A Derivation of the Loss

In this section we derive the objectives given in Eqs. (4) and (9), following a path similar to that presented in Ho et al. [10]. Let $\boldsymbol{s}_t = (\boldsymbol{x}_t, \boldsymbol{y}_t)$ be the state that combines both the inputs and observations. We wish to find the set of parameters θ which maximise the likelihood given the initial state, $p_\theta(\boldsymbol{s}_0)$. While a direct evaluation of the likelihood appears intractable,

$$p_\theta(\boldsymbol{s}_0) = \int p_\theta(\boldsymbol{s}_{0\ldots T}) d\boldsymbol{s}_{1\ldots T}\,,$$

this may be recast in a form which allows for a comparison to be drawn between the forward and reverse trajectories [31]

$$p_\theta(\boldsymbol{s}_0) = \int p_\theta(\boldsymbol{s}_T) q(\boldsymbol{s}_{1\ldots T}|\boldsymbol{s}_0) \prod_{t=1}^{T} \frac{p_\theta(\boldsymbol{s}_{t-1}|t)}{q(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})} d\boldsymbol{s}_{1\ldots T}\,.$$

The appeal of introducing the reverse process is that it is tractable when conditioned on the first state $\boldsymbol{s}_0$, taking a Gaussian form

$$q(\boldsymbol{s}_{t-1} \mid \boldsymbol{s}_t, \boldsymbol{s}_0) = \mathcal{N}(\boldsymbol{s}_{t-1} \mid \tilde{\boldsymbol{\mu}}(\boldsymbol{s}_t, \boldsymbol{s}_0), \tilde{\beta}_t \mathbf{I}).$$

Our loss function reflects a lower bound on the negative log likelihood

$$\mathbb{E}[-\log p_\theta(\boldsymbol{s}_0)] \leq \mathbb{E}_{q(\boldsymbol{s}_{0:T})}\left[\log \frac{q(\boldsymbol{s}_{1:T} \mid \boldsymbol{s}_0)}{p_\theta(\boldsymbol{s}_{0:T})}\right] := \mathcal{L}_\theta\,,$$

which may be decomposed into two edge terms and a sum over the intermediate steps, as follows

$$\mathcal{L}_\theta = \mathbb{E}_q\left[L_T + \sum_{t=2}^{T} L_{t-1} + L_0\right]\,,$$

where

$$\begin{aligned} L_0 &= -\log p_\theta(\boldsymbol{s}_0 \mid \boldsymbol{s}_1)\,, \\ L_{t-1} &= \mathrm{KL}(q(\boldsymbol{s}_{t-1} \mid \boldsymbol{s}_t, \boldsymbol{s}_0) \| p_\theta(\boldsymbol{s}_{t-1} \mid \boldsymbol{s}_t))\,, \\ L_T &= \mathrm{KL}(q(\boldsymbol{s}_T \mid \boldsymbol{s}_0) \| p_\theta(\boldsymbol{s}_T))\,. \end{aligned} \tag{10}$$

We can write

$$L_{t-1} = \mathbb{E}_{\boldsymbol{s}_0, \boldsymbol{\varepsilon}}\left[\frac{1}{2\sigma^2} \|\tilde{\boldsymbol{\mu}}(\boldsymbol{s}_t, \boldsymbol{s}_0) - \boldsymbol{\mu}_\theta(\boldsymbol{s}_t, t)\|^2\right]\,, \tag{11}$$

where the mean is a function of the previous and first state

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{s}_t, \boldsymbol{s}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{s}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{s}_0$$

and the variance

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\,.$$

It is helpful to consider the relationship between the initial and final states

$$\boldsymbol{s}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{s}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_s),$$

where $\boldsymbol{\varepsilon}_s = (\boldsymbol{\varepsilon}_x, \boldsymbol{\varepsilon}_y)$, so that we can rewrite the mean as

$$\tilde{\boldsymbol{\mu}}(\boldsymbol{s}_t, \boldsymbol{\varepsilon}_s) = \frac{1}{\sqrt{\alpha_t}}(\boldsymbol{s}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\varepsilon}_s)\,. \tag{12}$$

