Kristian Kersting. Characteristic circuits. In *Advances in Neural Information Processing Systems 36 (NeurIPS)*. Curran Associates, 2023.

Appendix A. Background and Related Work

The relationship between continuous and discrete representations is fundamental to computational science. At its core, digital computation relies on discrete structures, with real-valued quantities encoded as finite-length bitstrings (Ch. 4 Knuth, 1997). Floating-point arithmetic provides an approximation to continuous values within this discrete framework, ensuring efficient numerical operations while introducing inherent precision limitations (Ch. 1 Sterbenz, 1974). In recent years, this foundational connection has gained renewed attention in machine learning, particularly due to advances in quantization and low-precision arithmetic. While these techniques are primarily motivated by hardware constraints, they also present an opportunity: if inference can be formulated directly over discrete bitstring representations, it may unlock new efficiencies in probabilistic modelling.

Bayesian inference provides a principled framework for reasoning under uncertainty, yet exact inference remains intractable in most real-world scenarios. This has led to the development of approximate inference techniques, such as variational inference (VI) (Blei et al., 2017; Jordan et al., 1999; Wainwright and Jordan, 2008). VI formulates inference as an optimization problem, where a parametric distribution is fitted to approximate the posterior while minimizing the reverse KL divergence. Despite its scalability, VI is often constrained by its reliance on continuous parameterizations, which can introduce numerical instabilities and bias due to restrictive approximations such as mean-field or unimodality assumptions. These limitations are apparent when operating under low-precision, raising the question: *Can we perform inference directly in a discrete representation space?*

Probabilistic circuits (PCs) are a recent framework to study tractable representations of complex probability distribution Choi et al. (2020). Depending on the structural properties of the PC, certain inference scenarios can be rendered tractable (polynomial in the model complexity) under the circuit while maintaining a high expressivity. While PCs are typically employed for exact probabilistic inference, they have found successful application in approximate Bayesian inference, for example, as surrogate through compilation Lowd and Domingos (2010), as variational distribution for structured discrete models (Shih and Ermon, 2020), or in discrete probabilistic programs (Saad et al., 2021). Most related to our work, Garg et al. (2024) utilized PCs over bitstring representation for efficient approximate inference in probabilistic programs. This works highlight that PCs are a natural and promising representational framework for approximate Bayesian and uncertainty quantification.

Appendix B. Motivation

Given a target density p, we aim to find a variational approximation q that minimises the divergence of p from q. As commonly done, we will focus on the reverse Kullback–Leibler (KL) divergence of q from p, instead of the forward KL. Moreover, we assume that q takes a parametric form with parameters $\boldsymbol{\theta}$, *i.e.*, $q_{\boldsymbol{\theta}}$. Therefore, the goal is find $\boldsymbol{\theta}$ such that

$$\mathrm{KL}(q_{\boldsymbol{\theta}} \,\|\, p) = \int_{x \in \mathcal{X}} q_{\boldsymbol{\theta}}(x) \log\left(\frac{q_{\boldsymbol{\theta}}(x)}{p(x)}\right) \mathrm{d}x, \tag{5}$$

is minimized, assuming that $\mathcal{X} \subseteq \mathbb{R}^d$ for some $d \geq 1$.

In general, computing Eq. (5) is intractable for two reasons: *(i)* p is often only known up to an unknown normalisation constant Z_p and *(ii)* p and q do not exhibit sufficient structure to render the integration tractable (Wang et al., 2024). Henceforth, one typically optimises the evidence lower bound (ELBO), which can be written as

$$\mathcal{L}(q_{\boldsymbol{\theta}}, p) = \mathbb{E}_{x \sim q_{\boldsymbol{\theta}}} [\log p(x)] + \mathcal{H}(q_{\boldsymbol{\theta}}), \tag{6}$$

where $\mathcal{H}(q_{\boldsymbol{\theta}}) = -\mathbb{E}_{x \sim q_{\boldsymbol{\theta}}} [\log q_{\boldsymbol{\theta}}(x)]$ denotes the entropy of the variational distribution $q_{\boldsymbol{\theta}}$. In case $q_{\boldsymbol{\theta}}$ admits a tractable computation of the entropy, only the first term in Eq. (6) requires numerical approximation.

Crucially, when computing either Eq. (5) or Eq. (6) on a computer each x will inevitably be represented in a discretised form. In fact, every real-valued number is represented by a series of bitstrings and mapped to the real line by a mapping function $\phi \colon \{0,1\}^B \to \mathbb{R}$ given by the choosen number system. Consequently, any distribution p or q represented on a computer can be expressed in terms of a distribution over bitstrings. Fig. 5 illustrates the representation of a real-valued number using an 8-bit fixed point representation.

