Quantifying policy uncertainty in generative flow networks with uncertain rewards

Ramón Nartallo-Kaluarachchi[1,2], **Robert Manson-Sawko**[1], **Shashanka Ubaru**[3], **Dongsung Huh**[4], **Malgorzata J. Zimoń**[1,5], **Lior Horesh**[3]
1. IBM Research Europe, Daresbury, United Kingdom
2. Mathematical Institute, University of Oxford, United Kingdom
3. IBM Research, Thomas J. Watson Research Center, USA
4. MIT-IBM Watson AI Lab, Massachusetts, USA
5. Department of Mathematics, University of Manchester, United Kingdom

Abstract

Generative flow networks are able to sample, via sequential construction, high-reward, complex objects according to a reward function. However, such reward functions are often approximated from noisy data leading to epistemic uncertainty in the learnt policy. We present an approach to quantify this uncertainty by constructing a surrogate model composed of a polynomial chaos expansion, fit on a small ensemble of trained flow networks. This model learns the relationship between reward functions, parametrised in a low-dimensional space, and the probability distributions over actions at each step along a trajectory. The surrogate model can then be used for inexpensive Monte Carlo sampling to estimate the uncertainty in the policy given uncertain rewards. We illustrate the performance of our approach on a Bayesian structure learning task, and compare it to a basic multilayer perceptron.

1 Introduction

Generative artifical intelligence (AI) describes the subclass of techniques which learn a probabilistic model of the data-generating process. This yields advantages such as the ability to sample new data and to estimate the confidence in a model's output. It is common to describe such models as 'uncertainty aware', as they are typically able to compute the likelihood of each sample. This *aleatoric uncertainty* in a model's prediction or output is that which stems from the probabilistic process that the model has learnt, but assumes that the generative model is exactly correct. However, models learn a generative process from finite data or through self-play, via stochastic optimisation. This implies an additional form of uncertainty at the level of the model itself, so-called *epistemic uncertainty*, as a variety of generating processes can be learnt given alternative data or realisations of a stochastic input [7]. Quantifying this model uncertainty is key to understanding the capabilities and limitations of a given generative model and its predictions [4].

A similar situation is common in a variety of computational engineering problems with stochastic input [5], where repeated, high-fidelity simulations are computationally intractable, yet an understanding of the output uncertainty is desired. Such problems have led to the development of the field of uncertainty quantification (UQ) [23]. A cornerstone approach in UQ is to construct computationally-efficient 'surrogate' models using an array of techniques such as polynomial chaos expansions (PCEs) [26, 28], machine learning (ML) models [5, 25], Gaussian processes [20], or a combination of these approaches [21]. Typically, these methods approximate an input-output relationship by fitting to a limited number of examples obtained from the high-fidelity, 'black-box' model. Finally, performing

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: ML×OR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making.

Monte-Carlo (MC) sampling from the surrogate model allows model uncertainty to be quantified efficiently and inexpensively [23, 24]. Similarly, the training procedure of a large ML model operates as a black-box. Repeated training of models with different input data, spanning the full distribution, is akin to MC sampling from a high-fidelity model, and therefore computationally prohibitive. To address this, we investigate surrogate-modelling approaches for generative models, focusing on generative flow networks (GFNs) in situations where the reward is uncertain, and illustrate this with an example from Bayesian structure learning [6]. This paper is structured as follows. We present a brief introduction to GFNs with uncertain rewards, as well as to PCE. We then illustrate our approach on the problem of learning a linear Gaussian network.

2 Generative flow networks with uncertain rewards

GFNs are a novel and powerful architecture at the interface between generative and reinforcement learning [2], which have been used for generating molecular structures [1], solving combinatorial optimisation problems [29], Bayesian structure learning [6], and biological sequence design [13]. Variants exist for both continuous reward functions [16] and stochastic transition graphs [19]. In most applications, such as molecular candidate sampling or Bayesian structure learning, the reward function is obtained from noisy experimental data, and/or expressed by a neural network (NN) [1, 6]. This leads to epistemic uncertainty that fails to be expressed in the GFN. Two well-known paradigms for quantifying such uncertainty in ML are the use of Bayesian neural networks, [3, 18], or MC sampling - these approaches are often combined [8, 17]. Whilst there have been some attempts to encode such uncertainty in GFNs [13, 30], Bayesian methods fail to scale to large models, whilst the MC approach is prohibitively slow. As a result, this remains an open problem.