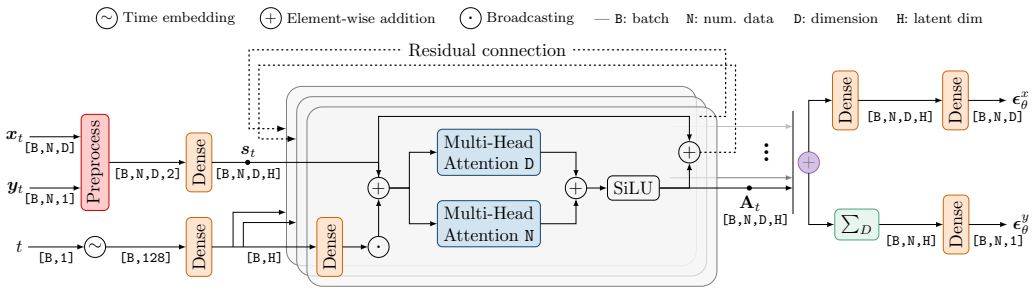


Figure 7: Architecture of the NDP's neural network noise models $\boldsymbol{\epsilon}_\theta^x$ and $\boldsymbol{\epsilon}_\theta^y$. Compared to Fig. 3 this architecture has two outputs, one to predict the corruption on the inputs $\boldsymbol{x}_t$ and one to predict the corruption on the function outputs $\boldsymbol{y}_t$. Both output share a lot of weights in the network and only bifurcate in the last few layers.

Equation 11 can now be expressed as

$$L_{t-1} = \mathbb{E}_{\boldsymbol{s}_0, \boldsymbol{\varepsilon}_s}\left[\frac{\beta_t^2}{2\sigma^2\alpha_t(1-\bar{\alpha}_t)}\|\boldsymbol{\varepsilon}_s - \boldsymbol{\epsilon}_\theta^s\|^2\right]. \tag{13}$$

Finally, since the variance schedule is fixed, and the edge terms are not found to improve empirical performance, our simplified training objective is given by

$$\mathcal{L}_\theta = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{y}_0, \boldsymbol{\varepsilon}_x, \boldsymbol{\varepsilon}_y, t}\left[\|\boldsymbol{\varepsilon}_x - \boldsymbol{\epsilon}_\theta^x(\boldsymbol{x}_t, \boldsymbol{y}_t, t)\|^2 + \|\boldsymbol{\varepsilon}_y - \boldsymbol{\epsilon}_\theta^y(\boldsymbol{x}_t, \boldsymbol{y}_t, t)\|^2\right]. \tag{14}$$

So far we have derived the objective of the 'full' NDP model (i.e., the model which diffuses both $\boldsymbol{x}$ and $\boldsymbol{y}$) given in Eq. (9) of the main paper. In Fig. 7 we detail the architecture for the noise models $\boldsymbol{\epsilon}_\theta^x(\cdot)$ and $\boldsymbol{\epsilon}_\theta^y(\cdot)$.

A.1 Deterministic input locations

In supervised learning the inputs are typically known in advance, which renders the joint generative distribution $p(\boldsymbol{x}_0, \boldsymbol{y}_0)$ of less importance than the conditional $p(\boldsymbol{y}_0 \mid \boldsymbol{x}_0)$. Therefore, if regression is the use-case of your NDP model, we recommend not to diffuse the inputs, but instead keeping them fixed, i.e. $\boldsymbol{x}_0 = \boldsymbol{x}_1 = \ldots = \boldsymbol{x}_T$. Mathematically, this is equivalent to setting $\boldsymbol{\varepsilon}_x$ deterministically to zero, which renders its corresponding predictor $\boldsymbol{\epsilon}_\theta^x(\cdot)$ unnecessary and thus reduces Eq. (14) to

$$\mathcal{L}_\theta = \mathbb{E}_{\boldsymbol{x}_0, \boldsymbol{y}_0, \boldsymbol{\varepsilon}_y, t}\left[\|\boldsymbol{\varepsilon}_y - \boldsymbol{\epsilon}_\theta^y(\boldsymbol{x}_0, \boldsymbol{y}_t, t)\|^2\right], \tag{15}$$

which matches Eq. (4) from the main paper. The noise model $\boldsymbol{\epsilon}_\theta^y(\cdot)$ now acts on the *uncorrupted* inputs $\boldsymbol{x}_0$ but still on corrupted observations $\boldsymbol{y}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{y}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\varepsilon}_y$.

B Algorithms

In this section we list pseudo-code for training and sampling NDPs. The code to handle the special case of deterministic input locations (Appendix A.1) is marked by 'OR'.