Appendix C. Technical Details

C.1. The Fixed-point Number Representation System

The fixed-point number representation system represents one possible way to convert a bitstring to a real valued number. Here we briefly review the sign-magnitude form of this number representation system, in which the most significant bit is used to represent the sign of a number (1 for negative, 0 for positive). The remaining bits are referred to as magnitude bits, which are further divided into integer bits and fractional bits. As suggested by their names, integer bits encode the integer part of the real value being represented and the fractional bits the fractional part. In this case, integer bits encode which non-negative powers of two are present in the integer being encoded and the fractional bits encode which negative powers of two are present in the fractional number being encoded. For example in Fig. 5, the bit-string illustrated encodes the computation $0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4}$ (not including the sign-bit).

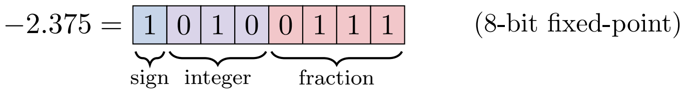


Figure 5: Representation of '−2.375' using an 8-bit fixed-point number system with sign, integer, and fraction bits.

C.2. Probabilistic Circuits

We will briefly review the main concepts related to probabilistic circuits (PC), relevant for this work. For a more extensive discussion, we refer the reader to (Choi et al., 2020). Put briefly, PCs are directed acyclic graphs that represent a probability distribution. Nodes in a PC consist of sum nodes S, product nodes P and leaf nodes L. PCs are computational graphs, where sum nodes compute weighted sum of their child nodes and product nodes compute a product of their children. Leaf nodes represent univariate or multivariate functions, such as Gaussians. Given certain constraints on the structure of a PC graph, inference tasks such as marginalisation can be performed efficiently. Below, a few of these properties are reviewed.

Definition 2 (Scope of a node) *The scope of a node is the set of variables it depends on. See (Trapp et al., 2019) for details.*

Definition 3 (Smooth & decomposable circuit) *A sum node is smooth if its children have the same scope. A product node is decomposable if its children have pairwise disjoint scopes. A circuit is smooth (resp. decomposable) if all its sum nodes are smooth (resp. product nodes are decomposable).*

Definition 4 (Deterministic probabilistic circuit) *A probabilistic circuit $f(\boldsymbol{x})$ is multilinear function represented by a computational graph consisting of sum nodes $\mathsf{S}(\boldsymbol{x}) = \oplus_i w_i f_i(\boldsymbol{x})$, product nodes $\mathsf{P}(\boldsymbol{x}) = \otimes_i f_i(\boldsymbol{x})$, and leaf nodes consisting of tractable (univariate) functions $\psi_i(x)$. The circuit f characterises a multivariate probability distribution over random variables $X_1, \ldots, X_d$ by, for example, representing its mass, density or characteristic function (Yu et al., 2023; Broadrick et al., 2024). Note that we assume that the circuit is smooth and decomposable (Choi et al., 2020) and refer to App. C for details.*

We call a sum node S deterministic if for each instantiation $\boldsymbol{x}$ only one summand is positive. Consequently f is deterministic if all sum nodes are deterministic (Choi et al., 2020).

In this work, we only consider circuits that fullfil both smoothness and decomposability conditions as they both are required to render common inference tasks, such as density evaluation and marginalisation, tractable.

C.3. Multivariate Bitstring Representations

So far, our induced variational distribution is only defined on the real line (univariate case). To extend the approach to the multivariate case, we considered two approaches: *(i)* a mean-field variational family, and *(ii)* a variational family model dependencies between dimensions. To represent dependencies between the dimensions, we construct a deterministic PC over the states of the bits of all the dimension jointly. In case of the fixed-point number systems, the resulting circuit model recursively splits the domain into hyper-rectangles by performing axis-aligend splits, alternate between dimensions in the construction. Note that this construction results in a binary tree consisting of 2^{B*D} many leaves, where B is the number of bits and D the number of dimensions. Thus, making it useful in low-dimensional or low-precision settings. However, including conditional independencies in the model can results in substantially more compact representations (Peharz et al., 2020; Garg et al., 2024). We provide further details on the construction in App. C.

Applying the inverse CDF reparameterization for multivariate densities modelled with BitVI requires further considerations. In case of a mean-field approximation, we apply the inverse CDF reparameterization as described above independently for each dimension. If BitVI represents a variational distribution that models dependencies between dimensions,