Generative flow networks A *Markovian flow network* is defined by a pair (G, F), where $G = (\mathcal{S}, \mathcal{E})$ is a directed acyclic graph (DAG) with each vertex representing a state $s \in \mathcal{S}$, and where each edge $e = (s \to s') \in \mathcal{E}$ is a directed connection representing a possible transition between states, a so-called *action*. The *flow function* $F : \mathcal{T} \to \mathbb{R}^+$ is a positively-valued function, defined on $\mathcal{T}$, the set of all trajectories through the state-space DAG. In order for a function F to define a valid flow network, it must ensure the *flow matching condition*,

$$F(s) = \sum_{(s'' \to s) \in \mathcal{E}} F(s'' \to s) = \sum_{(s \to s') \in \mathcal{E}} F(s \to s'), \tag{1}$$

is satisfied for all $s \in \mathcal{S}$. The set of states for which there are no possible forward transitions is denoted *terminating states*, $\mathcal{S}_f \subseteq \mathcal{S}$. A valid flow function yields a stochastic process over $\mathcal{S}$ with forward and backward transition probabilities given by,

$$P_f(s'|s) = \frac{F(s \to s')}{\sum_{s'':(s \to s'') \in \mathcal{E}} F(s \to s'')}, \qquad P_b(s'|s) = \frac{F(s' \to s)}{\sum_{s'':(s'' \to s) \in \mathcal{E}} F(s'' \to s)}, \tag{2}$$

which, given the state s, are the probabilities of transitioning to state s' forwards, P_f, or backwards, P_b, respectively. Finally, we define $P_T(s)$ to be the *terminating* probability of each terminating state $s \in \mathcal{S}_f$, i.e. the probability of a trajectory ending in that terminating state [2].

A *reward function*, $R : \mathcal{S}_f \to \mathbb{R}^+$, defined on the terminating states, determines the total amount of flow in the network, $Z = \sum_{s \in \mathcal{S}_f} R(s)$. With the constraint that $F(s) = R(s)$ for $s \in \mathcal{S}_f$, then $P_T(s) = R(s)/Z \propto R(s)$ for $s \in \mathcal{S}_f$ i.e. a valid flow will sample terminating states in proportion to their relative reward [2]. Given a DAG and a target reward function, $R(s)$, we use a NN to approximate the flow function by minimising the error in some flow consistency condition. For further details on flow networks and the loss functions derived from flow consistency conditions, see App. A.

Uncertain reward functions and uncertain policies We focus on the situation of an uncertain reward function sampled from a distribution, $R \sim \mathcal{R}(\hat{R})$, where $\hat{R}$ is the true (expected) reward. This variation arises from epistemic uncertainty in the reward itself, such as rewards which are calculated from finite data. A GFN trained on a sampled reward function learns the flow $F_\theta(\cdot|R)$ which is a sample from the distribution over possible flow functions given a particular architecture, training

objective, and reward function. The goal of UQ is to build up a picture of the marginal distribution over $\mathcal{R}$, for a fixed architecture and objective. The space of all possible trajectories is prohibitively large; thus, we focus on the distribution *along a trajectory*. Given a trajectory, $\tau = s_0 \to ... \to s_n$, we estimate the distribution and expected value of the the policy along that trajectory, which defines a collection of random variables (RV) $\{P_\theta(\cdot|s_t), t = 0, 1, ..., n-1\}$, where each is a distribution over the set of actions available at s_t. We say that the distribution $P(\cdot|s_t)$ is a sample from the distribution over policies $\mathcal{P}(s_t)$.