B.1 Training

The training procedure is concerned with fitting the parameters of the NDP's noise model neural network, with outputs $\boldsymbol{\epsilon}_\theta^x$ and $\boldsymbol{\epsilon}_\theta^y$, such that they can accurately predict the noise that was added to the corrupted function draws. In each iteration, we create a batch of datasets, each originating from a Gaussian process draw. We stress that the number of datapoints and even the input dimensionality of the datasets can vary within the training procedure.

Algorithm 1 Training

input Input distribution $q(\boldsymbol{x}_0)$ and covariance function k_ψ, with prior over hyperparameters p_ψ. Noise schedule β_t for $t \in \{1, 2, \ldots, T\}$. A loss function L (e.g., MSE or MAE).

begin
Precompute $\gamma_t = \sqrt{1 - \bar{\alpha}_t}$ and $\bar{\alpha}_t = \prod_{j=1}^{t}(1 - \beta_j)$.
for $i = 1, 2, \cdots, N_{\text{iter}}$ **do**
  Sample $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$, $\psi \sim p_\psi$, $\boldsymbol{y}_0 \sim \mathcal{N}(\mathbf{0}, k_\psi(\boldsymbol{x}_0, \boldsymbol{x}_0) + \sigma^2\mathbf{I})$.
  Sample $\boldsymbol{\varepsilon}_x, \boldsymbol{\varepsilon}_y \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $t \sim \mathcal{U}(\{1, \ldots, T\})$.
  Compute $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \gamma_t\boldsymbol{\varepsilon}_x$ and $\boldsymbol{y}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{y}_0 + \gamma_t\boldsymbol{\varepsilon}_y$.
  Update θ using gradient $\nabla_\theta \big[L(\boldsymbol{\varepsilon}_x, \boldsymbol{\epsilon}^x_\theta(\boldsymbol{x}_t, \boldsymbol{y}_t, t)) + L(\boldsymbol{\varepsilon}_y, \boldsymbol{\epsilon}^y_\theta(\boldsymbol{x}_t, \boldsymbol{y}_t, t))\big]$. Eq. (9)
  OR, in the case that we only diffuse on the observations: Eq. (4)
    Update θ using gradient $\nabla_\theta L(\boldsymbol{\varepsilon}_y, \boldsymbol{\epsilon}^y_\theta(\boldsymbol{x}_0, \boldsymbol{y}_t, t))$.

B.2 Prior and Conditional Sampling

In practice, the code to sample the prior is a special case of the conditional code with an empty context dataset. However, here we list them both for the clarity of the exposition.

Algorithm 2 Prior Sampling

input A pretrained NDP noise model $\boldsymbol{\epsilon}^x_\theta(\cdot)$ and $\boldsymbol{\epsilon}^y_\theta(\cdot)$. Precomputed γ_t and $\bar{\alpha}_t$ from a given noise schedule β_t.

begin
Sample a random initial state $\boldsymbol{x}_T$ and $\boldsymbol{y}_T$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
for $t = T, T-1, \ldots, 1$ **do**
  Sample using backward kernel: Eq. (3)

$$\begin{bmatrix}\boldsymbol{x}_{t-1}\\ \boldsymbol{y}_{t-1}\end{bmatrix} \sim \mathcal{N}\Big(\frac{1}{\sqrt{1-\beta_t}}\Big(\begin{bmatrix}\boldsymbol{x}_t\\ \boldsymbol{y}_t\end{bmatrix} - \frac{\beta_t}{\gamma_t}\begin{bmatrix}\boldsymbol{\epsilon}^x_\theta(\boldsymbol{x}_t, \boldsymbol{y}_t, t)\\ \boldsymbol{\epsilon}^y_\theta(\boldsymbol{x}_t, \boldsymbol{y}_t, t)\end{bmatrix}\Big), \frac{\gamma_t^2}{\gamma_{t-1}^2}\beta_t\mathbf{I}\Big).$$

  OR, in the case that we only diffuse on the observations and the inputs $\boldsymbol{x}_0$ are given:

$$\boldsymbol{y}_{t-1} \sim \mathcal{N}\big(\frac{1}{\sqrt{1-\beta_t}}(\boldsymbol{y}_t - \frac{\beta_t}{\gamma_t}\boldsymbol{\epsilon}^y_\theta(\boldsymbol{x}_0, \boldsymbol{y}_t, t)), \frac{\gamma_t^2}{\gamma_{t-1}^2}\beta_t\mathbf{I}\big).$$

return $(\boldsymbol{x}_0, \boldsymbol{y}_0)$

The conditional sampling algorithm is based on recent advances in image inpainting using diffusion models [19]. In Algorithm 3, for optimal results, we follow Lugmayr et al. [19] and use a re-sampling step which mixes the predictions with the context dataset. In all our experiments U is set to 5.