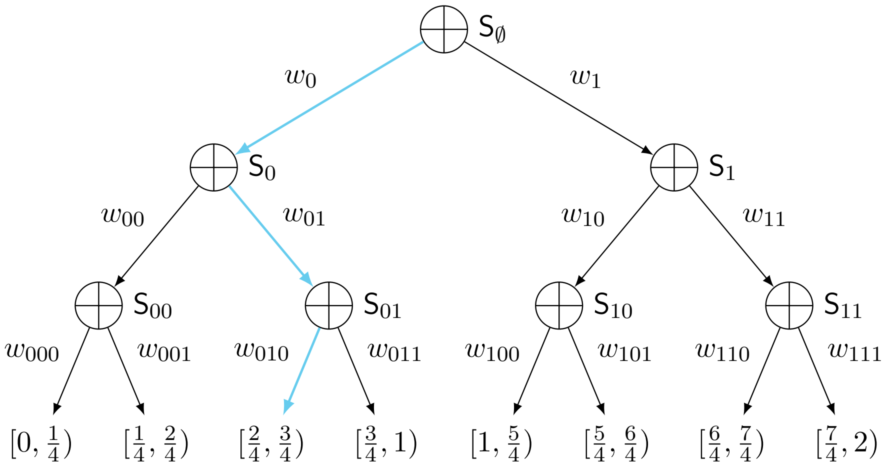


Figure 6: The decision process reflected the by the circuit.

we emply the inverse of the tree-CDF transformation (Awaya and Ma, 2024) which is a map $\mathbb{R}^D \to [0,1]^D$ where D is the number of dimensions. In particular, for a given input $\boldsymbol{y} \in [0,1]^D$, we compute the inverse tree-CDF transform of $\boldsymbol{y}$ by applying the following axis-aligned linear transformations at each sum node, where $\mathsf{S}_{d,\epsilon_d}$ denotes the sum node for dimension $d \leq D$ under bitstring ϵ_d. The axis-aligned transformations are given as:

$$F^{-1}_{\mathsf{S}_{d,\epsilon_d}}(y_d) = \begin{cases} F^{-1}_{\mathsf{C}_1}\left(\frac{y_d - w_{d,\epsilon_d 0}}{w_{d,\epsilon_d 1}}\right) & \text{if } y_d > w_{d,\epsilon_d 0} \\ F^{-1}_{\mathsf{C}_0}\left(\frac{y_d}{w_{d,\epsilon_d 0}}\right) & \text{otherwise} \end{cases}, \tag{7}$$

where with some abuse of notation C_0 denotes the left child of $\mathsf{S}_{d,\epsilon_d}$, which corresponds to a bit value of zero, and C_1 denotes the right child (bit value of one). As we alternate dimensions at each level in the tree, decision are made only based on the 'selected' dimension at each step. Computing the inverse of the tree-CDF transformation can still be perfomed efficiently, *i.e.*, in $\mathcal{O}(B * D)$ for B bits. To encourage that q has a smooth density in the limit of infinite precision, we leverage a depth regularisation scheme when optimizing the circuit weights (see App. C.4 for further details).

As outlined in the main text, for multivariate distributions we generate a circuit model that represents a distribution over hyper-rectangles. Let Ω denote the domain of the distribution, we recursively construct a dyadic partition of the domain into measurable subsets. This process is done by selecting a splitting dimension at each level of the tree and spliting the hyper-rectange according to the number system representation, *i.e.*, in the middle for fixed-point numbers. At the next level, we select a splitting dimension our of the remaining dimension (those that have not been split yet) and split the hyper-rectange accordingly. We make sure each dimension has been split in the process, before restarting the splitting. The construction ends if each dimension has been split B many times, where B is the number of bits used in the number system. Fig. 7 illustrates the recursive splitting of the input domain Ω into sub-domains (hyper-rectangles).

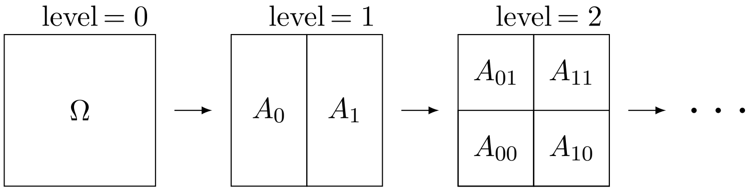


Figure 7: Illustration of the iterative axis-aligned splitting of the domain into hyper-rectangles (sub-domains) by the circuit.

C.4. Depth Regularization

To encourage that q has a smooth density in the limit of infinite precision, we leverage a depth regularisation. The depth regularisation is based on Pólya tree prior constructions for priors over continuous probability distributions. Specifically, Ferguson (1974) proposed to use a Beta prior one each weight of a Pólya tree with symmetric α-parameter that has a quadratic increase in depth j of the tree, *i.e.*, $\alpha(j) = j^2$. An alternative parameterization is was given by Castillo (2017), and is given as $\alpha(j) = 2^j$. In essence, both approaches ensure that with increasing depth the prior probability of uniform distributed weight increases fast enough. We adopt this approach and use Laplace smoothing of the circuit weights with a depth depending smoothing factor. In particular, for bit b_j (depth j) with $j \geq 0$ we define each weight for $b_j = 0$ as

$$w_{\epsilon 0} = \frac{v_{\epsilon 0} + c\alpha(j)}{v_{\epsilon 0} + v_{\epsilon 1} + 2c\alpha(j)}, \tag{8}$$

where ϵ denotes a $j-1$ long binary string, $v_{\epsilon 0} > 0$ is an unnormalised weight, and $c > 0$ is a hyper-parameter. The weight for $\epsilon 0$ is given analogously.