3 Surrogate modelling with polynomial chaos expansions

In order to perform surrogate modelling, we assume that there is a mapping $\Lambda_t : \text{supp}(\mathcal{R}) \to \text{supp}(\mathcal{P}(s_t))$, between a reward function and the policy at each step in the trajectory. We approximate Λ_t with a flexible model that is easy to fit using only a few realisations, and which allows for fast sampling, eschewing the need to retrain a great number of GFNs. However, $\text{supp}(\mathcal{R})$ is a space of functions, which may be high-dimensional and challenging to parametrise. When an obvious parameterisation is not available, we must learn a latent representation of the input space using dimensionality reduction such as principal component analysis (PCA), or for nonlinear data, a variational autoencoder (VAE). This defines a mapping $\phi : \Gamma \to \text{supp}(\mathcal{R})$ where $\phi(\boldsymbol{\mu}) = R$ and $\boldsymbol{\mu} \in \Gamma$ is a (low-dimensional) set of parameters that represent the reward function. The surrogate model then learns the mapping $\Lambda_t \circ \phi(\boldsymbol{\mu}) = P(\cdot|s_t)$, which maps between the input space Γ where each point represents a reward function, and the set of policies at step s_t.

Polynomial chaos expansions To approximate this mapping between rewards and policies, we use a PCE. A random variable, $Y \in \mathbb{R}$, can be expressed as a polynomial function of a random vector, $\boldsymbol{X} \in \mathbb{R}^m$, where the polynomial basis is orthogonal with respect to the distribution of the random vector, $Y = \sum_{j \in \mathbb{N}^m} c_{\mathbf{j}} \varphi_{\mathbf{j}}(\boldsymbol{X})$. The $c_{\mathbf{j}}$ are coefficients and $\varphi_{\mathbf{j}}$ form a basis of polynomials that satisfy the orthonormality condition (see App. B) [23, 28]. Given input-output data $(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^m$ and $\mathbf{Y} = (y_1, ..., y_n)$ with $y_i \in \mathbb{R}$, a surrogate model can be constructed by estimating the coefficients $c_{\mathbf{j}}$ which best fit this data. A simple approach to fitting these coefficients is to perform 'least-squares' regression or regularise the coefficients with common alternatives such as LASSO or ridge regression [11]. PCEs carries the advantage that the surrogate model has an interpretable functional form, allowing for sensitivity analysis through the calculation of Sobol indices [22]. For further details on PCE, see App. B.

4 Example: Bayesian structure learning a linear Gaussian network

A Bayesian network (BN) is a DAG which shows the direct dependence relationships between random variables [15]. This is a general representation of a system that can be applied to a wide array of data-sets in various scientific fields, helping to unravel the causal mechanisms within a system. However, reconstructing the dependence graph from finite observed data is a challenging and open problem. Deleu et al. [6] present an approach for using GFNs to sample candidate BNs in proportion with their likelihood. They illustrate this on a linear Gaussian model using an Erdös-Rényi graph, as well as on flow cytometry data. The reward function is given by the Bayesian Gaussian equivalent (BGe) score, which computes the likelihood of the data given a candidate graph [6, 15]. As a result, each finite data-set yields an uncertain reward function that is sampled from the distribution $R \sim \mathcal{R}(\hat{R})$.

Due to both the generality and importance of Bayesian structure learning, and the nature of uncertainty in the rewards, we choose to illustrate our surrogate modelling approach on this example. First, we train two ensembles of 250 GFNs each, namely a 'training' and 'testing' set. Each model is trained with respect to a reward function calculated using the BGe score associated with a (different) dataset of 100 random samples from the (same) linear Gaussian network with 5 variables (see Fig. 1). To parametrise these reward functions in a low-dimensional space, we start by calculating the $r-$matrix[1] for each data-set. This is flattened to give a 25-d vector that represents each reward function. We

[1]The $r-$matrix is a Bayesian form of the covariance matrix associated with the BGe score [9]. In our case, this is a 5×5 matrix that is flattened before performing PCA.

