# $\pi$VAE: a stochastic process prior for Bayesian deep learning with MCMC

Swapnil Mishra[1,2*†], Seth Flaxman[3†], Tresnia Berah[4], Harrison Zhu[4], Mikko Pakkanen[4] and Samir Bhatt[1,2†]

[1*]MRC Centre for Global Infectious Disease Analysis, Jameel Institute for Disease and Emergency Analytics, School of Public Health, Imperial College London, London, UK.
[2]Section of Epidemiology, Department of Public Health, University of Copenhagen, Copenhagen, Denmark.
[3]Department of Computer Science, University of Oxford, Oxford, UK.
[4]Department of Mathematics, Imperial College London, London, UK.

*Corresponding author(s). E-mail(s): s.mishra@imperial.ac.uk;
†These authors contributed equally to this work.

**Abstract**

Stochastic processes provide a mathematically elegant way to model complex data. In theory, they provide flexible priors over function classes that can encode a wide range of interesting assumptions. However, in practice efficient inference by optimisation or marginalisation is difficult, a problem further exacerbated with big data and high dimensional input spaces. We propose a novel variational autoencoder (VAE) called the prior encoding variational autoencoder ($\boldsymbol{\pi}$VAE). $\boldsymbol{\pi}$VAE is a new continuous stochastic process. We use $\boldsymbol{\pi}$VAE to learn low dimensional embeddings of function classes by combining a trainable feature mapping with generative model using a VAE. We show that our framework can accurately learn expressive function classes such as Gaussian processes, but also properties of functions such as their integrals. For popular tasks, such as spatial interpolation, $\boldsymbol{\pi}$VAE achieves state-of-the-art performance both in terms of accuracy and computational efficiency. Perhaps most usefully, we demonstrate an elegant and scalable means of performing fully Bayesian inference for stochastic processes within probabilistic programming languages such as Stan.

**Keywords:** Bayesian inference, MCMC, VAE, Spatio-temporal

## 1 Introduction

A central task in machine learning is to specify a function or set of functions that best generalises to new data. Stochastic processes (Pavliotis, 2014; Ross, 1996) provide a mathematically elegant way to define a class of functions, where each element from a stochastic process is a (usually infinite) collection of random variables. Popular examples of stochastic processes in computational statistics and machine learning are Gaussian processes (Rasmussen & Williams, 2006), Dirichlet processes (Antoniak, 1974), log-Gaussian Cox processes (Møller, Syversveen, & Waagepetersen, 1998), Hawkes processes (Hawkes, 1971), Mondrian processes (Roy & Teh, 2009) and Gauss-Markov processes (Lindgren, Rue, & Lindström, 2011). Many of these processes are intimately connected with popular techniques in deep learning, for example, both the infinite width limit of a single layer neural network and the evolution of a deep neural network by gradient descent are Gaussian processes (Jacot, Gabriel, & Hongler, 2018; R. Neal, 1996). However, while stochastic processes have many favourable properties, they are often cumbersome to work with in practice. For example, inference and prediction using a Gaussian process requires matrix inversions that scale cubicly with data size, log-Gaussian Cox processes require the evaluation of an intractable integral and Markov processes are often highly correlated. Bayesian inference can be even more challenging due to complex high dimensional posterior topologies. Gold standard evaluation of posterior expectations is done by Markov Chain Monte Carlo (MCMC) sampling, but high autocorrelation, narrow typical sets (Betancourt, Byrne, Livingstone, & Girolami, 2017) and poor scalability have prevented use in big data and complex model settings. A plethora of approximation algorithms exist (Blundell, Cornebise, Kavukcuoglu, & Wierstra, 2015; Lakshminarayanan, Pritzel, & Blundell, 2017; Minka, 2001; Ritter, Botev, & Barber, 2018; Welling & Teh, 2011), but few actually yield accurate posterior estimates (Hoffman, Blei, Wang, & Paisley, 2013; Huggins, Kasprzak, Campbell, & Broderick, 2019; J. Yao, Pan, Ghosh, & Doshi-Velez, 2019; Y. Yao, Vehtari, Simpson, & Gelman, 2018). In this paper, rather than relying on approximate Bayesian inference to solve complex models, we extend variational autoencoders (VAE) (Kingma & Welling, 2014; Rezende, Mohamed, & Wierstra, 2014) to develop portable models that can work with state-of-the-art Bayesian MCMC software such as Stan (Carpenter et al., 2017). Inference on the resulting models is tractable and yields accurate posterior expectations and uncertainty.

An autoencoder (Hinton & Salakhutdinov, 2006) is a model comprised of two component networks. The encoder $e : \mathcal{X} \to \mathcal{Z}$ encodes inputs from space $\mathcal{X}$ into a latent space $\mathcal{Z}$ of lower dimension than $\mathcal{X}$. The decoder $d : \mathcal{Z} \to \mathcal{X}$ decodes latent codes in $\mathcal{Z}$ to reconstruct the input. The parameters of $e$ and $d$ are learned through the minimisation of a reconstruction loss on a training dataset. A VAE extends the autoencoder into a generative model (Kingma & Welling, 2014). In a VAE, the latent space $\mathcal{Z}$ is given a distribution, such as standard normal, and a variational approximation to the posterior is estimated. In a variety of applications, VAEs do a superb job reconstructing training datasets and enable the generation of new data: samples from the latent space are decoded to generate synthetic data (Kingma & Welling, 2019). In this paper we propose a novel use of VAEs: we learn low-dimensional representations of samples from a given function class (e.g. sample paths from a Gaussian process prior). We then use the resulting low dimensional representation and the decoder to perform Bayesian inference.

One key benefit of this approach is that we decouple the prior from inference to encode arbitrarily complex prior function classes, without needing to calculate any data likelihoods. A second key benefit is that when inference is performed, our sampler operates in a low dimensional, uncorrelated latent space which greatly aids efficiency and computation, as demonstrated in the spatial statistics setting in PriorVAE (Semenova et al., 2022). One limitation of this approach (and of PriorVAE) is that we are restricted to encoding finite-dimensional priors, because VAEs are not stochastic processes. To overcome this limitation, we take as inspiration the Karhunen-Loève decomposition of a stochastic process as a random linear combination of basis functions and introduce a new VAE called the prior encoding VAE ($\pi$VAE). $\pi$VAE is a valid stochastic process by construction, it is capable of learning a set of basis functions, and it incorporates a VAE, enabling simulation and highly effective fully Bayesian inference.

We employ a two step approach: first, we encode the prior using our novel architecture; second we use the learnt basis and decoder network—a new stochastic process in its own right—as a prior, combining it with a likelihood in a fully Bayesian modeling framework, and use MCMC to fit our model and infer the posterior. We believe our framework's novel decoupling into two stages is critically important for many complex scenarios, because we do not need to compromise in terms of either the expressiveness of deep learning or accurately characterizing the posterior using fully Bayesian inference.

We thus avoid some of the drawbacks of other Bayesian deep learning approaches which rely solely on variational inference, and the drawbacks of standard MCMC methods for stochastic processes which are inefficient and suffer from poor convergence.

Taken together, our work is an important advance in the field of Bayesian deep learning, providing a practical framework combining the expressive capability of deep neural networks to encode stochastic processes with the effectiveness of fully Bayesian and highly efficient gradient-based MCMC inference to fit to data while fully characterizing uncertainty.

Once a $\pi$VAE is trained and defined, the complexity of the decoder scales linearly in the size of the largest hidden layer. Additionally, because the latent variables are penalised via the KL term from deviating from a standard normal distribution, the latent space is approximately uncorrelated, leading to high effective sample sizes in MCMC sampling. The main contributions of this paper are:

- We apply the generative framework of VAEs to perform full Bayesian inference. We first encode priors in training and then, given new data, perform inference on the latent representation while keeping the trained decoder fixed.
- We propose a new generative model, $\pi$VAE, that generalizes VAEs to be able to learn priors over both functions and properties of functions. We show that $\pi$VAE is a valid (and novel) stochastic process by construction.
- We show the performance of $\pi$VAE on a range of simulated and real data, and show that $\pi$VAE achieves state-of-the-art performance in a spatial interpolation task.

The rest of this paper is structured as follows. Section 2 details the proposed framework and the generative model along with toy fitting examples. The experiments on large real world datasets are outlined in Section 3. We discuss our findings and conclude in Section 4.

# 2 Methods

## 2.1 Variational Autoencoders (VAEs)

A standard VAE has three components:

1. an encoder network $e(x, \gamma)$ which encodes inputs $x \in \mathcal{X}$ using learnable parameters $\gamma$,
2. random variables $z$ for the latent subspace,
3. a decoder network $d(z, \psi)$ which decodes latent embeddings $z$ using learnable parameters $\psi$.

In the simplest case we are given inputs $x \in \mathbb{R}^d = \mathcal{X}$ such as a flattened image or discrete time series. The encoder $e(x, \gamma)$ and decoder $d(z, \psi)$ are fully connected neural networks (though they could include convolution or recurrent layer). The output of the encoder network are vectors of mean and standard deviation parameters $z_\mu$ and $z_{sd}$. These vectors can thus be used to define the random variable $\mathcal{Z}$ for the latent space:

$$[z_\mu, z_{sd}]^\top = e(x, \gamma) \tag{1}$$

$$\mathcal{Z} \sim \mathcal{N}(z_\mu, z_{sd}^2 \mathbb{I}) \tag{2}$$

For random variable $\mathcal{Z}$, the decoder network reconstructs the input by producing $\hat{x}$:

$$\hat{x} = d(\mathcal{Z}, \psi) \tag{3}$$

To train a VAE, a variational approximation is used to estimate the posterior distribution

$$p(\mathcal{Z} \mid x, \gamma, \psi) \propto p(x \mid \mathcal{Z}, \gamma, \psi) \times p(\mathcal{Z})$$

The variational approximation greatly simplifies inference by turning a marginalisation problem into an optimisation problem. Following (Kingma & Ba, 2014), the optimal parameters for the encoder and decoder are found by maximising the evidence lower bound:

$$\underset{\gamma,\psi}{\arg\max} \quad \mathbb{E}_{\mathcal{Z}}\Big[\log p\left(x \mid \mathcal{Z}, \gamma, \psi\right) - \mathrm{KL}\left(\mathcal{Z} \parallel \mathcal{N}(0, \mathbb{I})\right)\Big] \tag{4}$$

The first term in Eq. (4) is the likelihood quantifying how well $\hat{x}$ matches $x$. In practice we can simply adopt the mean squared error loss directly, referred to as the reconstruction loss, without taking a probabilistic perspective. The second term is a Kullback-Leibler divergence to ensure that $\mathcal{Z}$ is as similar as possible to the prior distribution, a standard normal. Again, this second term can be specified directly without the evidence lower bound derivation: we view the KL-divergence as a regularization penalty to ensure that the latent parameters are approximately uncorrelated by penalizing how far they deviate from $\mathcal{N}(0, \mathbb{I})$.