Appendix D. Derivations

D.1. Entropy calculation example for a deterministic PC

Let $\mathcal{C}$ be a deterministic PC with a sum node S_0 as its root and two children P_{00} and P_{01}. Product nodes have two children, one of which is a leaf node and the other a sum node. For example, P has a leaf node $\mathsf{L}_0 0$ and a sum node $\mathsf{S}_0 0$ as it's children. Leaf nodes are indicator functions $\mathsf{L}_{00}(\mathbf{x}) = \mathbb{1}\{x_0 = 0\}$, where $\mathbb{1}\{\cdot\}$ is the indicator function. All sum nodes in the circuit have two children. Hence, S_0 has weights w_{00} and $1 - w_{00}$, where $0 \leq w_{00} \leq 1$.

$$
\begin{aligned}
\mathcal{H}\left[\mathcal{C}(\mathbf{x})\right] &= -\int_{\mathbf{x}\in\mathcal{X}} \mathcal{C}(\mathbf{x}) \log \mathcal{C}(\mathbf{x}) \qquad (9)\\
&= -\int_{\mathbf{x}\in\mathcal{X}} \mathsf{S}_0(\mathbf{x}) \log\left(\mathsf{S}_0(\mathbf{x})\right) \qquad (10)\\
&= -\int_{\mathbf{x}\in\mathcal{X}} \left[w_{00}\mathsf{P}_{01}(\mathbf{x}) + (1-w_{00})\mathsf{P}_{00}(\mathbf{x})\right] \log\left\{\left[w_{00}\mathsf{P}_{01}(\mathbf{x}) + (1-w_{00})\mathsf{P}_{00}(\mathbf{x})\right]\right\} \qquad (11)\\
&= -\int_{\mathbf{x}\in\mathcal{X}} \left[w_{00}\mathsf{L}_{01}(x_0) * \mathsf{S}_{00}(\hat{\mathbf{x}}) + (1-w_{00})\mathsf{L}_{01}(x_0) * \mathsf{S}_{00}(\hat{\mathbf{x}})\right] * \qquad (12)\\
&\log\left\{\left[w_{0,0}\mathsf{L}_{01}(x_0) * \mathsf{S}_{0,0}(\hat{\mathbf{x}}) + (1-w_{00})\mathsf{L}_{00}(x_0)\mathsf{S}_{0,1}(\hat{\mathbf{x}})\right]\right\} \qquad (13)\\
&= -\int_{\mathbf{x}\in\mathcal{X}} \left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\hat{\mathbf{x}}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\hat{\mathbf{x}})\right] * \qquad (14)\\
&\log\left\{\left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\hat{\mathbf{x}}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\hat{\mathbf{x}})\right]\right\}. \qquad (15)
\end{aligned}
$$

Where $\hat{\mathbf{x}}$ denotes the vector $\mathbf{x}$ without variable x_0.

Next partition the integral into two integrals leveraging the fact that,

$$
\int_{x\in\mathcal{X}} f(x)dx = \int_{x\in\mathcal{X}_A} f(x)\mathrm{d}x + \int_{x\in\mathcal{X}_B} f(x)\mathrm{d}x,\ \mathcal{X} = \mathcal{X}_A \bigcup \mathcal{X}_B, \mathcal{X}_A \bigcap \mathcal{X}_B = \emptyset. \qquad (16)
$$

Here, the integral splits into subsets $\mathcal{X}_{x_0}$ representing the set of all bit vectors $\boldsymbol{x}$ with $x_0 = 1$, and respectively $\mathcal{X}_{\neg x_0}$ with all bit vectors $\boldsymbol{x}$ with $x_0 = 0$. Hence,

$$
\begin{aligned}
\mathcal{H}[\mathcal{C}(\boldsymbol{x})] = -\int_{\mathbf{x}\in\mathcal{X}_{x_0}} &\left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\mathbf{x}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\mathbf{x})\right] * \qquad (17)\\
&\log\left\{\left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\mathbf{x}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\mathbf{x})\right]\right\} \qquad (18)\\
-\int_{\mathbf{x}\in\mathcal{X}_{\neg x_0}} &\left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\mathbf{x}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\mathbf{x})\right] * \qquad (19)\\
&\log\left\{\left[w_{00}\mathbb{1}\{x_0 = 1\}\mathsf{S}_{01}(\mathbf{x}) + (1-w_{00})\mathbb{1}\{x_0 = 0\}\mathsf{S}_{00}(\mathbf{x})\right]\right\}. \qquad (20)
\end{aligned}
$$