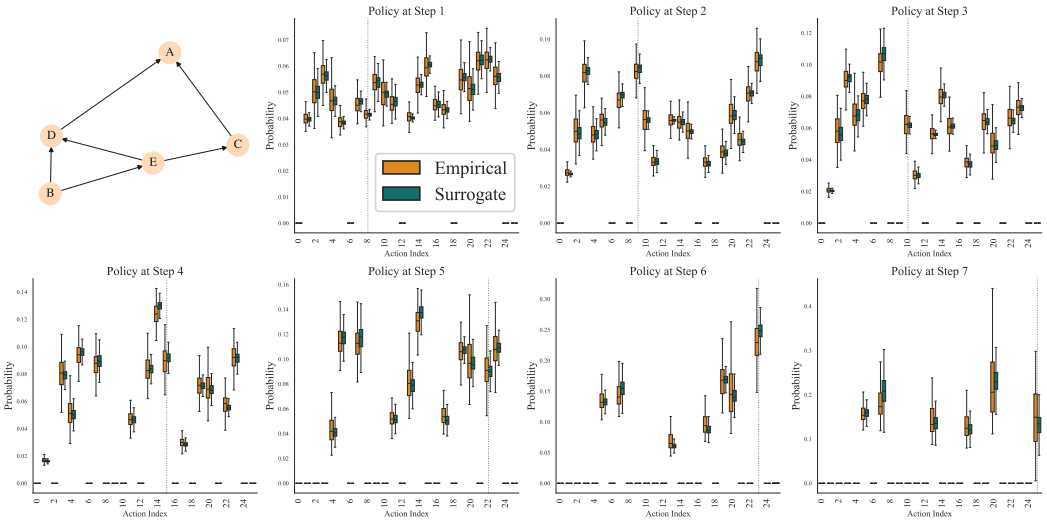

Figure 1: **UQ for a GFN learning a linear Gaussian network**. *Top left*. The ground-truth Gaussian network. *Panels 2-8*. We show the policy, a distribution over 26 actions, for each of 7 steps along a trajectory. The policy distribution of the 250 empirical testing samples is shown in orange, whilst the 10,000 surrogate samples are in teal. The dotted line shows the action taken at each step. Outlier values beyond fences are not shown, but full scatters are presented in Fig. 3.

then perform PCA, to extract the first two components, thus each reward function is mapped to a point in 2-d. This distribution in $\mathbb{R}^2$ is well approximated by a mean-zero Gaussian with independent components, where we estimate the variance in each direction. Next, we consider a sequence of actions that constructs the ground-truth graph[2]. Each action adds an edge to the graph, with cycles being prohibited. We map between a $(\texttt{source}, \texttt{target})$ edge and an $\texttt{action}$ using,

$$\texttt{action} = 5 * \texttt{source} + \texttt{target}, \tag{3}$$

where $\texttt{source}, \texttt{target} \in \{0, 1, 2, 3, 4\}$ and $\texttt{action} \in \{0, ..., 25\}$ with $\texttt{action} = 25$ corresponding to a transition to the terminal state. We extract the policy at each step along the action sequence to yield a tensor $\boldsymbol{P} \in \mathbb{R}^{250\times7\times26}$, where 7 is the number of actions needed to build the ground-truth graph shown in the first panel of Fig. 1. For each of the 7×26 variables, we fit a degree 7 PCE with Hermite polynomials using ridge regression using the 250 training samples as the input-output data. We then sample 10,000 additional datasets from the linear Gaussian network, project them into the latent space, and use the PCE to predict the policy at these additional inputs. As the probability values are in $[0, 1]$, but the PCE is unbounded, we instead fit to the logit transform of the $\boldsymbol{P}$ values and then transform the predictions back to $[0, 1]$ using the soft-max (see App. D). Fig. 1 shows the distribution of the policy for the 250 empirical testing samples and 10,000 surrogate samples for each of the 7 steps along the trajectory. Our surrogate model allows us to get a more complete estimate of the distribution of the policy stemming from uncertainty in the reward calculated from finite data. In App. E, we compare this approach to a multi-layer perceptron (MLP).

Our approach shows that surrogate modelling can be used to effectively compute policy uncertainty in a GFN with uncertain rewards. In future work, it would be interesting to focus on more complex examples, such as learning molecular structures, where reward functions are represented by NNs, and cannot be embedded with linear methods, but through unsupervised learning with VAEs. Additionally, focusing on the disentanglement capabilities of VAEs [30], we will investigate the use of adaptive polynomial degrees along latent dimensions, as well as sensitivity analysis using Sobol indices [22]. These methods could be incorporated into sampling algorithms that are 'epistemic-uncertainty-aware'. Finally, whilst we have focused on GFNs, such methods could be adapted for other generative models in the presence of uncertain inputs.