Once training is complete, we fix $\psi$, and use the decoder as a generative model. To simplify subsequent notation we refer to a fully trained decoder as $d(z)$. Generating a new sample is simple: first draw a random variable $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I})$ and then apply the decoder, which is a deterministic transformation to obtain $d(\mathcal{Z})$. We see immediately that $d(\mathcal{Z})$ is itself a random variable. In the next section, we will use this generative model as a prior in a Bayesian framework by linking it to a likelihood to obtain a posterior.

## 2.2 VAEs for Bayesian inference

VAEs have been typically used in the literature to create or learn a generative model of observed data (Kingma & Welling, 2014), such as images. (Semenova et al., 2022) introduced a novel application of VAEs in a Bayesian inference setting, using a two stage approach that is closely related to ours. In brief, in the first stage, a VAE is trained to encode and decode a large dataset of vectors consisting of samples drawn from a specified prior $p(\theta)$ over random vectors. In the second stage, the original prior is replaced with the approximate prior: $\theta := d(\mathcal{Z})$ where $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I})$.

To see how this works in a Bayesian inference setting, consider a likelihood $p(y \mid \theta)$ linking the parameter $\theta$ to data $y$. Bayes' rule gives the unnormalized posterior:

$$p(\theta \mid y) \propto p(y \mid \theta) \times p(\theta) \tag{5}$$

The trained decoder serves as a drop-in replacement for the original prior class in a Bayesian setting:

$$p(\mathcal{Z} \mid y, d) \propto p(y \mid d(\mathcal{Z})) \times p(\mathcal{Z})\,. \tag{6}$$

The implementation within a probabilistic programming language is very straightforward: a standard normal prior and deterministic function (the decoder) are all that is needed.

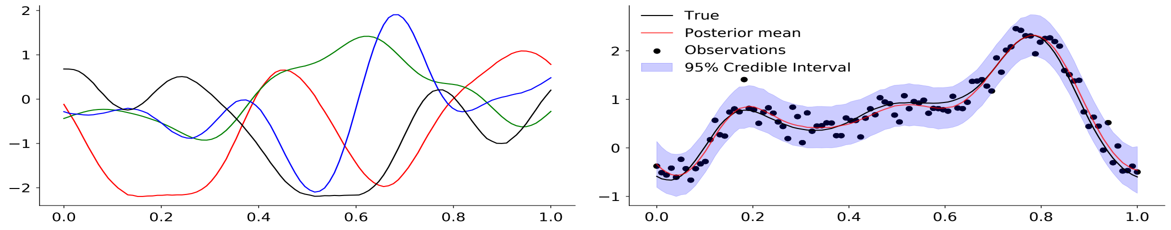
It is useful to contrast the inference task from Eq. (6) to a Bayesian neural network (BNN) (R. Neal, 1996) or Gaussian process in primal form (Rahimi & Recht, 2008). In a BNN with parameters $\omega$ and hyperparameters $\lambda$, the unnormalised posterior would be

$$p(\omega, \lambda \mid y) \propto p(y \mid \omega, \lambda) \times p(\omega \mid \lambda) \times p(\lambda)\,. \tag{7}$$

The key difference between Eq. (7) and Eq. (6) is the term $p(\omega \mid \lambda)$. The dimension of $\omega$ is typically huge, sometimes in the millions, and is conditional on $\lambda$, whereas in Eq. (6) the latent dimension of $\mathcal{Z}$ is typically small ($< 50$), uncorrelated and unconditioned. Full batch MCMC training is typically prohibitive for BNNs due to large datasets and the high-dimensionality of $\omega$, but approximate Bayesian inference algorithms tend to poorly capture the complex posterior (J. Yao et al., 2019; Y. Yao et al., 2018). Additionally, $\omega$ tends to be highly correlated, making efficient MCMC nearly impossible. Finally, as the dimension and depth increases, the posterior distribution suffers from complex multimodality, and concentration to a narrow typical set (Betancourt et al., 2017). By contrast, off-the-shelf MCMC methods are very effective for equation (6) because the prior space they need to explore is as simple as it could be: a standard normal distribution, while the complexity of the model lives within the deterministic (and differentiable) decoder. In a challenging spatial statistics setting, (Semenova et al., 2022) used this approach and achieved MCMC effective sample sizes *exceeding* actual sample sizes, due to the incredible efficiency of the MCMC sampler.

An example of using VAEs to perform inference is shown in Figure 1 where we train a VAE with latent dimensionality 10 on samples drawn from a zero mean Gaussian process with RBF kernel ($K(\delta) = e^{-\delta^2/8^2}$) observed on the grid $0, 0.01, 0.02, \ldots, 1.0$. In Figure 1 we closely recover the true function and correctly estimate the data noise parameter. Our MCMC samples showed virtually no autocorrelation, and all diagnostic checks were excellent (see Appendix). Solving the equivalent problem using a Gaussian process prior would not only be considerably more expensive ($\mathcal{O}(n^3)$) but correlations in the parameter space would complicate MCMC sampling and necessitate very long chains to achieve even modest effective sample sizes.

This example demonstrates the promise that VAEs hold to improve Bayesian inference by encoding function classes in a two stage process. While this simple example proved useful in some settings (Semenova et al., 2022), inference and prediction is not possible at new input locations, because a VAE is not a stochastic process. As described above, a VAE provides a novel prior over random vectors. Below, we take the next step by



(a)

(b)

**Fig. 1**: Learning functions with VAE: (a) Prior samples from a VAE trained on Gaussian process samples (b) we fit our VAE model to data drawn from a GP (blue) plus noise (black points). The posterior mean of our model is in red with the 95% epistemic credible intervals shown in purple.

introducing $\pi$VAE, a new stochastic process capable of approximating useful and widely used priors over function classes, such as Gaussian processes.

## 2.3 Encoding stochastic processes with $\pi$VAE

To create a model with the ability to perform inference on a wide range of problems we have to ensure that it is a valid stochastic process. Previous attempts in deep learning in this direction have been inspired by the Kolmogorov Extension Theorem and have focused on extending from a finite-dimensional distribution to a stochastic process. Specifically, (Garnelo et al., 2018) introduced an aggregation step (typically an average) to create an order invariate global distribution. However, as noted by (Kim et al., 2019), this can lead to underfitting.

We take a different approach with $\pi$VAE, inspired by the Karhunen-Loève Expansion (Karhunen, 1947; Loeve, 1948). Recall that a centered stochastic process $f(s)$ can be written as an infinite sum:

$$f(s) = \sum_{j=1}^{\infty} \beta_j \phi_j(s) \tag{8}$$

for pairwise uncorrelated random variables $\beta_j$ and continuous real-valued functions forming an orthonormal basis $\phi_j(s)$. The random $\beta_j$'s provide a linear combination of a fixed set of basis functions, $\phi_j$. This perspective has a long history in neural networks, cf. radial basis function networks.

What if we consider a trainable, deep learning parameterization of Eq. (8) as inspiration? We need to learn deterministic basis functions while allowing the $\beta_j$'s to be random. Let $\Phi(s)$ be a feature mapping with weights $w$, i.e. a feed-forward neural network architecture over the input space, representing the basis functions. Let $\beta$ be a vector of weights on the basis functions, so $f(s) = \beta^\top \Phi(s)$. We use a VAE architecture to encode and decode $\beta$, meaning we maintain the random variable perspective and at the same time learn a flexible low-dimensional non-linear generative model.

How can we specify and train this model? As with the VAE in the previous section, $\pi$VAE is trained on draws from a prior. Our goal is to encode a stochastic process prior $\Pi$, so we consider $i = 1, \ldots, N$ function realizations denoted $f_i(s)$. Each $f_i(s)$ is an infinite dimensional object, a function defined for all $s$, so we further assume that we are given a finite set of $K_i$ observation locations. We set $K_i = K$ for simplicity of implementation i.e. the number of evaluations for each function is constant across all draws $i$. We denote the observed values as $y_i^k := f_i(s_i^k)$. The training dataset thus consists of $N$ sets of $K$ observation locations and function values:

$$\{(s_i^1, y_i^1) \ldots, (s_i^K, y_i^K)\}_{i=1}^N$$

Note that the set of $K$ observation locations varies across the $N$ realizations.

We now return to the architecture of $\pi$VAE (Fig. 2). The feature mapping $\Phi(s)$ is shared across all $i = 1, \ldots, N$ function draws, so it consists of a feedforward neural network and is parameterized by a set of global parameters $w$ which must be learned. However, a particular random realization $f_i(s)$ is represented by a random vector $\beta_i$, for which we use a VAE architecture. We note the following non-standard setup: $\beta_i$ is a learnable parameter of our model, but it is also the input to the encoder of the VAE. The decoder attempts to reconstruct $\beta_i$ with an output $\hat{\beta}_i$. We denote the encoder and decoder as:

$$[z_\mu, z_{sd}]^\top = e(\beta, \gamma) \tag{9}$$

$$\mathcal{Z} \sim \mathcal{N}(z_\mu, z_{sd}^2 \mathbb{I}) \tag{10}$$

$$\hat{\beta} = d(\mathcal{Z}, \psi) \tag{11}$$

We are now ready to express the loss, which combines the two parts of the network, summing across all observations. Rather than deriving an evidence lower bound, we proceed directly to specify a loss function, in three parts. In the first, we use MSE to check the fit of the $\beta_i$'s and $\Phi$ to the data:

$$\text{Loss 1}: \frac{1}{NK}\sum_{i,k}(y_i^k - \beta_i^\top \Phi(s_i^k))^2$$

In the second, we use MSE to check the fit of the reconstructed $\hat{\beta}_i$'s and $\Phi$ to the data:

$$\text{Loss 2}: \frac{1}{NK}\sum_{i,k}(y_i^k - \hat{\beta}_i^\top \Phi(s_i^k))^2$$

We also require the standard variational loss:

$$\text{KL}\left(\mathcal{Z} \parallel \mathcal{N}(0, \mathbb{I})\right)$$

Note that we do not consider reconstruction loss $\|\beta_i - \hat{\beta}_i\|^2$ because in practice this did not improve training.