Now, each integral can be simplified since the indicator functions will always evaluate to 0 or 1 in the respective subsets of $\mathcal{X}$, i.e.,

$$
\begin{aligned}
\mathcal{H}[\mathcal{C}(\boldsymbol{x})] = &-\int_{\mathbf{x}\in\mathcal{X}_{x_0}} w_{00}\mathsf{S}_{01}(\mathbf{x}) \log\left\{\left[w_{00}\mathsf{S}_{01}(\mathbf{x})\right]\right\} \qquad (21)\\
&-\int_{\mathbf{x}\in\mathcal{X}_{\neg x_0}} \left[(1-w_{00})\mathsf{S}_{00}(\mathbf{x})\right] * \log\left\{(1-w_{00})\mathsf{S}_{00}(\mathbf{x})\right\}. \qquad (22)
\end{aligned}
$$

As the two integrals have the same form, for notational simplicity we will only consider the first integral (in orange). The second integral can be computed in the same way.

$$
\begin{aligned}
& -\int_{\mathbf{x}\in\mathcal{X}_{x_0}} w_{00}\mathsf{S}_{01}(\mathbf{x}) \log \left\{[w_{00}\mathsf{S}_{01}(\mathbf{x})]\right\} && (23)\\
&= -\int_{\mathbf{x}\in\mathcal{X}_{x_0}} w_{00}\mathsf{S}_{01}(\mathbf{x}) \left[\log w_{00} + \log\left(\mathsf{S}_{01}(\mathbf{x})\right)\right] && (24)\\
&= -\int_{\mathbf{x}\in\mathcal{X}_{x_0}} w_{00}\log(w_{00})\mathsf{S}_{01}(\mathbf{x}) + w_{00}\mathsf{S}_{01}(\mathbf{x}) * \log\left(\mathsf{S}_{01}(\mathbf{x})\right) && (25)\\
&= -w_{00}\log(w_{00}) \int_{\mathbf{x}\in\mathcal{X}_{x_0}} \mathsf{S}_{01}(\mathbf{x}) - w_{00}\int_{\mathbf{x}\in\mathcal{X}_{x_0}} \mathsf{S}_{01}(\mathbf{x}) * \log\left(\mathsf{S}_{01}(\mathbf{x})\right). && (26)
\end{aligned}
$$

Notice, that the second integral is the entropy of the sum node S_{01}. Hence,

$$
= -w_{00}\log(w_{00}) \int_{\mathbf{x}\in\mathcal{X}_{x_0}} \mathsf{S}_{01}(\hat{\mathbf{x}}) + w_{00}\mathcal{H}(\mathsf{S}_{01}(\hat{\boldsymbol{x}})). \tag{27}
$$

Furthermore, if S_{01} is normalized, then $\int_{\mathbf{x}\in\mathcal{X}_{x_0}} \mathsf{S}_{01}(\mathbf{x})\mathrm{d}\boldsymbol{x} = 1$, leading to the further simplification,

$$
= -w_{00}\log(w_{00}) + w_{00}\mathcal{H}(\mathsf{S}_{01}(\hat{\boldsymbol{x}})). \tag{28}
$$

D.2. Reverse KL Divergence Calculation

Let us define a density q and a density p. The reverse KL divergence of q from p is denoted as $\mathrm{KL}(q \,\|\, p)$, and defined as:

$$
\begin{aligned}
\mathrm{KL}(q \,\|\, P) &= \int q(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \mathrm{d}\boldsymbol{x} && (29)\\
&= -\int q(\boldsymbol{x}) \log q(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} + \int q(\boldsymbol{x}) \log p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}. && (30)
\end{aligned}
$$

Note that $-\int q(\boldsymbol{x}) \log q(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$ is the entropy of distribution q, and will be denoted as $-\mathcal{H}(q)$:

$$
\mathrm{KL}(q \,\|\, p) = -\int q(\boldsymbol{x}) \log p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} - \mathcal{H}\left(q\right). \tag{31}
$$

Note also that $\int q(\boldsymbol{x}) \log p(\boldsymbol{x})\, \mathrm{d}$ is the expected value of the log-likelihood of p w.r.t. q:

$$
\mathrm{KL}(q \,\|\, p) = -\mathbb{E}_{\boldsymbol{x}\sim q}\left[\log p(\boldsymbol{x})\right] - \mathcal{H}\left(q\right). \tag{32}
$$

Appendix E. Experimental Details

E.1. 2D Densities

We present results for 2D non-Gaussian target distributions. In Fig. 8, we include additional results for typical benchmark target densities (mixture, Neal's funnel, two-modal Gaussian, ring, and banana) that we approximate with 4-bit/8-bit BitVI, which captures the overall density and cross-dependencies well.