[2]We could use any trajectory from the state-space DAG, but it is informative to see the policy along the desired solution. This is one trajectory that leads to the ground truth graph, but there are many equivalent ones i.e. we can add the same edges in a different order.

References

[1] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio. Flow network based generative models for non-iterative diverse candidate generation. In *35th Conference on Neural Information Processing Systems*, 2021.

[2] Y. Bengio, S. Lahlou, T. Deleu, E. J. Hu, M. Tiwari, and E. Bengio. GFlowNet foundations. *Journal of Machine Learning Research*, 24(210):1–55, 2023.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *32nd International Conference on Machine Learning*, 2015.

[4] T. Chakraborti, C. R. S. Banerji, A. Marandon, V. Hellon, R. Mitra, B. Lehmann, L. Bräuninger, S. McGough, C. Turkay, A. F. Frangi, G. Bianconi, W. Li, O. Rackham, D. Parashar, C. Harbron, and B. MacArthur. Personalized uncertainty quantification in artificial intelligence. *Nature Machine Intelligence*, 7:522–530, 2025.

[5] P. Conti, M. Guo, A. Manzoni, A. Frangi, S. L. Brunton, and J. N. Kutz. Multi-fidelity reduced-order surrogate modelling. *Proceedings of the Royal Society A*, 480(2283), 2024.

[6] T. Deleu, A. Góis, C. Emezue, M. Rankawat, S. Lacoste-Julien, S. Bauer, and Y. Bengio. Bayesian structure learning with generative flow networks. In *38th Conference on Uncertainty in Artificial Intelligence*, volume PMLR 180, page 518–528, 2022.

[7] Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[8] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *33rd International Conference on Machine Learning*, 2016.

[9] D. Geiger and D. Heckerman. Learning Gaussian networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 235–243, 1994.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[11] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009.

[12] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243, 1995.

[13] M. Jain, E. Bengio, A.-H. Garcia, J. Rector-Brooks, B. F. P. Dossou, C. Ekbote, J. Fu, T. Zhang, M. Kilgour, D. Zhang, L. Simine, P. Das, and Y. Bengio. Biological sequence design with GFlowNets. In *39th International Conference on Machine Learning*, 2022.

[14] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[15] N. K. Kitson, A. C. Constantinou, Z. Guo, Y. Liu, and K. Chobtham. A survey of Bayesian network structure learning. *Artificial Intelligence Review*, 56:8721–8814, 2023.

[16] S. Lahlou, T. Deleu, P. Lemos, D. Zhang, A. Volokhova, A. Hernández-García, L. N. Ezzine, Y. Bengio, and N. Malkin. A theory of continuous generative flow networks. In *40th International Conference on Machine Learning*, volume PMLR 202, 2023.

[17] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *31st Conference on Neural Information Processing Systems*, 2017.

[18] D. J. C. MacKay. Probable networks and plausible predictions-a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(469), 1995.

[19] L. Pan, D. Zhang, M. Jain, L. Huang, and Y. Bengio. Stochastic generative flow networks. In *39th Conference on Uncertainty in Artificial Intelligence*, volume PMLR 202, 2023.

[20] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[21] P. F. Shustin, S. Ubaru, V. Kalantzis, L. Horesh, and H. Avron. PCENet: High dimensional surrogate modeling for learning uncertainty. *arXiv:2202.05063*, 2022.

[22] B. Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering and System Safety*, 93(7):964–979, 2008.

[23] T. Sullivan. *Introduction to Uncertainty Quantification*. Springer, 2015.

[24] R. K. Tripathy and I. Bilionis. Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375(15):565–588, 2018.

[25] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4:359–366, 2022.

[26] N. Wiener. The homogeneous chaos. *American Journal of Mathematics*, 60(4):897–936, 1938.

[27] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, 2010.

[28] D. Xiu and G. E. Karniadakis. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM Journal of Scientific Computing*, 24(2), 2002.