To provide more intuition: the feature map $\Phi(s)$ transforms each observed location to a fixed feature space that is shared for all locations across all functions. $\Phi(s)$ could be an explicit feature representation for an RKHS (e.g. an RBF network or a random Fourier feature basis (Rahimi & Recht, 2008)), a neural network of arbitrary construction or, as we use in the examples in this paper, a combination of both. Following this transformation, a linear basis $\beta$ (which we obtain from a non-linear decoder network) is used to predict function evaluations at an arbitrary location. The intuition behind these two transformations is to learn the association between locations and observations while allowing for randomness—$\Phi$ provides the correlation structure over space and $\beta$ the randomness. Explicit choices can lead to existing stochastic processes: we can obtain a Gaussian process with kernel $k(\cdot,\cdot)$ using a single-layer linear VAE for $\beta$ (meaning the $\beta$s are simply standard normals) and setting $\Phi(s) = L^\top s$ for $L$ the Cholesky decomposition of the Gram matrix $K$ where $K_{ij} = k(s_i, s_j)$.

In contrast to a standard VAE encoder that takes as input the data to be encoded, $\pi$VAE first transforms input data (locations) to a higher dimensional feature space via $\Phi$, and then connects this feature space to outputs, $y$, through a linear mapping, $\beta$. The $\pi$VAE decoder takes outputs from the encoder, and attempts to recreate $\beta$ from a lower dimensional probabilistic embedding. This re-creation, $\hat{\beta}$, is then used as a linear mapping with the *same* $\Phi$ to get a reconstruction of the outputs $y$. It is crucial to note that a single global $\beta$ vector is *not* learnt. Instead, for each function $i = 1, \ldots, N$ a $\beta_i$ is learnt.

In terms of number of parameters, we need to learn $w$, $\gamma$, $\psi$, $\beta_1, \ldots, \beta_N$. While this may seem like a huge computational task, $K$ is typically quite small ($< 200$) and so learning can be relatively quick (dominated by matrix multiplication of hidden layers). Algorithm 1 in the Appendix presents the step-by-step process of training $\pi$VAE.

### 2.3.1 Simulation and Inference with $\pi$VAE

Given a trained embedding $\Phi(\cdot)$ and trained decoder $d(z)$, we can use $\pi$VAE as a generative model to simulate sample paths $f$ as follows. A single function $f$ is obtained by first drawing $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I})$ and defining $f(s) := d(\mathcal{Z})^\top \Phi(s)$. For a fixed $\mathcal{Z}$, $f(s)$ is a deterministic function—a sample path from $\pi$VAE defined for all $s$. Varying $\mathcal{Z}$ produces different sample paths. Computationally, $f$ can be efficiently evaluated at any arbitrary location $s$ using matrix algebra: $f(s) = d(\mathcal{Z})^\top \Phi(s)$. We remark that the stochastic process perspective is readily apparent: for a random variable $\mathcal{Z}$, $d(\mathcal{Z})^\top \Phi(s)$ is a random variable defined on the same probability space for all $s$.

Algorithm 3 in the Appendix presents the step-by-step process for simulation with $\pi$VAE.

$\pi$VAE can be used for inference on new data pairs $(s_j, y_j)$, where the unnormalised posterior distribution is

$$p(\mathcal{Z} \mid d, y_j, s_j, \Phi) \propto p(y_j \mid d, s_j, \mathcal{Z}, \Phi) p(\mathcal{Z}) \tag{12}$$

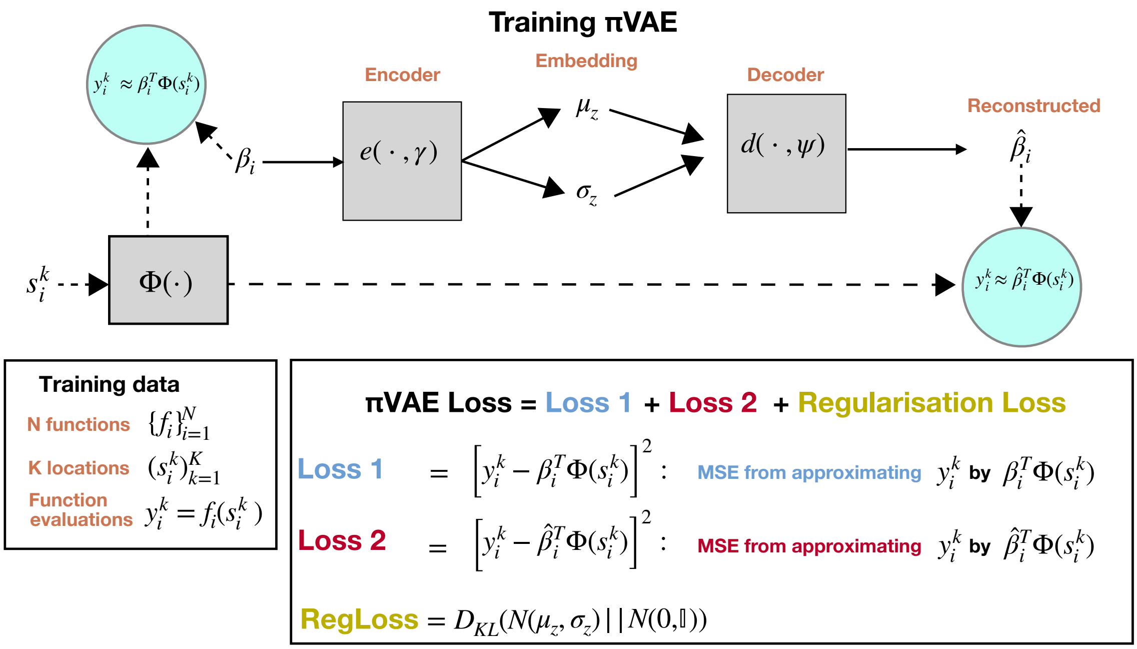

Training πVAE
$y_i^k \approx \beta_i^T\Phi(s_i^k)$
Encoder
Embedding
Decoder
Reconstructed
$\beta_i$
$e(\cdot,\gamma)$
$\mu_z$
$\sigma_z$
$d(\cdot,\psi)$
$\hat{\beta}_i$
$s_i^k$
$\Phi(\cdot)$
$y_i^k \approx \hat{\beta}_i^T\Phi(s_i^k)$
Training data
N functions $\{f_i\}_{i=1}^N$
K locations $(s_i^k)_{k=1}^K$
Function evaluations $y_i^k = f_i(s_i^k)$
πVAE Loss = Loss 1 + Loss 2 + Regularisation Loss
Loss 1 $= \left[y_i^k - \beta_i^T\Phi(s_i^k)\right]^2$ : MSE from approximating $y_i^k$ by $\beta_i^T\Phi(s_i^k)$
Loss 2 $= \left[y_i^k - \hat{\beta}_i^T\Phi(s_i^k)\right]^2$ : MSE from approximating $y_i^k$ by $\hat{\beta}_i^T\Phi(s_i^k)$
RegLoss $= D_{KL}(N(\mu_z,\sigma_z)\,||\,N(0,\mathbb{I}))$

**Fig. 2**: Schematic description of end-to-end trainig procedure for $\pi$VAE including the reconstruction loss. Dashed arrows contribute to the loss, blue circles are reconstructions, and grey boxes are functions.

with likelihood $p(y_j \mid d, s_j, \mathcal{Z}, \Phi)$ and prior $p(\mathcal{Z})$. MCMC can be used to efficiently obtain samples from the posterior distribution over $\mathcal{Z}$ using Equation (12). An implementation in probabilistic programming languages such as Stan (Carpenter et al., 2017) is very straightforward.

The posterior predictive distribution of $y_j$ at a location $s_j$ is given by:

$$p(y_j \mid d, s_j, \Phi) = \int p(y_j \mid d, s_j, \Phi, \mathcal{Z})p(\mathcal{Z} \mid d, y_j, s_j, \Phi)d\mathcal{Z} \tag{13}$$

While equations Eqs. (12)-(13) are written for a single location $s_j$, we can extend them to any arbitrary collection of locations without loss of generality, a necessary condition for $\pi$VAE to be a valid stochastic process. Further, the distinguishing difference between Eq. (6) and Eqs. (12)-(13) is conditioning on input locations and $\Phi$. It is $\Phi$ that ensures $\pi$VAE is a valid stochastic process. We formally prove this below.

Algorithm 2 in the Appendix presents the step-by-step process for inference with $\pi$VAE.