Algorithm 3 Conditional Sampling

input A context dataset $\mathcal{D} = \{([\boldsymbol{x}_0^c]_i \in \mathcal{X}, [\boldsymbol{y}_0^c]_i \in \mathbb{R})\}_{i=1}^N$. A pretrained NDP noise model $\boldsymbol{\epsilon}_\theta^x(\cdot)$ and $\boldsymbol{\epsilon}_\theta^y(\cdot)$. Precomputed γ_t and $\bar{\alpha}_t$ from a given noise schedule β_t.

begin
Let the augmented states be $\breve{\boldsymbol{x}}_T = [\boldsymbol{x}_T^c, \boldsymbol{x}_T]$ and $\breve{\boldsymbol{y}}_T = [\boldsymbol{y}_T^c, \boldsymbol{y}_T]$. Sample them according to $\breve{\boldsymbol{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\breve{\boldsymbol{y}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
Create a mask $m = [0, \dots, 0, 1, \dots, 1]$ indicating the positions of $\boldsymbol{x}_t^c$/$\boldsymbol{y}_t^c$ with 0s and $\boldsymbol{x}_t$/$\boldsymbol{y}_t$ with 1s.
for $t = T, T-1, \dots, 1$ **do**
  for $u = 1, \dots, U$ **do**
    Sample backward from augmented intermediate state: Eq. (3)

$$\begin{bmatrix}\breve{\boldsymbol{x}}_{t-1} \\ \breve{\boldsymbol{y}}_{t-1}\end{bmatrix} \sim \mathcal{N}\Big(\frac{1}{\sqrt{1-\beta_t}}\Big(\begin{bmatrix}\breve{\boldsymbol{x}}_t \\ \breve{\boldsymbol{y}}_t\end{bmatrix} - \frac{\beta_t}{\gamma_t}\begin{bmatrix}\boldsymbol{\epsilon}_\theta^x(\breve{\boldsymbol{x}}_t, \breve{\boldsymbol{y}}_t, t) \\ \boldsymbol{\epsilon}_\theta^y(\breve{\boldsymbol{x}}_t, \breve{\boldsymbol{y}}_t, t)\end{bmatrix}\Big), \frac{\gamma_t^2}{\gamma_{t-1}^2}\beta_t\mathbf{I}\Big)$$

    Select unknown part from the augmented state using the mask:

$$\boldsymbol{x}_{t-1} = m \odot \breve{\boldsymbol{x}}_{t-1} \quad \text{and} \quad \boldsymbol{y}_{t-1} = m \odot \breve{\boldsymbol{y}}_{t-1}$$

    Sample context points $(t-1)$-steps forward from context dataset:

$$\begin{bmatrix}\boldsymbol{x}_{t-1}^c \\ \boldsymbol{y}_{t-1}^c\end{bmatrix} \sim \mathcal{N}\Big(\begin{bmatrix}\boldsymbol{x}_{t-1}^c \\ \boldsymbol{y}_{t-1}^c\end{bmatrix} \mid \sqrt{\bar{\alpha}_{t-1}}\begin{bmatrix}\boldsymbol{x}_0^c \\ \boldsymbol{y}_0^c\end{bmatrix}, \gamma_{t-1}^2\mathbf{I}_n\Big)$$

    Update the augmented intermediate state using the forward sampled context dataset and the new intermediate state:

$$\breve{\boldsymbol{x}}_{t-1} = \begin{bmatrix}\boldsymbol{x}_{t-1}^c \\ \boldsymbol{x}_{t-1}\end{bmatrix} \quad \text{and} \quad \breve{\boldsymbol{y}}_{t-1} = \begin{bmatrix}\boldsymbol{y}_{t-1}^c \\ \boldsymbol{y}_{t-1}\end{bmatrix}$$