[29] D. Zhang, H. Dai, N. Malkin, A. Courville, Y. Bengio, and L. Pan. Let the flows tell: Solving graph combinatorial problems with GFlowNets. In *37th Conference on Neural Information Processing Systems*, 2023.

[30] D. Zhang, L. Pan, R. T. Q. Chen, A. Courville, and Y. Bengio. Distributional GFlowNets with quantile flows. *Transactions on Machine Learning Research*, 2024.

Technical Appendices and Supplementary Material

Code availability

The code supporting these results is available at `https://github.com/rnartallo/uq4gfn`.

A Flow networks and loss functions

In Sec. 2, we briefly introduce the definition of a GFN. Here, we expand on this formulation, including specifying possible loss functions [2]. Previously, we defined the flow function, F, and introduced the flow matching condition, which we now derive more explicitly. In order for a function F to define a valid flow network, it must satisfy,

$$F(A) = \sum_{\tau \in A} F(\tau), \tag{4}$$

for all $A \subseteq \mathcal{T}$ [2]. From this definition, we can define the *flow through a state* $s \in \mathcal{S}$,

$$F(s) := \sum_{\tau \in \mathcal{T} : s \in \tau} F(\tau), \tag{5}$$

and the *flow through an edge* $(s \to s') \in \mathcal{E}$,

$$F(s \to s') := \sum_{\tau \in \mathcal{T} : s \to s' \in \tau} F(\tau). \tag{6}$$

In other words, this enforces that flows are 'consistent', i.e. that the flow into a state is equal to the flow out of the state. As a consequence, the *flow matching condition*,

$$F(s) = \sum_{(s'' \to s) \in \mathcal{E}} F(s'' \to s) = \sum_{(s \to s') \in \mathcal{E}} F(s \to s'), \tag{7}$$

is satisfied for all $s \in \mathcal{S}$.

A.1 Learning a flow approximation

Given a DAG and a target reward function, constructing a valid flow is a non-trivial problem. We define a flow parametrisation of the pair (G, R), which acts as a function approximator and seeks to minimise the error in some flow consistency condition. The simplest loss is derived directly from the *flow matching* condition in Eq. (1). The parameters are trained to minimise the loss,

$$\mathcal{L}_{\text{FM}}(s) = \left(\log \frac{\sum_{(s'' \to s) \in A} F_\theta(s'' \to s)}{\sum_{(s \to s') \in A} F_\theta(s \to s')} \right)^2, \tag{8}$$

with a similar objective pushing the in-flow into terminal states to equal their reward [1].
A second choice is the *detailed balance loss* [2]. A valid flow satisfies the well-known detailed balance construct for Markov chains given by,

$$F(s) P_f(s'|s) = F(s') P_b(s|s'), \tag{9}$$

which can be used to define the loss function,

$$\mathcal{L}_{\text{DB}}(s, s') = \left(\log \frac{F_\theta(s) P_f(s'|s;\theta)}{F_\theta(s') P_b(s|s';\theta)} \right)^2, \tag{10}$$

again with a similar condition on the terminal nodes [2].

B Polynomial chaos expansions

In Sec. 3, we briefly introduce PCE, which we elaborate on here. First, we take the PCE defined previously, $Y = \sum_{j \in \mathbb{N}^m} c_{\mathbf{j}} \varphi_{\mathbf{j}}(\boldsymbol{X})$, where the $\varphi_{\mathbf{j}}$ satisfy the orthonormality condition,

$$\int_{\mathcal{X}} \varphi_{\mathbf{i}}(\mathbf{x}) \varphi_{\mathbf{j}}(\mathbf{x}) \rho_{\boldsymbol{X}}(\mathbf{x}) = \delta_{\mathbf{ij}}. \tag{11}$$

Here $\rho_{\boldsymbol{X}}$ is the density of the random vector $\boldsymbol{X}$ over support $\mathcal{X}$. In practical settings, it is necessary to truncate the infinite polynomial basis at a particular *degree*, d. We denote by $\Theta_{d,m}$ the subset of $\mathbb{N}^m$ which corresponds to the numeration of terms in a truncation of the polynomial basis with m inputs and maximum-degree d. Given input-output data, we fit a PCE using regression by solving the optimisation problem,

$$\{c_{\mathbf{j}}\} = \text{argmin}_{\tilde{c}_{\mathbf{j}}} \sum_{k=1}^{n} ||y_k - \sum_{\mathbf{j} \in \Theta_{d,m}} c_{\mathbf{j}} \varphi_{\mathbf{j}}(\mathbf{x}_k)||^2. \tag{12}$$

Regression does not impose constraints on the *collocation points*, $\{\mathbf{x}_k\}_{k=1}^n$, unlike pseudo-spectral projection methods [27]. The coefficients can be regularised with techniques such as LASSO and ridge regression, which we opt for here [11].