### 2.3.2 $\pi$VAE is a stochastic process

**Claim.** $\pi$VAE is a stochastic process.

Recall that, mathematically, a stochastic process is defined as a collection $\{f(s) : s \in S\}$, where $f(s)$ for each location $s \in S$ is a random variable on a common probability space $(\Omega, \mathcal{F}, P)$, see, e.g., Pavliotis (2014, Definition 1.1). This technical requirement is necessary to ensure that for any locations $s_1, \ldots, s_n \in S$, the random variables $f(s_1), \ldots, f(s_n)$ have a well-defined joint distribution. Subsequently, it also ensures consistency. Namely, writing $f_i := f(s_i)$ and integrating $f_n$ out, we get

$$p(f_1, \ldots, f_{n-1}) = \int_{f_n} p(f_1, \ldots, f_n)df_n.$$

**Proof.** For $\pi$VAE, we have $f(\cdot) := d(\mathcal{Z})\Phi(\cdot)$, where $\mathcal{Z}$ is a multivariate Gaussian random variable, hence defined on some probability space $(\Omega, \mathcal{F}, P)$. Since $d$ and $\Phi$ are deterministic (measurable) functions, it follows that $f(s_i) := d(\mathcal{Z})\Phi(s_i)$ for any $i = 1, \ldots, n$, is a random variable on $(\Omega, \mathcal{F}, P)$, whereby $\{f(s) : s \in S\}$ is a stochastic process. ■

We remark here that $\pi$VAE is a new stochastic process. If $\pi$VAE is trained on samples from a zero mean Gaussian process with a squared exponential covariance function, and similarly choose $\Phi$ to have the same covariance function, *and* $d$ is linear, then $\pi$VAE will be a Gaussian process. But for a non-positive definite $\Phi$ and / or non-linear $d$, even if $\pi$VAE is trained on samples from a Gaussian process, it will not truly be a Gaussian process, but some other stochastic process which approximates a Gaussian process. We do not know the theoretical conditions under which $\pi$VAE will perform better or worse than existing classes of stochastic processes; its general construction means that theoretical results will be challenging to prove in full generality. We demonstrate below that in practice, $\pi$VAE performs very well.

### 2.4 Examples

We first demonstrate the utility of our proposed $\pi$VAE model by fitting the simulated 1-D regression problem introduced in (Hernández-Lobato & Adams, 2015). The training points for the dataset are created by uniform sampling of 20 inputs, $x$, between $(-4, 4)$. The corresponding output is set as $y \sim \mathcal{N}(x^3, 9)$. We fit two different variants of $\pi$VAE, representing two different prior classes of functions. The first prior produces cubic monotonic functions and the second prior is a GP with an RBF kernel and a two layer neural network. We generated $10^4$ different function draws from both priors to train the respective $\pi$VAE. One important consideration in $\pi$VAE is to chose a sufficiently expressive $\Phi$, we used a RBF layer (see Appendix D) with trainable centres coupled with two layer neural network with 20 hidden units each. We compare our results against 20,000 Hamiltonian Monte Carlo (HMC) samples (R.M. Neal, 1993) implemented using Stan (Carpenter et al., 2017). Details of the implementation for all the models can be found in the Appendix.

Figure 3(a) presents results for $\pi$VAE with a cubic prior, Figure 3(b) with an RBF prior, and Figure 3(c) for standard Gaussian processes fitting using an RBF kernel. The mean absolute error (MAE) for all three methods are presented in Table 1. Both, the mean estimates and the uncertainty from $\pi$VAE variants, are closer, and more constrained than the ones using Gaussian processes with HMC. Importantly, $\pi$VAE with cubic prior not only produces better point estimates but is able to capture better uncertainty bounds.

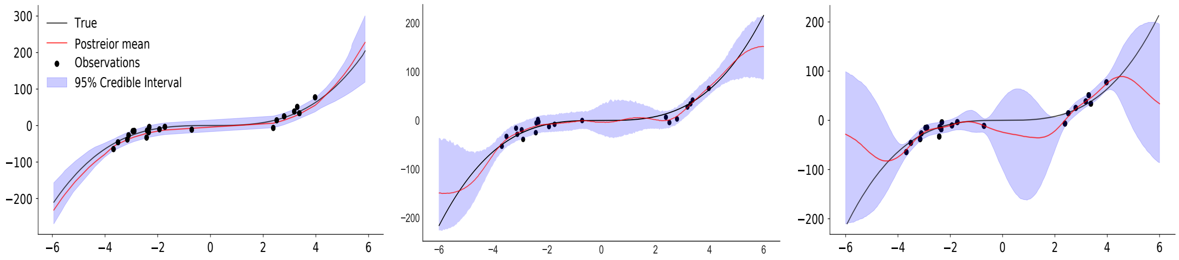
| Method | Test MAE |
|---|---|
| $\pi$VAE (cubic functions) | 10.47 |
| $\pi$VAE (Gaussian process with RBF kernel) | 33.15 |
| Gaussian process with RBF kernel | 67.37 |

**Table 1**: Test results of fitting to a cubic function with noise $y \sim \mathcal{N}(x^3, 9)$.

We note that $\pi$VAE does not exactly replicate an RBF Gaussian process, but does retain the main qualitative features inherent to GPs - such as the concentration of the posterior where there is data. Despite $\pi$VAE ostensibly learning an RBF function class, differences are to be expected from the VAE low dimensional embedding. This simple example demonstrates that $\pi$VAE can be used to incorporate domain knowledge about the functions being modelled.

In many scenarios, learning just the mapping of inputs to outputs is not sufficient as other functional properties are required to perform useful (interesting) analysis. For example, using point processes requires knowing the underlying intensity function, however, to perform inference we need to calculate the integral of that intensity function too. Calculating this integral, even in known analytical form, is very expensive. Hence, in order to circumvent the issue, we use $\pi$VAE to learn both function values and its integral for the observed events. Figure 4 shows $\pi$VAE prediction for both the intensity and integral of a simulated 1-D log-Gaussian Cox Process (LGCP).

In order to train $\pi$VAE to learn from the function space of 1-D LGCP functions, we first create a training set by drawing 10,000 different samples of the intensity function using an RBF kernel for 1-D LGCP. For each of the drawn intensity function, we choose an appropriate time horizon to sample 80 observed events (locations) from the intensity function. $\pi$VAE is trained on the sampled 80 locations with their corresponding intensity and the integral. $\pi$VAE therefore outputs both the instantaneous intensity and the integral of the intensity. The implementation details can be seen in the Appendix. For testing, we first draw a new intensity function (1-D LGCP) using the same mechanism used in training and sample 100 events (locations). As seen in Figure 4 our estimated intensity is very close to true intensity and even the estimated integral is close to the true integral. This example

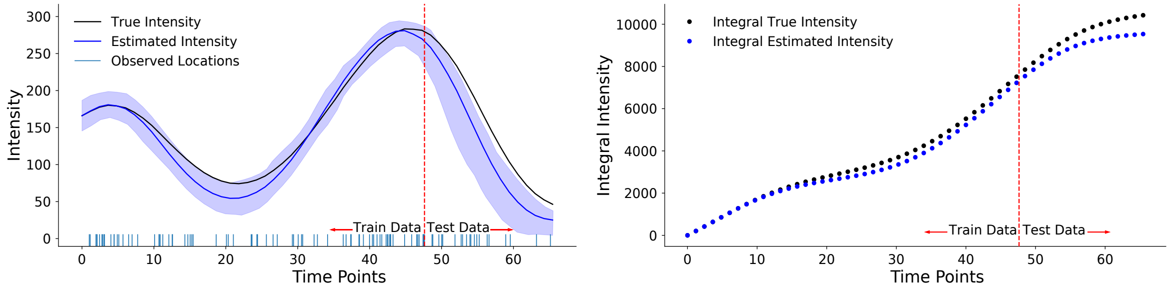


(a)

(b)

(c)

**Fig. 3**: Fitting to a cubic function with noise $y \sim \mathcal{N}(x^3, 9)$. (a) $\pi$VAE trained on a class of cubic functions, (b) $\pi$VAE trained on samples from a Gaussian process with RBF kernel and (c) is a Gaussian process with RBF kernel. All methods use Hamiltonian Markov Chain Monte Carlo for posterior inference.



(a)

(b)

**Fig. 4**: Inferring the intensity of a log-Gaussian Cox Process. (a) compares the posterior distribution of the intensity estimated by $\pi$VAE to the true intensity function on train and test data. (b) compares the posterior mean of the cumulative integral over time estimated by $\pi$VAE to the true cumulative integral on train and test data.

shows that the $\pi$VAE approach can be used to learn not only function evaluations but properties of functions.

# 3 Results

Here we show applications of $\pi$VAE on three real world datasets. In our first example we use $\pi$VAE to predict the deviation in land surface temperature in East Africa (Ton et al., 2018). We have the deviation in land surface temperatures for ~89,000 locations across East Africa. Our training data consisted of 6,000 uniformly sampled locations. Temperature was predicted using only the spatial locations as inputs. Figure 5 and Table 2 shows the results of the ground truth (a), our $\pi$VAE (b), a full rank Gaussian process with Matérn kernel (Gardner, Pleiss, Bindel, Weinberger, & Wilson, 2018) (c), and low rank Gauss Markov random field (GMRF) (a widely used approach in the field of geostatistics) with $1,046$ ($\frac{1}{6}$th of the training size) basis functions (Lindgren et al., 2011; Rue, Martino, & Chopin, 2009) (d). We train our $\pi$VAE model on $10^7$ functions draws from 2-D GP with small lengthscales between $10^{-5}$ to 2. $\Phi$ was set to be a Matérn layer ( see Appendix D) with 1,000 centres followed by a two layer neural network of 100 hidden units in each layer. The latent dimension of $\pi$VAE was set to 20. As seen in Figure 5, $\pi$VAE is able to capture small scale features and produces a far better reconstruction than the both full and low rank GP and despite having a much smaller latent dimension of 20 vs 6,000 (full) vs 1,046 (low). The testing error for $\pi$VAE is substantially better than the full rank GP which leads to the question, why does $\pi$VAE perform so much