    Diffuse forward by one step

$$\begin{bmatrix}\breve{\boldsymbol{x}}_t \\ \breve{\boldsymbol{y}}_t\end{bmatrix} \sim \mathcal{N}\Big(\sqrt{1-\beta_t}\begin{bmatrix}\breve{\boldsymbol{x}}_{t-1} \\ \breve{\boldsymbol{y}}_{t-1}\end{bmatrix}, \beta_t\mathbf{I}\Big)$$

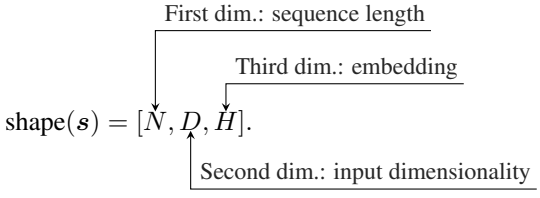
return $(\boldsymbol{x}_0, \boldsymbol{y}_0)$

C PROOFS

In this section, we shall formally demonstrate that NDP's noise model adheres to the symmetries associated with permutations of the datapoint orderings and the permutations of the input dimensions. We focus our attention to the full noise model of Fig. 7 because Fig. 3 is a simplification of it in which we only keep a single output. We start by proving properties of its main building block: the bi-dimensional attention block. We can then straightforwardly prove the equivariance and invariance properties that hold for the NDP's noise model. Before that, we start by setting the notation.

C.1 NOTATION, DEFINITIONS AND PRELIMINARY LEMMAS

Notation. Let $\boldsymbol{s} \in \mathbb{R}^{N \times D \times H}$ be a tensor of rank (or dimension) three, where we refer to each dimension according to the following convention:



In the next definitions we will use NumPy-based indexing and slicing notation. We assume the reader is familiar with this convention. Most notably, we use a colon (:) to reference every element in a dimension.

Definition 1. *Let Π_N be the set of all permutations of indices $\{1, \dots, N\}$. Let $\pi_n \in \Pi_N$ and $\boldsymbol{s} \in \mathbb{R}^{N \times D \times H}$. Then $(\pi_n \circ \boldsymbol{s}) \in \mathbb{R}^{N \times D \times H}$ denotes a tensor where the ordering of indices in the first dimension are reshuffled (i.e., permuted) according to π_n. We write*

$$\pi_n \circ \boldsymbol{s} = \boldsymbol{s}_{\pi_n(1), \pi_n(2), \dots, \pi_n(N); :; :}$$

Definition 2. *Let Π_D be the set of all permutations of indices $\{1, \dots, D\}$. Let $\pi_d \in \Pi_D$ and $\boldsymbol{s} \in \mathbb{R}^{N \times D \times H}$. Then $(\pi_d \circ \boldsymbol{s}) \in \mathbb{R}^{N \times D \times H}$ denotes a tensor where the ordering of indices in the second dimension are reshuffled (i.e., permuted) according to π_d. We write*

$$\pi_d \circ \boldsymbol{s} = \boldsymbol{s}_{:; \pi_d(1), \pi_d(2), \dots, \pi_d(D); :}$$

Definition 3. *A function $f : \mathbb{R}^{N \times D \times H} \to \mathbb{R}^{N \times D \times H}$ is equivariant to Π if for any permutation $\pi \in \Pi$, we have*

$$f(\pi \circ \boldsymbol{s}) = \pi \circ f(\boldsymbol{s}).$$

Definition 4. *A function $f : \mathbb{R}^{N \times D \times H} \to \mathbb{R}^{N \times D \times H}$ is invariant to Π if for any permutation $\pi \in \Pi$, we have*

$$f(\pi \circ \boldsymbol{s}) = f(\boldsymbol{s}).$$

Invariance in layman's terms means that the output is not affected by a permutation of the inputs.