C Bayesian structure learning example

In Sec. 4, we use the GFN model presented in Deleu et al. [6], to demonstrate our approach. Here we describe the example problem in greater detail. We focus on the problem of sampling candidate graphs using data sampled from a linear Gaussian network.

C.1 Linear Gaussian networks

A *linear Gaussian network* is a stochastic model defined by a causal DAG with adjacency matrix (A_{ij}). The model is defined by the relationship,

$$x_j = \sum_{x_i \in \text{Pa}(x_j)} \beta_{ij} x_i + \epsilon, \tag{13}$$

where x_j is a random variable and $\text{Pa}(x_j)$ are the set of its *parent nodes* i.e. the set of nodes k such that $k \to j$ is in the DAG; $\beta_{ij} \sim \mathcal{N}(0, 1)$ if $A_{ij} = 1$ and 0 otherwise; and $\epsilon \sim \mathcal{N}(0, 0.01)$ is Gaussian noise [6]. Given a dataset D and a candidate DAG, G, the reward is given by,

$$R(G) = P(G) P(D|G), \tag{14}$$

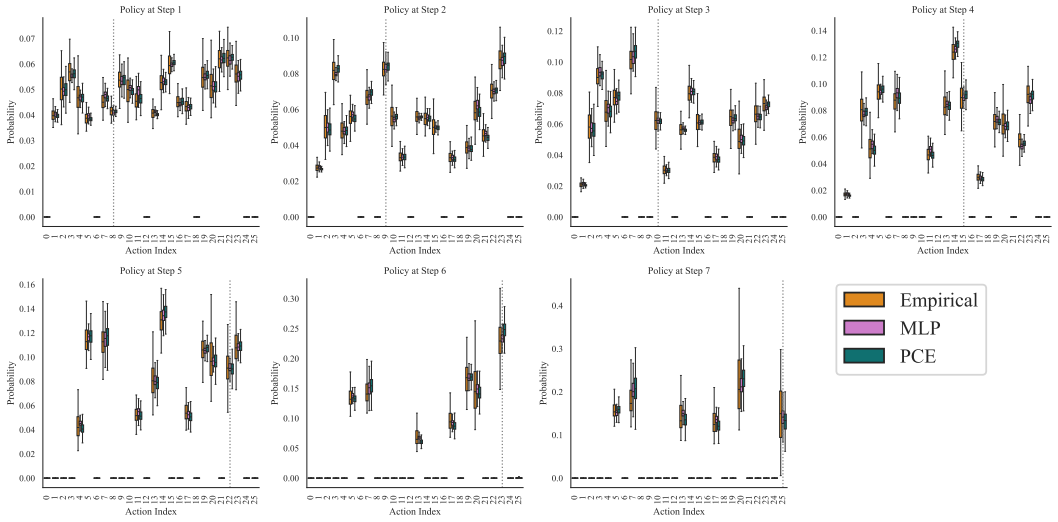

Figure 2: **Distribution comparison with a MLP.** We generate 10,000 samples using a MLP as a surrogate model and compare the empirical testing samples and the PCE surrogate. We find that both models are able to generate a realistic distribution of surrogate values.

where $P(G)$ is the prior over DAGs, and $P(D|G)$ is the marginal likelihood of the observed data given G. Calculating the marginal likelihood is challenging, thus we use the BGe score, under the assumption that the prior over both the parameters and structure of the DAG is *modular*. For further details, we direct the interested reader to Refs. [6, 9, 12]. In Sec. 4, we use a single underlying DAG with 5 nodes, shown in Fig. 1, and sample datasets with 100 points.