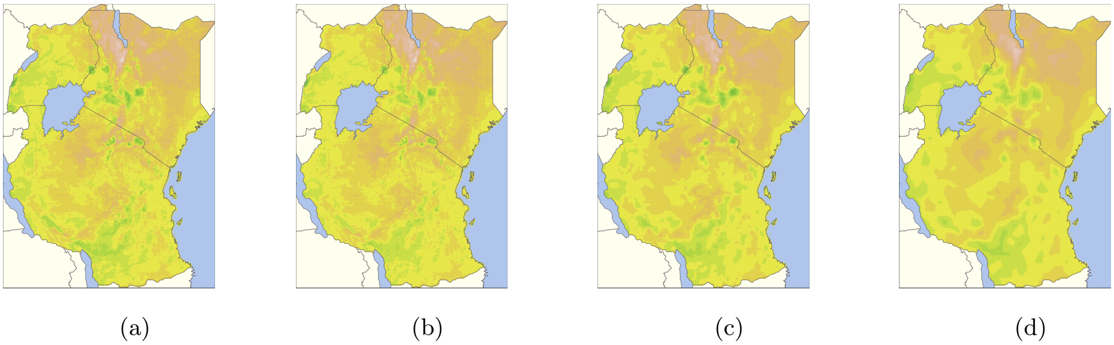


**Fig. 5**: Deviation in land surface temperature for East Africa trained on 6000 random uniformly chosen locations (Ton et al., 2018). Plots: (a) the data, (b) our $\pi$VAE approach (testing MSE: 0.38), (c) a full rank GP with Matérn $\frac{3}{2}$ kernel (testing MSE: 2.47), and (d) a low rank SPDE approximation with 1046 basis functions (Lindgren et al., 2011) and a Matérn $\frac{3}{2}$ kernel (testing MSE: 4.36). $\pi$VAE not only has substantially lower test error, it captures fine scale features much better than Gaussian processes or neural processes.

| Method | Test MSE |
|---|---|
| Full rank GP | 2.47 |
| $\pi$VAE | 0.38 |
| low rank GMRF ($basis = 1046$) | 4.36 |

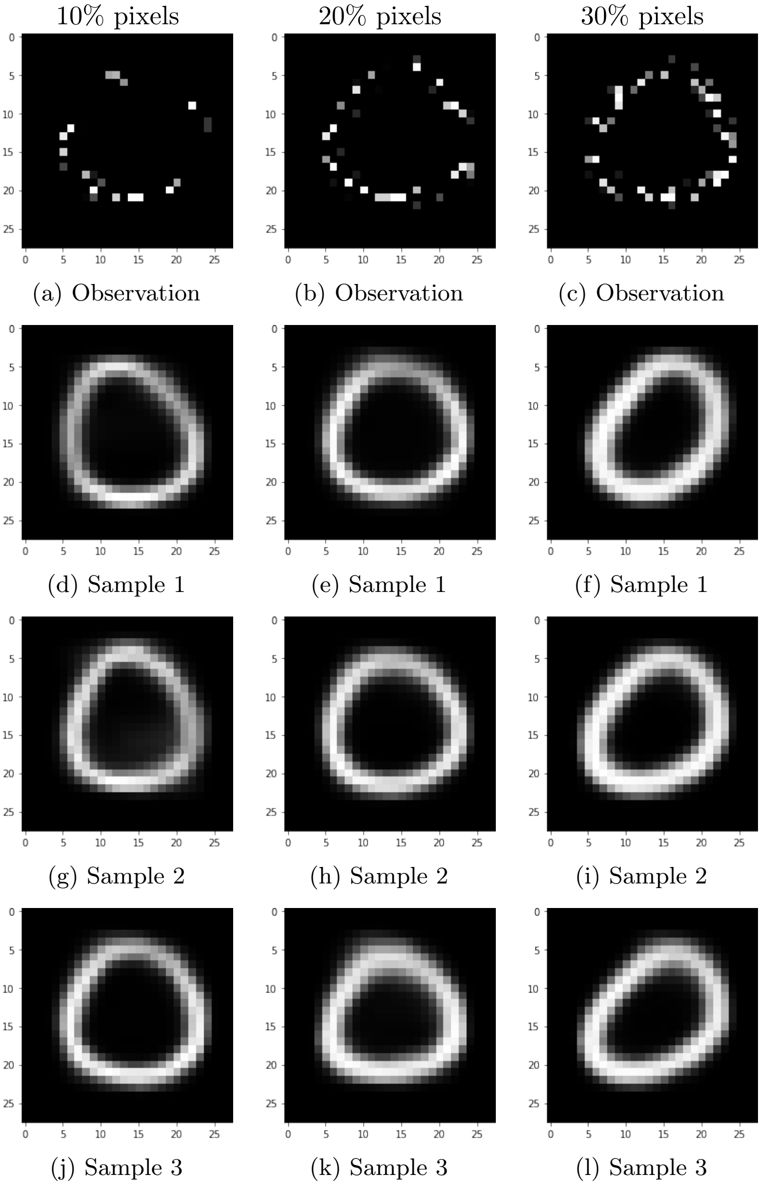
**Table 2**: Test results for $\pi$VAE, a full rank GP, and low rank GMRF on *land surface temperature for East Africa trained on 6000 random uniformly chosen locations (Ton et al., 2018).*

| Method | RMSE | NLL |
|---|---|---|
| Full rank GP | 0.099 | -0.258 |
| $\pi$VAE | 0.112 | 0.006 |
| SGPR ($m = 512$) | 0.273 | 0.087 |
| SVGP ($m = 1024$) | 0.268 | 0.236 |

**Table 3**: Test results for $\pi$VAE, a full rank GP and approximate algorithms SGPR and SVGP on *Kin40K*.

better than a GP, despite being trained on samples from a GP? One possible reason is that the extra hidden layers in $\Phi$ create a much richer structure that could capture elements of non-stationarity (Ton et al., 2018). Alternatively, the ability to use state-of-the-art MCMC and estimate a reliable posterior expectation might create resilience to overfitting. The training/testing error for $\pi$VAE is 0.07/0.38, while the full rank GP is 0.002/2.47. Therefore the training error is 37 times smaller in the GP, but the testing error is only 6 times smaller in $\pi$VAE suggesting that, despite marginalisation, the GP is still overfitting.

Table 3 compares $\pi$VAE on the *Kin40K* (Schwaighofer & Tresp, 2003) dataset to state-of-the-art full and approximate GPs, with results taken from (Wang et al., 2019). The objective was to predict the distance of a robotic arm from the target given the position of all 8 links present on the robotic arm. In total we have 40,000 samples which are divided randomly into $\frac{2}{3}$ training samples and $\frac{1}{3}$ test samples. We train $\pi$VAE on $N = 10^7$ functions drawn from an 8-D GP, observed at $K = 200$ locations, where each of the 8 dimensions had values drawn uniformly from the range $(-2, 2)$ and lengthscale varied between $10^{-3}$ and 10. Once $\pi$VAE was trained on the prior function we use it to infer the posterior distribution for the training examples in *Kin40K*. Table 3 shows results for RMSE and negative log-likelihood (NLL) of $\pi$VAE against various GP methods on test samples. The full rank GP results reported in (Wang et al., 2019) are better than those from $\pi$VAE, but we are competitive, and far better than the approximate GP methods. We also note that the exact GP is estimated via maximising the log marginal likelihood in closed form, while $\pi$VAE performs full Bayesian inference; all posterior checks yielded excellent convergence measured via $\hat{R}$ and effective samples sizes. Calibration was checked using posterior predictive intervals. For visual diagnostics see the Appendix.



(a) Observation (b) Observation (c) Observation

(d) Sample 1 (e) Sample 1 (f) Sample 1

(g) Sample 2 (h) Sample 2 (i) Sample 2

(j) Sample 3 (k) Sample 3 (l) Sample 3

**Fig. 6**: MNIST reconstruction after observing only 10, 20 or 30% of pixels from original data.

Finally, we apply $\pi$VAE to the task of reconstructing MNIST digits using a subset of pixels from each image. Similar to the earlier temperature prediction task, image completion can also be seen as a regression task in 2-D. The regression task is to predict the intensity of pixels given the pixel locations. We first train neural processes on full MNIST digits from the training split of the dataset, whereas $\pi$VAE is trained on $N = 10^6$ functions drawn from a 2-D GP. The latent dimension of $\pi$VAE is set to be 40. As with previous examples, the decoder and encoder networks are made up of two layer neural networks. The hidden units for the encoder are 256 and 128 for the first and second layer respectively, and the reverse for decoder.

Once we have trained $\pi$VAE we now use images from the test set for prediction. Images in the testing set are sampled in such a way that only 10, 20 or 30% of pixel values are observed. Inference is performed with $\pi$VAE to predict the intensity at all other pixel locations using Eq. (13). As seen from Figure 6, the performance of $\pi$VAE increases with increase in pixel locations available during prediction but still even with 10% pixels our model is able to learn a decent approximation of the image. The uncertainty in prediction can be seen from the different samples produced by the model for the same data. As the number of given locations increases, the variance between samples decreases with quality of the image also increasing. Note that results from neural processes, as seen in Figure C4, look better than from $\pi$VAE. Neural processes performed better in the MNIST case because they were specifically trained on full MNIST digits from the training dataset, whereas piVAE was trained on the more general prior class of 2D GPs.

# 4 Discussion and Conclusion

In this paper we have proposed a novel VAE formulation of a stochastic process, with the ability to learn function classes and properties of functions. Our $\pi$VAEs typically have a small (5-50) , uncorrelated latent dimension of parameters, so Bayesian inference with MCMC is straightforward and highly effective at successfully exploring the posterior distribution. This accurate estimation of uncertainty is essential in many areas such as medical decision-making.

$\pi$VAE combines the power of deep learning to create high capacity function classes, while ensuring tractable inference using fully Bayesian MCMC approaches. Our 1-D example in Figure 3 demonstrates that an exciting use of $\pi$VAE is to incorporate domain knowledge about the problem. Monotonicity or complicated dynamics can be encoded directly into the prior (Caterini, Doucet, & Sejdinovic, 2018) on which $\pi$VAE is trained. Our log-Gaussian Cox Process example shows that not only functions can be modelled, but also properties of functions such as integrals. Perhaps the most surprising result is the performance of $\pi$VAE on spatial interpolation. Despite being trained on samples from a Gaussian process, $\pi$VAE substantially outperforms a full rank GP. We conjecture this is due to the more complex structure of the feature representation $\Phi$ and due to a resilience to overfitting.

There are costs to using $\pi$VAE, especially the large upfront cost in training. For complex priors, training could take days or weeks and will invariably require the heuristics and parameter searches inherent in applied deep learning to achieve a good performance. However, once trained, a $\pi$VAE network is applicable on a wide range of problems, with the Bayesian inference MCMC step taking seconds or minutes.

Future work should investigate the performance of $\pi$VAE on higher dimensional settings (input spaces $> 10$). Other stochastic processes, such as Dirichlet processes, should also be considered.