Lemma 1. *The composition of equivariant functions is equivariant.*

Proof. Let f and g be equivariant to Π, then for all $\pi \in \Pi$:

$$(f \circ g)(\pi \circ \boldsymbol{s}) = f(g(\pi \circ \boldsymbol{s})) = f(\pi \circ g(\boldsymbol{s})) = \pi \circ f(g(\boldsymbol{s})),$$

which shows that the composition, $f \circ g$, is equivariant as well. □

Lemma 2. *An element-wise operation between equivariant functions remains equivariant.*

Proof. Let f and g be equivariant to Π, then for all $\pi \in \Pi$ and an element-wise operation $\oplus$ (e.g., addition), we have

$$(f \oplus g)(\pi \circ \boldsymbol{s}) = f(\pi \circ \boldsymbol{s}) \oplus g(\pi \circ \boldsymbol{s}) = (\pi \circ f(\boldsymbol{s})) \oplus (\pi \circ g(\boldsymbol{s})) = \pi \circ (f \oplus g).$$

which shows that the composition, $f \oplus g$, is equivariant as well. □

Lemma 3. *A function $f : \mathbb{R}^{N \times D \times H} \to \mathbb{R}^{N \times D \times H}$ which applies the same function g to all its rows, i.e. $f : \boldsymbol{s} \mapsto [g(\boldsymbol{s}_{1;:;:}), g(\boldsymbol{s}_{2;:;:}), \dots, g(\boldsymbol{s}_{N;:;:})]$ with $g : \mathbb{R}^{D \times H} \to \mathbb{R}^{D \times H}$ is equivariant in the first dimension.*

Proof. Follows immediately from the structure of f. □

C.2 Bi-dimensional Attention Block

With the notation, definitions and lemmas in place, we now prove that the bi-dimensional attention block is equivariant in its first and second dimension, respectively to permutations in the set Π_N and Π_D. We start this section by formally defining the bi-dimensional attention block and its main components attention components Attn_N and Attn_D. Finally, in Appendix C.3 we prove the properties of the noise models $\boldsymbol{\epsilon}_\theta^x$ and $\boldsymbol{\epsilon}_\theta^y$ making use of the results in this section.

Definition 5. *Ignoring the batch dimension B, let $\mathbf{A}_t : \mathbb{R}^{N \times D \times H} \to \mathbb{R}^{N \times D \times H}; \boldsymbol{s} \mapsto \mathbf{A}_t(\boldsymbol{s})$ be the bi-dimensional attention block. As illustrated in Fig. 7, it operates on three dimensional tensors in $\mathbb{R}^{N \times D \times H}$ and applies attention across the first and second dimension using Attn_N and Attn_D, respectively. The final output $\mathbf{A}_t(\boldsymbol{s})$ is obtained by summing the two attention outputs, before applying an element-wise non-linearity.*

We now proceed by defining the component Attn_D and its properties. Subsequently, in Definition 7 we define Attn_N and its properties. Finally, we combine both to prove Proposition 1 in the main paper about the bi-dimensional attention block.

Definition 6. *Let $\text{Attn}_D : \mathbb{R}^{N\times D\times H} \to \mathbb{R}^{N\times D\times H}$ be a self-attention block [35] acting across D. Let σ be a softmax activation function operating on the last dimension of a tensor. Then, Attn_D is defined as*

$$\textit{Attn}_D(\boldsymbol{s})[n,d,h] = \sum_{d'=1}^{D} \boldsymbol{\Sigma}_{n,d,d'}(\boldsymbol{s})\boldsymbol{s}^v_{n,d',h}, \quad \textit{where} \quad \boldsymbol{\Sigma}_{n,d,d'}(\boldsymbol{s}) = \sigma\big(\frac{1}{\sqrt{H}}\sum_l \boldsymbol{s}^k_{n,d,\ell}\boldsymbol{s}^q_{n,d',\ell}\big)$$

given a linear projection of the inputs $\boldsymbol{s}$ which maps them into keys (k), queries (q) and values (v)

$$\boldsymbol{s}^k_{n,d,\ell} = \sum_j \boldsymbol{s}_{n,d,j}\mathbf{W}^k_{j,\ell}, \quad \boldsymbol{s}^q_{n,d',\ell} = \sum_j \boldsymbol{s}_{n,d,j}\mathbf{W}^q_{j,\ell}, \quad \boldsymbol{s}^v_{n,d',h} = \sum_j \boldsymbol{s}_{n,d,j}\mathbf{W}^v_{j,\ell}.$$

Proposition 3. *Attn_D is equivariant to Π_N and Π_D (i.e., across sequence length and input dimensionality).*

Proof. We prove the equivariance to Π_N and Π_D separately. First, from the definition we can see that Attn_D is a function that acts on each element row of $\boldsymbol{s}$ separately. Thus, by Lemma 3, Attn_D is equivariant to Π_N.