E.2. MLP Neural Network Models

The experiments with the *Bayesian-benchmarks* data sets used the following hyperparameters and setup:

- Adam optimizer with a learning rate of 0.001
- Hidden layer size 16×16 for $D \leq 500$ and 32×32 for $D > 500$
- Batch size of 32 for $D \leq 500$ and 128 for $D > 500$
- 64 samples for computing the Monte Carlo approximation of the posterior log-joint
- Weight representations used 2 integer bits, except for the 2-bit model which used 0 integer bits
- LayerNorm (Ba et al., 2016) applied to hidden layers (pre-activation)
- Depth-based regularization for circuit parameters $\epsilon\, d^2$ with $\epsilon = 0.1$
- Early stopping based on the validation set ELBO loss after 2000 epochs
- Circuit weights were initialized from a beta distribution based on the height of the sum node in the circuit. The beta distribution α and β were set as 2^h where h is the height of the sum node in the circuit.
- 5-fold cross-validation into train and test sets
- Validation set split from the train set with 20% of the train set data

E.3. Ablation Studies

Banana Chopping

- Training set of 2048 points
- Validation set of 512 points
- Adam optimizer with a learning rate of 0.01
- Batch size of 256
- LayerNorm (Ba et al., 2016) applied to hidden layers (pre-activation)
- Weight representations used 10 bits with no integer bits. A sign bit and 9 fractional bits.
- Depth-based regularization for circuit parameters $\epsilon\, d^2$ with $\epsilon = 0.001$.
- Circuit weights were initialized from a beta distribution based on the height of the sum node in the circuit. The beta distribution α and β were set as 2^h where h is the height of the sum node in the circuit.

Appendix F. Additional Results

The following section contains additional results.

F.1. Ablation Studies

Increasing Complexity of Target Distribution We consider an ablation study where we control the target distribution complexity. For this, we constructed a mixture of equidistant Gaussians and assessed the entropy of BitVI under varying number of bits under three different amounts of variance for each Gaussian. Fig. 9 shows the fitted results of BitVI (black) with 16 bits for target distributions with increasing complexity (gray) alongside the entropy of BitVI under varying number of bits. The entropy (lower figures) shows the cut-off for number of bits needed to represent each target, indicating that BitVI naturally exhibits a parsimonious behaviour.

F.2. Trade-off Between Model Complexity and Bitstring Depth

In relation to NN applications, an interesting question is whether we actually need fine-grained numerical accuracy for representing the model weights in the first place. Recent advances in large-scale model training an inference suggest that rather than numerical accuracy, the models benefit from more parameters which enable further flexibility. In relation to this question, we study whether the models benefit from higher numerical granularity w.r.t. probabilistic treatment.

In Table 2, we vary both the neural network complexity (units in the two hidden layers) and the bitstring depth. We considser 2–12 bit models (with only fractional bits). The negative log predictive density (NLPD, smaller better) on the two moons data suggests that even low bit depth models perform well and the dominating factor in expressivity is the number of units in the NN. In App. F, we include similar tables for both accuracy and expected calibration error (ECE).

Do Bitstrings Capture Hierarchies in NNs? Finally, we use a neural network model to study the hierarchies captured by BitVI. We start from a 10-bit NN BitVI results on the Banana binary classification data set and gradually decrease the fractional precision of the trained model, chopping off more granular levels of the model. Fig. 10 shows the results for 10, 8, 6, 4, and 2 bit models (2 integer bits each, except for the 2-bit model). Even the 4-bit model (2 integer bits and 1 fractional bits) captures the overall structure well, whereas the 2-bit model (with no integer bits; only a sign bit and a fraction bit) struggles.