# Declarations

## Funding

SB acknowledge the The Novo Nordisk Young Investigator Award (NNF20OC0059309) which also supports SM. SB also acknowledges the Danish National Research Foundation Chair grant, The Schmidt Polymath Award and The NIHR Health Protection Research Unit (HPRU) in Modelling and Health Economics. SM and SB acknowledge funding from the MRC Centre for Global Infectious Disease Analysis (reference MR/R015600/1) and Community Jameel. SF acknowledges the EPSRC (EP/V002910/2) and the Imperial College COVID-19 Research Fund.

## Conflict of interest/Competing interests

NA

## Ethics approval

NA

## Consent to participate

NA

## Consent for publication

NA

## Availability of data and materials

All data used in the paper is available at https://github.com/MLGlobalHealth/pi-vae.

## Code availability

Code is available at https://github.com/MLGlobalHealth/pi-vae.

## References

Antoniak, C.E. (1974). Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, 1152–1174.

Betancourt, M., Byrne, S., Livingstone, S., Girolami, M. (2017). The geometric foundations of Hamiltonian Monte Carlo. *Bernoulli*.

10.3150/16-BEJ810

Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D. (2015). Weight uncertainty in neural networks. *32nd international conference on machine learning, icml 2015.*

Broomhead, D.S., & Lowe, D. (1988, March). Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks. *DTIC*. Retrieved from https://apps.dtic.mil/sti/citations/ADA196234

Carpenter, B., Gelman, A., Hoffman, M.D., Lee, D., Goodrich, B., Betancourt, M., . . . Riddell, A. (2017). Stan : A Probabilistic Programming Language. *Journal of Statistical Software*, *76*(1), 1–32.

10.18637/jss.v076.i01

Caterini, A.L., Doucet, A., Sejdinovic, D. (2018). Hamiltonian variational auto-encoder. *Advances in neural information processing systems.*

Gardner, J.R., Pleiss, G., Bindel, D., Weinberger, K.Q., Wilson, A.G. (2018). Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. *Advances in neural information processing systems.*

Garnelo, M., Rosenbaum, D., Maddison, C.J., Ramalho, T., Saxton, D., Shanahan, M., . . . Eslami, S.M. (2018). Conditional neural processes. *35th international conference on machine learning, icml 2018.*

Hawkes, A.G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, *58*(1), 83–90.

10.1093/biomet/58.1.83

Hernández-Lobato, J.M., & Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. *International conference on machine learning* (pp. 1861–1869).

Hinton, G.E., & Salakhutdinov, R.R. (2006). Reducing the dimensionality of data with neural networks. *science*, *313*(5786), 504–507.

Hoffman, M.D., Blei, D.M., Wang, C., Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, *14*(1), 1303–1347.

Huggins, J.H., Kasprzak, M., Campbell, T., Broderick, T. (2019). Practical posterior error bounds from variational objectives. *arXiv preprint arXiv:1910.04102*.

Jacot, A., Gabriel, F., Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems.*

Karhunen, K. (1947). On linear methods in probability theory. *Annales academiae scientiarum fennicae, ser. al* (Vol. 37, pp. 3–79).

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, S.M.A., Rosenbaum, D., . . . Teh, Y.W. (2019). Attentive Neural Processes. *CoRR*, *abs/1901.0*.

Kingma, D.P., & Ba, J. (2014, 12). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D.P., & Welling, M. (2014). Auto-Encoding Variational Bayes (VAE, reparameterization trick). *ICLR 2014*.

Kingma, D.P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307–392.

Lakshminarayanan, B., Pritzel, A., Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems.*

Lindgren, F., Rue, H., Lindström, J. (2011). An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *73*(4), 423–498.

10.1111/j.1467-9868.2011.00777.x

Loeve, M. (1948). Functions aleatoires du second ordre. *Processus stochastique et mouvement Brownien*, 366–420.

Minka, T.P. (2001). Expectation propagation for approximate bayesian inference. *Proceedings of the seventeenth conference on uncertainty in artificial intelligence* (p. 362–369). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Mishra, S., Rizoiu, M.-A., Xie, L. (2016). Feature driven and point process approaches for popularity prediction. *Proceedings of the 25th acm international on conference on information and knowledge management* (p. 1069–1078). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2983323.2983812 10.1145/2983323.2983812

Møller, J., Syversveen, A.R., Waagepetersen, R.P. (1998). Log gaussian cox processes. *Scandinavian Journal of Statistics*, *25*(3), 451-482. https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9469.00115

10.1111/1467-9469.00115

Neal, R. (1996). Bayesian Learning for Neural Networks. *LECTURE NOTES IN STATISTICS -NEW YORK- SPRINGER VERLAG-*.

Neal, R.M. (1993). *Probabilistic inference using markov chain monte carlo methods.* Department of Computer Science, University of Toronto Toronto, Ontario, Canada.

Park, J., & Sandberg, I.W. (1991, June). Universal Approximation Using Radial-Basis-Function Networks. *Neural Comput.*, *3*(2), 246–257.

10.1162/neco.1991.3.2.246

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … others (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* (pp. 8024–8035).

Pavliotis, G.A. (2014). *Stochastic processes and applications: diffusion processes, the fokker-planck and langevin equations* (Vol. 60). Springer.

Rahimi, A., & Recht, B. (2008). Random features for large-scale kernel machines. *Advances in neural information processing systems* (pp. 1177–1184).

Rasmussen, C.E., & Williams, C.K.I. (2006). *Gaussian processes for machine learning.* MIT Press. Retrieved from http://www.worldcat.org/oclc/61285753

Rezende, D.J., Mohamed, S., Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *International conference on machine learning* (pp. 1278–1286).

Ritter, H., Botev, A., Barber, D. (2018). A scalable laplace approximation for neural networks. *6th international conference on learning representations, iclr 2018 - conference track proceedings.*

Ross, S.M. (1996). *Stochastic processes* (Vol. 2). John Wiley & Sons.

Roy, D.M., & Teh, Y.W. (2009). The Mondrian process. *Advances in neural information processing systems 21 - proceedings of the 2008 conference.*

Rue, H., Martino, S., Chopin, N. (2009, 4). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, *71*(2), 319–392.

10.1111/j.1467-9868.2008.00700.x

Schwaighofer, A., & Tresp, V. (2003). Transductive and inductive methods for approximate gaussian process regression. *Advances in neural information processing systems* (pp. 977–984).

Semenova, E., Xu, Y., Howes, A., Rashid, T., Bhatt, S., Mishra, S., Flaxman, S. (2022). Priorvae: encoding spatial priors with variational autoencoders for small-area estimation. *Journal of the Royal Society Interface*, *19*(191), 20220094.

Ton, J.-F., Flaxman, S., Sejdinovic, D., Bhatt, S. (2018). Spatial mapping with Gaussian processes and nonstationary Fourier features. *Spatial Statistics*.

10.1016/j.spasta.2018.02.002

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K.Q., Wilson, A.G. (2019). Exact gaussian processes on a million data points. *Advances in neural information processing systems* (pp. 14622–14632).

Welling, M., & Teh, Y.W. (2011). Bayesian learning via stochastic gradient langevin dynamics. *Proceedings of the 28th international conference on machine learning, icml 2011.*

Yao, J., Pan, W., Ghosh, S., Doshi-Velez, F. (2019). Quality of uncertainty quantification for bayesian neural network inference. *arXiv preprint arXiv:1906.09686*.

Yao, Y., Vehtari, A., Simpson, D., Gelman, A. (2018). Yes, but did it work?: Evaluating variational inference. *35th international conference on machine learning, icml 2018.*

# Appendix A MCMC diagnostics

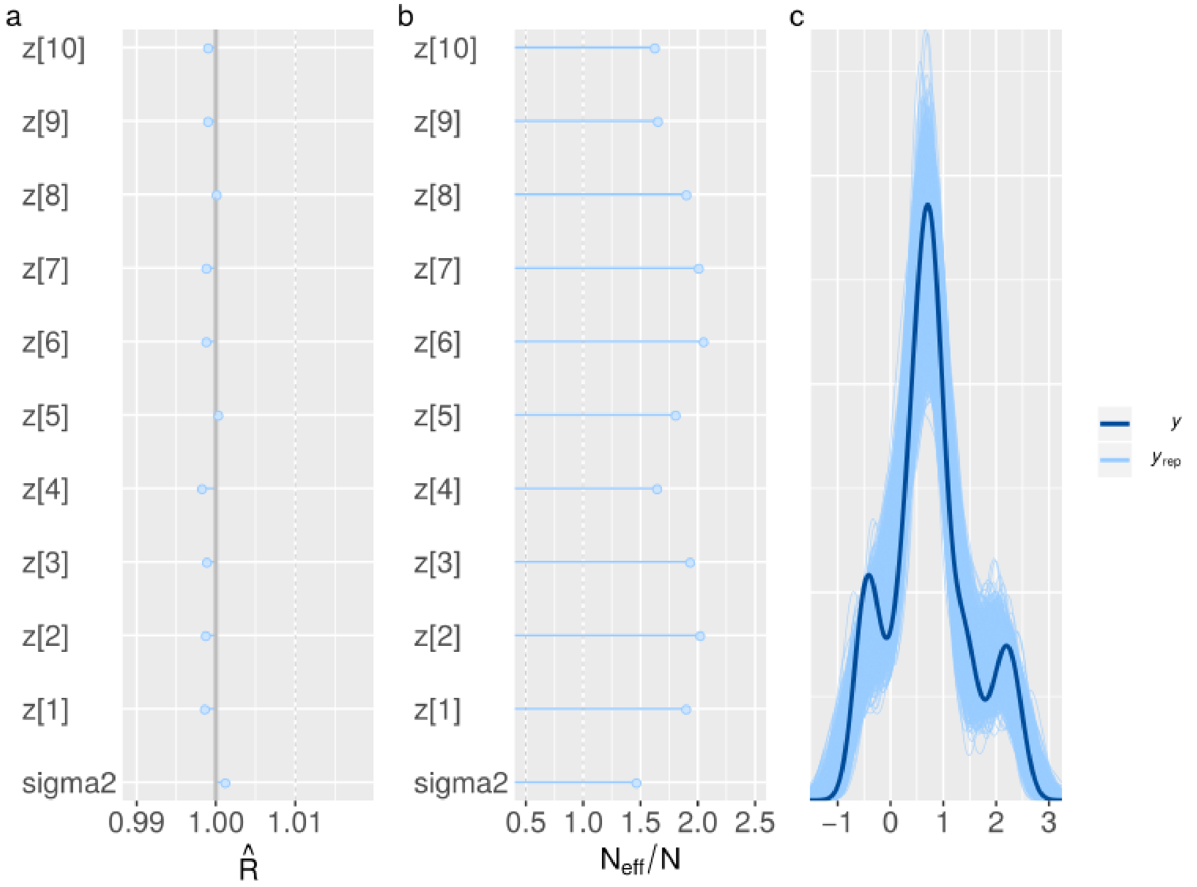


**Fig. A1**: MCMC diagnostics for VAE inference presented in Figure 1: (a) and (b) shows the values for $\hat{R}$ and $\frac{N_{eff}}{N}$ for all parameters inferred with Stan. (c) shows the true distribution of observations along with the draws from the posterior predictive distribution.