C.2 GFN model structure and architecture

We use the model from Deleu et al. [6]. Starting from the initial state s_0, a graph with no edges, graphs are constructed one edge at a time, masking actions that violate the DAG condition (see Appendix C of Ref. [6]). Each input graph is encoded as a set of edges, and each directed edge is embedded using an embedding for the source and target node, with an additional vector indicating the presence of edges in a graph G. The embeddings are passed through a linear transformer [14] with two output heads, the first of which gives the forward transition probabilities over possible actions, and the second gives the probability to terminate the trajectory [6]. For further details see Ref. [6].

D Logit transform and soft-max

In the BN example, the policy at each step is a distribution over 26 possible actions, which is represented by the vector $P \in [0,1]^{26}$ where $\sum_{i=1}^{26} P_i = 1$. As polynomial functions are not constrained to this range, we first transform P using the logit transform,

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right), \tag{15}$$

which takes values in $\mathbb{R}$, with $\text{logit}(0) = -\infty$. We then fit the PCE to the transformed values. When taking samples for the PCE model, we undo the transform using the soft-max function [10] to obtain a probability distribution over the 26 actions.

E Multilayer perceptron comparison

We compare our PCE approach to an alternative surrogate model using a MLP. We use a network with 2 input units, corresponding to our low-dimensional representation of the reward function, two hidden layers, each with 64 units, and a soft-max readout layer with 26 outputs. The network is trained to minimise the *Kullback-Leibler* divergence between the empirical and surrogate distributions. Fig. 2

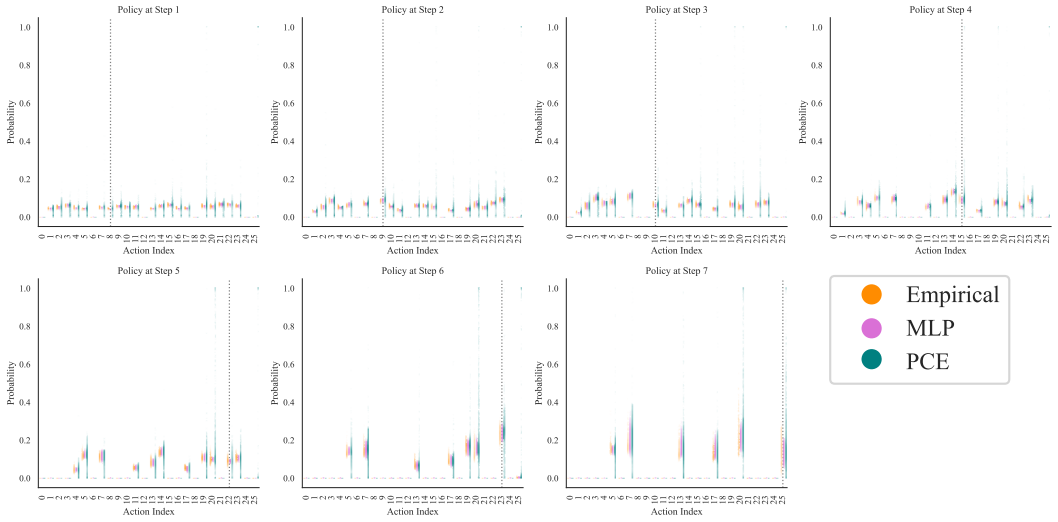

Figure 3: **Scatter comparison with a MLP.** We plot the same data as Fig. 2, but show the underlying points. We can see that the MLP suffers from over-shrinkage to the mean, a consequence of over-fitting, whereas the PCE captures a wider range of possible output values whilst still preserving the mean.

shows the box-plots of an empirical sample of 250 GFNs (test data) compared with 10,000 samples from the PCE and MLP models. We see that both surrogate models produce comparable distributions that are aligned with the testing data. Fig. 3 plots the scatter of these 10,000 points, showing that the MLP suffers from over-shrinkage to the mean, as it typically produces surrogate samples that are narrower or comparable to the empirical data. On the other hand, the PCE produces samples with a wider range (Fig. 3), yet still preserves the mean and standard deviation (Figs. 1, 2).