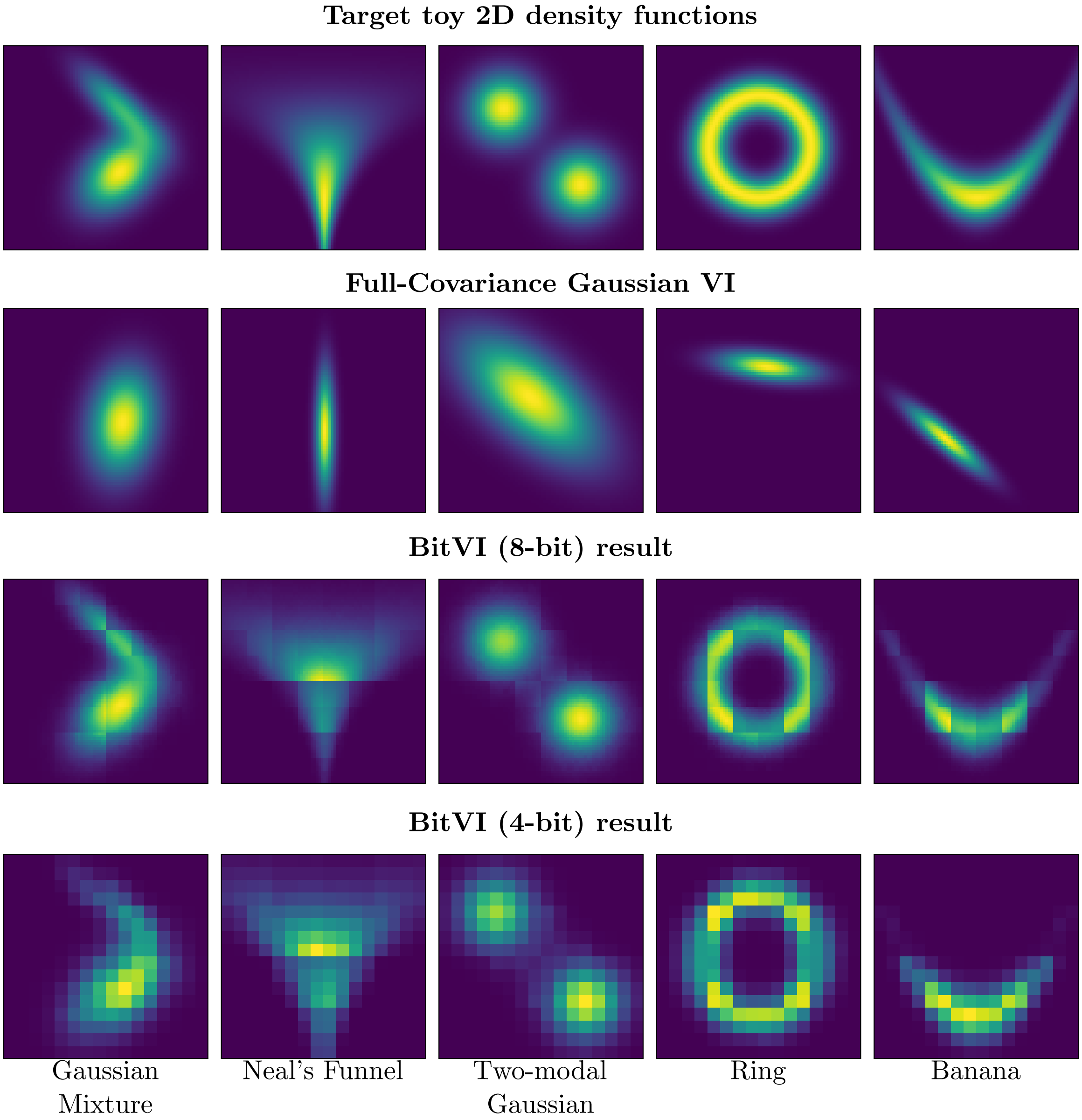


Figure 8: 2D non-Gaussian target distributions. We include results for typical benchmark target densities (mixture, Neal's funnel, two-modal Gaussian, ring, and banana) that we approximate with 4-bit/8-bit BitVI, which captures the overall density and cross-dependencies well.

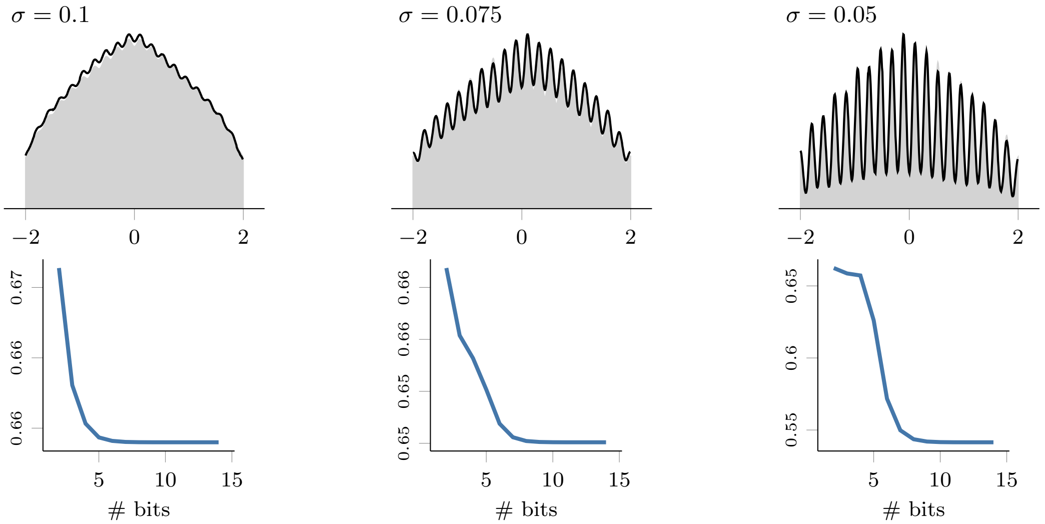


Figure 9: Ablation result of BitVI (black) for target distributions with increasing complexity (gray) and the precision used by the variational distribution to represent the target. The entropy (lower figures) shows the cut-off for bit-string depth needed to represent each target.