Figure A1 presents the MCMC diagnostics for the 1-D GP function learning example shown in Figure 1. Both $\hat{R}$ and effective sample size for all the inferred parameters (latent dimension of the VAE and noise in the observation) are well behaved with $\hat{R} \leq 1.01$ (Figure A1(a)) and effective sample size greater than 1 (Figure A1(b)). Furthermore, even the draws from the posterior predictive distribution very well capture the true distribution in observations as shown in Figure A1(c).

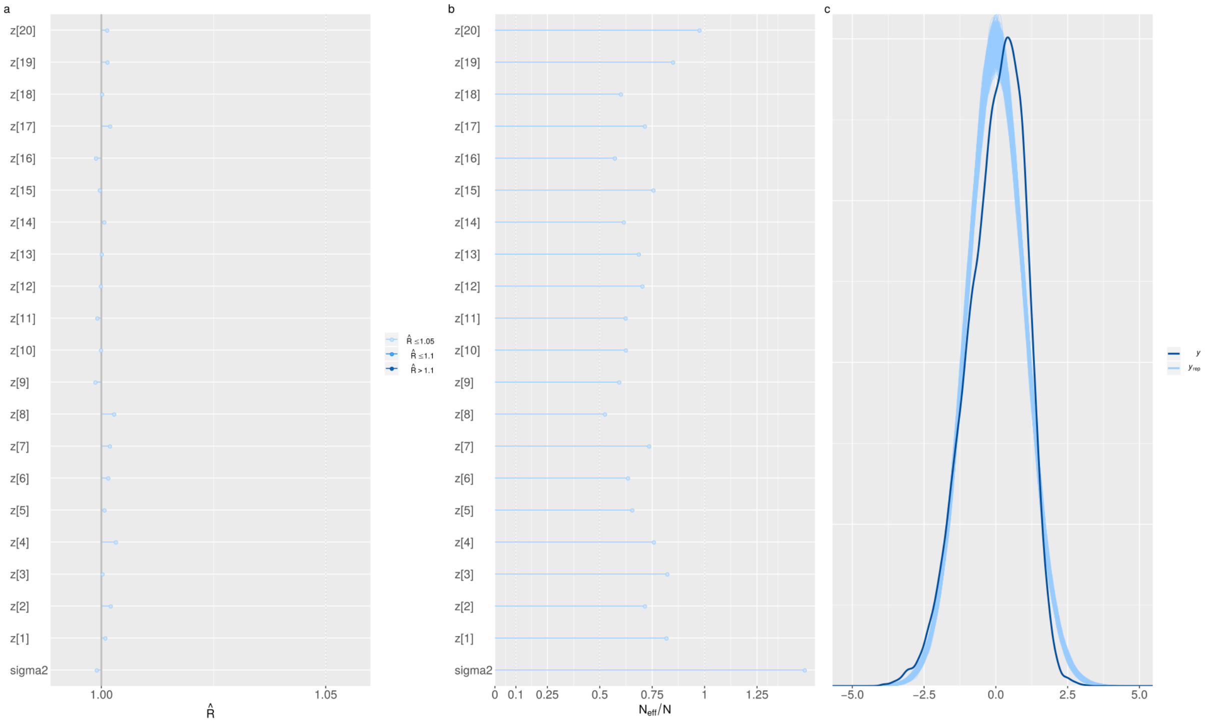


**Fig. A2**: MCMC diagnostics for $\pi$VAE inference presented in Table 3: (a) and (b) shows the values for $\hat{R}$ and $\dfrac{N_{eff}}{N}$ for all parameters inferred with Stan. (c) shows the true distribution of observations along with the draws from the posterior predictive distribution.

Figure A2 presents the MCMC diagnostics for the kin40K dataset with $\pi$VAE as shown in Table 3. Both $\hat{R}$ and effective sample size for all the inferred parameters (latent dimension of the VAE and noise in the observation) are well behaved with $\hat{R} \leq 1.01$ (Figure A2(a)) and effective sample size greater than 0.5 (Figure A2(b)). Furthermore, the draws from the posterior predictive distribution are shown against the true distribution in observations as shown in Figure A2(c).

Figure A3 presents the MCMC calibration plots for the posterior. Both the marginal predictive check and leave one out predictive intervals plots demonstrate that our posterior is well calibrated.

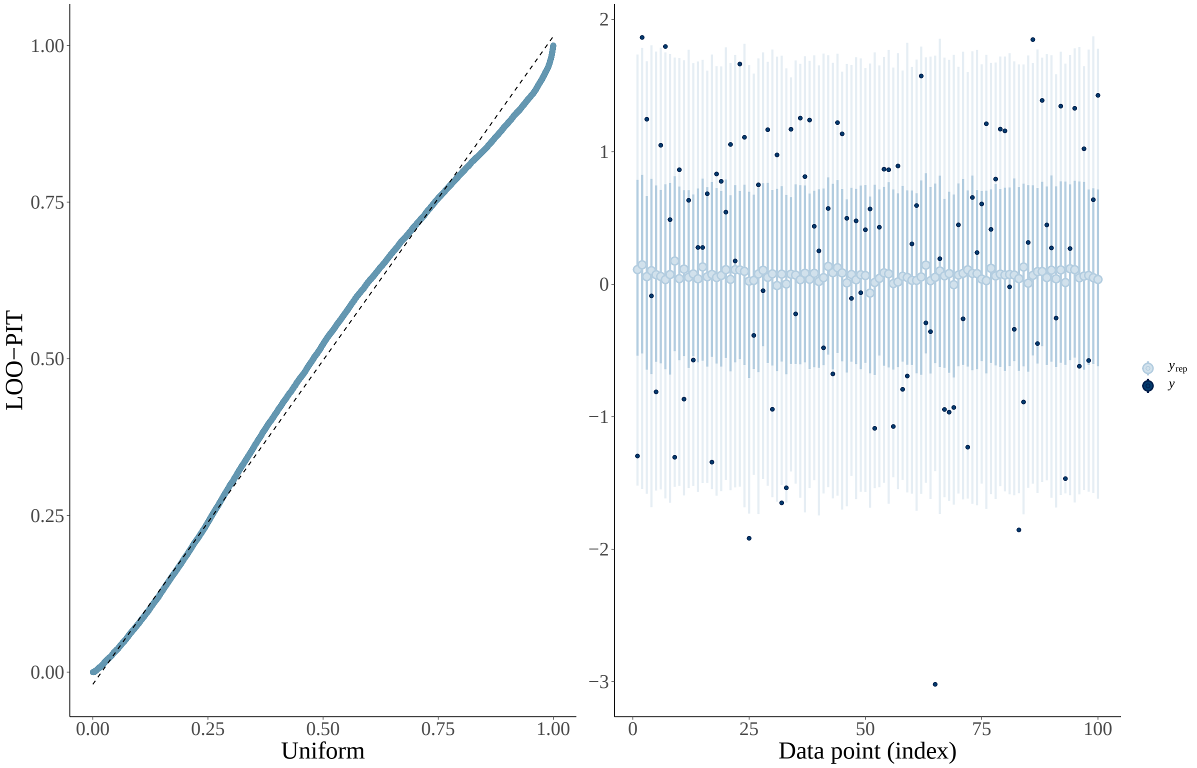


**Fig. A3**: MCMC calibration for πVAE inference presented in Table 3: (a) shows the marginal predictive check using a leave one out probability integral transform and (b) shows the leave one out predictive intervals compared to observations.

# Appendix B Algorithm

$\pi$VAE proceeds in two stages. In the first stage (Algorithm 1) we train $\pi$VAE, using a very large set of draws from a pre-specified prior class. In the second stage (Algorithm 2) we use the trained $\pi$VAE from Algorithm 1 as a prior, combine this with data using a likelihood, and perform inference. MCMC for Bayesian inference or optimization with a loss function are alternative approaches to learn the best $\mathcal{Z}$ to explain the data. While these two algorithms are all that is needed to apply $\pi$VAE, for completeness Algorithm 3 shows how one can use a trained $\pi$VAE, which encodes a stochastic process, to sample realisations from this stochastic process.

**Algorithm 1** Prior Training for $\pi$VAE (stage 1)

```
Simulate draws from N functions evaluated at K points to create input data consisting of location,
function value pairs: $\{(s_i^1, y_i^1), \dots, (s_i^K, y_i^K)\}_{i=1}^N$
repeat
    for each function i = 1, ..., N do
        for each location k = 1, ..., K do
            transform locations: $\Phi(s_i^k)$
            inner product with a linear basis:
                $\hat{y}_{i,1}^k \leftarrow \beta_i^T \Phi(s_i^k)$
        end for
        append $loss1$: $loss1 \leftarrow MSE(\hat{y}_{i,1}, y_i)$
        encode $\beta_i$ with VAE:
                $[z_\mu, z_{sd}]^\top = e(\beta_i, \gamma)$
        reparameterize for $\mathcal{Z}$: $\mathcal{Z} \sim \mathcal{N}(z_\mu, z_{sd}^2 \mathbb{I})$
        decode with VAE, $\hat{\beta}_i$ : $\hat{\beta}_i = d(\mathcal{Z}, \psi)$
        for each location k = 1, ..., K do
            transform locations: $\Phi(s_i^k)$
            inner product with decoded $\hat{\beta}_i$:
                $\hat{y}_{i,2}^k = \hat{\beta}_i^\top \Phi(s_i^k)$
        end for
        append $loss2$: $loss2 \leftarrow MSE(\hat{y}_{i,2}, y_i)$
        minimize $loss1 + loss2 + \mathrm{KL}\left(\mathcal{Z} \| \mathcal{N}(0, \mathbb{I})\right)$
                to get $\Phi, \beta_i, \gamma, \psi$
    end for
until termination criterion satisfied (epochs)
```

**Algorithm 2** Inference from $\pi$VAE (stage 2)

```
Require: Trained decoder $d$ ($\psi$ fixed) and $\Phi$ (learnt from Algorithm 1)
 1: Input: $J$ observations consist of location, function value pairs: $\{(s_j, y_j)\}_{j=1}^{J}$.
 2: Goal: infer latent function with parameters $\mathcal{Z}$.
 3: Sample $\mathcal{Z}$: $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I})$
 4: decode with VAE to get $\beta := d(\mathcal{Z}, \psi)$
 5: for each location j do
 6:     transform locations: $\Phi(s_j)$
 7:     inner product with decoder $\beta$: $\hat{y}_j \leftarrow \beta^T \Phi(s_j)$
 8: end for
 9: Perform Bayesian inference with MCMC for $\mathcal{Z}$ to obtain a set of draws from the posterior distribution:
        $p(\mathcal{Z} \mid d, y_1, s_1, \ldots, y_J, s_J, \Phi) \propto p(y_1, \ldots, y_J \mid d, s_1, \ldots, s_J, \mathcal{Z}, \Phi) p(\mathcal{Z})$
        $= p(y_1, \ldots, y_J \mid \hat{y}_1, \ldots, \hat{y}_J, \mathcal{Z}) p(\mathcal{Z})$
10: (alternative to step 9:) minimize an expected loss (e.g. gradient decent with mean squared error):
        $\arg\min_{\mathcal{Z}} \sum_{j=1}^{J} \|\hat{y}_j$ - $y_j\|^2$.
```

**Algorithm 3** Sampling from $\pi$VAE

```
Require: Trained decoder $d$ ($\psi$ fixed) and $\Phi$ (learnt from Algorithm 1)
1: Input: Locations $s_1, \ldots, s_J$ where we want to evaluate the sampled function.
2: Sample $\mathcal{Z}$: $\mathcal{Z} \sim \mathcal{N}(0, \mathbb{I})$
3: decode with VAE to get : $\beta := d(\mathcal{Z}, \psi)$
4: for each location j do
5:     transform locations: $\Phi(s_j)$
6:     inner product with $\beta$: $\hat{y}_j \leftarrow \beta^T \Phi(s_j)$
7: end for
```

# Appendix C MNIST Example

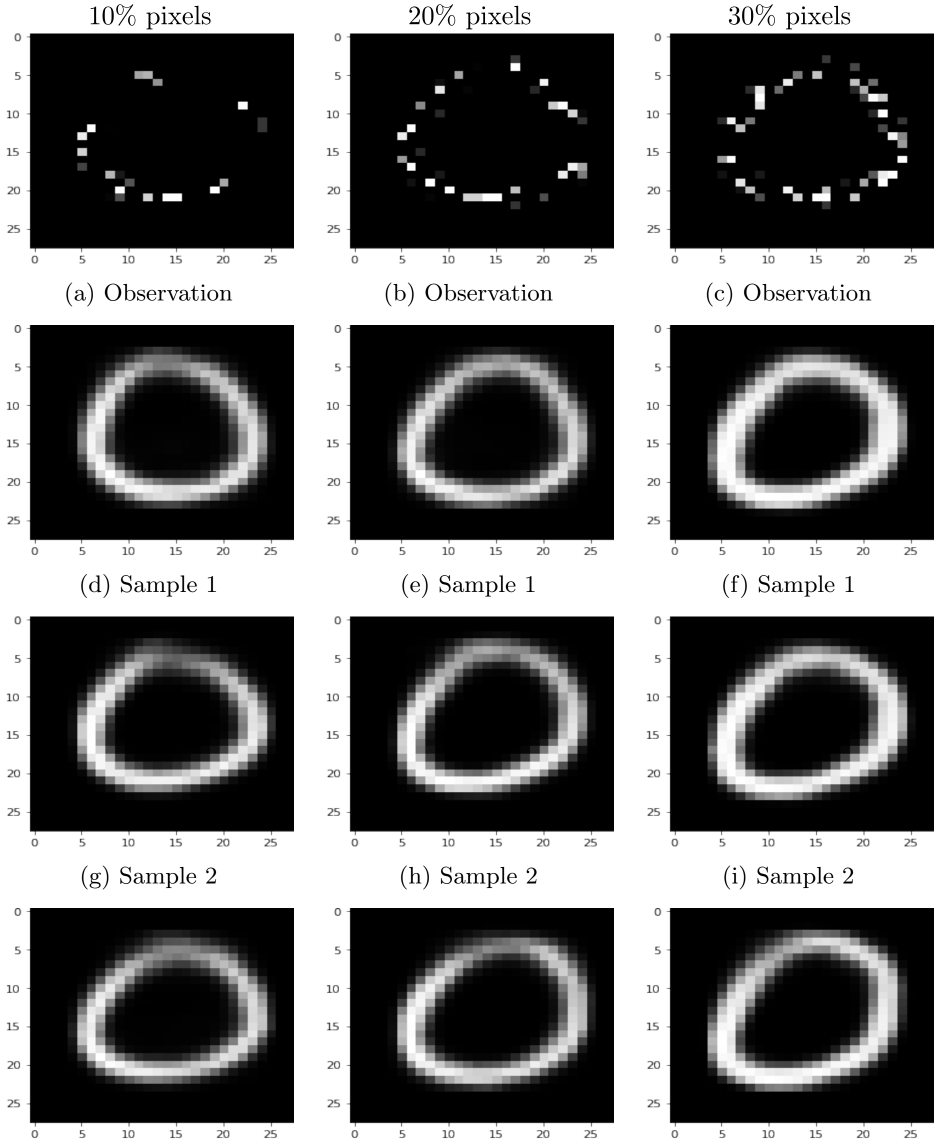
Figure C4 below is the MNIST example referenced in the main text for neural processes.

# Appendix D Implementation Details

All models were implemented with PyTorch (Paszke et al., 2019) in Python. For Bayesian inference Stan (Carpenter et al., 2017) was used. For training while using a fixed grid, when not mentioned in main text, in each dimension was on the range -1 to 1. Our experiments ran on a workstation with two NVIDIA GeForce RTX 2080 Ti cards.

**RBF and Matérn layers:** RBF and Matérn layers are implemented as a variant of the original RBF networks as described in (Broomhead & Lowe, 1988; Park & Sandberg, 1991). In our setting we define a set of $C$ trainable centers, which act as fixed points. Now for each input location we calculate a RBF or Matérn Kernel for all the fixed points. These calculated kernels are weighted for each fixed center and then summed over to create a scalar output for each location. We can describe the layer as follows:-

$$\Phi(s) = \sum_{i=1}^{c} \alpha_i \rho \left( \|s - c_i\| \right)$$



(a) Observation (b) Observation (c) Observation

(d) Sample 1 (e) Sample 1 (f) Sample 1

(g) Sample 2 (h) Sample 2 (i) Sample 2

(j) Sample 3 (k) Sample 3 (l) Sample 3

**Fig. C4**: MNIST reconstruction using neural processes after observing only 10, 20 or 30% of pixels from original data.

where $s$ is an input location (point), $\alpha_i$ is the weight for each center $c_i$, and $\rho\left(\|x - c_i\|\right)$ is RBF or Matérn kernel for RBF and Matérn layer respectively.

## LGCP simulation example

We study the function space of 1-D LGCP realisations. We define a nonnegative intensity function at any time $t$ as $\lambda(t) : T \to \mathcal{R}^+$. The number of events in an interval $[t_1, t_2]$ within some time period, $y_{[t_1,t_2]}$, is distributed Poisson via the following Bayesian hierarchical model:

$$Z(t) \sim \mathrm{GP}(0, k)$$

$$\begin{aligned} \lambda(t) &= \gamma \cdot \exp(Z(t)) \\ y_{[t_1,t_2]} &\sim \text{Poisson}\left(\int_{t_1}^{t_2} \lambda(t)dt\right) \end{aligned} \tag{D1}$$

where $\gamma$ is a constant event rate, set to 5 in our experiments.

To train $\pi$VAE on functions from an LGCP, we draw 10,000 samples from Eq (D1), assuming $k$ is an RBF kernel/covariance function with form $e^{-\sigma x^2}$, with inverse lengthscale $\sigma$ chosen randomly from the set $\{8, 16, 32, 64\}$. We then choose an observation window sufficiently large to ensure that 80 events are observed. This approach is meant to simulate the situation in which we observe a point process until a certain number of events have occurrence, at which point we conduct inference (Mishra, Rizoiu, & Xie, 2016).

Given the set of $80 \times 10,000$ events, we train $\pi$VAE with their corresponding intensity and integral of the intensity over the corresponding observation window. The integral is calculated numerically. We concatenate the integral of the intensity at the end with the intensity itself (value of the function evaluated the specific location). Note, in this setup we have $\beta$ setup as a $2-D$ vector, first value corresponding to the intensity and second to the integral of the intensity. The task for $\pi$VAE is to simultaneously learn both the instantaneous intensity and the integral of the intensity. At testing, we expand the number of events (and hence the time horizon) to 100, and compare the intensity and integral of $\pi$VAE compared to the true LGCP. As seen in Figure 4, in this extrapolation, our estimated intensity is very close to the true intensity and even the estimated integral is close to the true (numerically calculated) integral. This example shows that the $\pi$VAE approach can be used to learn not only function evaluations but properties of functions.