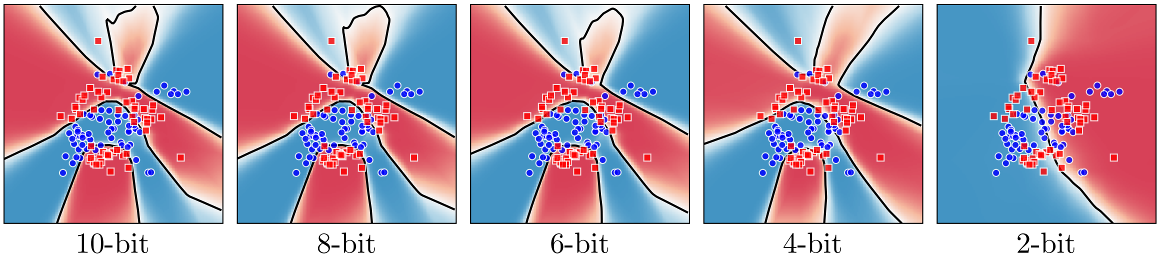


Figure 10: **Chopping the banana:** We start from a 10-bit NN BitVI results on the Banana binary classification data set and gradually decrease the fractional precision of the trained model. The low-bit models up to 4 bits capture the overall structure well. This is further confirmed by the results in Table 2.

Table 2: Trade-off between NN model complexity (units in hidden layers) and bitstring depth (2–12 bits). The negative log predictive density (NLPD, smaller better) on the two moons data suggests that even low bit depth models perform well and the dominating factor in expressivity is the number of units in the NN. See App. F for ACC/ECE.

	Increasing NN complexity →						
	[4, 4]	[6, 6]	[8, 8]	[10, 10]	[12, 12]	[14, 14]	[16, 16]
Bitstring depth 2	0.36	0.35	0.35	0.32	0.33	0.3	0.29
3	0.37	0.36	0.26	0.34	0.27	0.24	0.25
4	0.38	0.32	0.31	0.3	0.27	0.28	0.24
5	0.35	0.32	0.36	0.29	0.27	0.25	0.25
6	0.34	0.34	0.37	0.3	0.28	0.25	0.24
7	0.31	0.3	0.3	0.26	0.28	0.25	0.24
8	0.33	0.31	0.25	0.3	0.29	0.26	0.26
9	0.36	0.32	0.32	0.33	0.26	0.23	0.25
10	0.33	0.35	0.3	0.3	0.25	0.26	0.24
12	0.37	0.29	0.35	0.35	0.26	0.27	0.24

Table 3: Trade-off between NN model complexity (units in hidden layers) and bitstring depth (2–12 bits). Accuracy and expected calibration error (ECE) on the two moons data suggests that even low bit depth models perform well and the dominating factor in expressivity is the number of units in the NN. See Table 2 in the main paper for the NLPD.

Table 4: Accuracy

	[4, 4]	[6, 6]	[8, 8]	[10, 10]	[12, 12]	[14, 14]	[16, 16]
2	0.856	0.85	0.852	0.87	0.868	0.888	0.89
3	0.854	0.857	0.903	0.865	0.897	0.909	0.906
4	0.852	0.879	0.88	0.884	0.904	0.898	0.909
5	0.86	0.877	0.854	0.895	0.901	0.909	0.913
6	0.863	0.861	0.853	0.887	0.895	0.91	0.906
7	0.882	0.883	0.888	0.899	0.896	0.909	0.905
8	0.874	0.877	0.909	0.884	0.886	0.909	0.904
9	0.864	0.873	0.873	0.877	0.897	0.914	0.909
10	0.87	0.862	0.886	0.884	0.912	0.899	0.909
12	0.852	0.888	0.863	0.863	0.898	0.895	0.908

Table 5: ECE

	[4, 4]	[6, 6]	[8, 8]	[10, 10]	[12, 12]	[14, 14]	[16, 16]
2	0.059	0.053	0.061	0.064	0.065	0.055	0.06
3	0.06	0.071	0.051	0.053	0.046	0.042	0.049
4	0.064	0.064	0.055	0.045	0.042	0.045	0.038
5	0.061	0.057	0.058	0.053	0.042	0.043	0.044
6	0.06	0.067	0.057	0.048	0.044	0.039	0.046
7	0.063	0.053	0.048	0.046	0.041	0.042	0.046
8	0.061	0.051	0.038	0.047	0.048	0.045	0.042
9	0.055	0.056	0.053	0.053	0.045	0.04	0.042
10	0.057	0.058	0.056	0.05	0.037	0.047	0.046
12	0.064	0.05	0.055	0.053	0.041	0.05	0.045