Next, we want to prove equivariance to Π_D. We want to show that for all $\pi_d \in \Pi_D$:

$$\text{Attn}_D(\pi_d \circ \boldsymbol{s}) = \pi_d \circ \text{Attn}_D(\boldsymbol{s}).$$

The self-attention mechanism consists of a matrix multiplication of the attention matrix Σ and the projected inputs. We start by showing that the attention matrix is equivariant to permutations in Π_D

$$\begin{aligned}
\boldsymbol{\Sigma}_{n,d,d'}(\pi_d \circ \boldsymbol{s}) &= \sigma\Big(\frac{1}{\sqrt{H}}\sum_\ell \big(\pi_d \circ \boldsymbol{s}\big)^k_{n,d,\ell}\big(\pi_d \circ \boldsymbol{s}\big)^q_{n,d,\ell}\Big) \\
&= \sigma\Big(\frac{1}{\sqrt{H}}\sum_{\ell,j} \big(\boldsymbol{s}_{n,\pi_d(d),j}\mathbf{W}^k_{j,\ell}\big)\big(\boldsymbol{s}_{n,\pi_d(d'),j}\mathbf{W}^q_{j,\ell}\big)\Big) \\
&= \boldsymbol{\Sigma}_{n,\pi_d(d),\pi_d(d')}(\boldsymbol{s})
\end{aligned}$$

It remains to show that the final matrix multiplication step restores the row-equivariance

$$\begin{aligned}
\text{Attn}_D(\pi_d \circ \boldsymbol{s}) &= \sum_{d'} \boldsymbol{\Sigma}_{n,d,d'}(\pi_d \circ \boldsymbol{s})\big(\pi_d \circ \boldsymbol{s}\big)^v_{n,d',h} \\
&= \sum_{d'} \boldsymbol{\Sigma}_{n,\pi_d(d),\pi_d(d')}(\boldsymbol{s})\boldsymbol{s}^v_{n,\pi_d(d'),h} \\
&= \sum_{d'} \boldsymbol{\Sigma}_{n,\pi_d(d),d'}(\boldsymbol{s})\boldsymbol{s}^v_{n,d',h} = \pi_d \circ \text{Attn}_D(\boldsymbol{s}).
\end{aligned}$$

This concludes the proof. □

We continue by defining and proving the properties of the second main component of the bi-dimensional attention block: Attn_N.

Definition 7. *Let $\text{Attn}_N : \mathbb{R}^{N\times D\times H} \to \mathbb{R}^{N\times D\times H}$ be a self-attention block [35] acting across N (i.e. the sequence length). Let σ be a softmax activation function operating on the last dimension of a tensor. Then Attn_N is defined as*

$$\textit{Attn}_N(\boldsymbol{s})[n,d,h] = \sum_{n'=1}^{N} \boldsymbol{\Sigma}_{n,d,n'}(\boldsymbol{s})\boldsymbol{s}^v_{n',d,h}, \textit{ where} \quad \boldsymbol{\Sigma}_{n,d,n'}(\boldsymbol{s}) = \sigma\big(\frac{1}{\sqrt{H}}\sum_l \boldsymbol{s}^k_{n,d,\ell}\boldsymbol{s}^q_{n',d,\ell}\big)$$

given a linear projection of the inputs $\boldsymbol{s}$ which maps them into keys (k), queries (q) and values (v)

$$\boldsymbol{s}^k_{n,d,\ell} = \sum_j \boldsymbol{s}_{n,d,j}\mathbf{W}^k_{j,\ell}, \quad \boldsymbol{s}^q_{n',d,\ell} = \sum_j \boldsymbol{s}_{n',d,j}\mathbf{W}^q_{j,\ell}, \quad \boldsymbol{s}^v_{n',d,h} = \sum_j \boldsymbol{s}_{n',d,j}\mathbf{W}^v_{j,\ell}.$$

Proposition 4. *Attn_N is equivariant to Π_N and Π_D (i.e., across sequence length and input dimensionality).*

Proof. Follows directly from Proposition 3 after transposing the first and second dimension of the input. □

Finally, we have the necessary ingredients to prove Proposition 1 from the main paper.

Proposition 5. *The bi-dimensional attention block* $\mathbf{A}_t$ *is equivariant to* Π_D *and* Π_N.

Proof. The bi-dimensional attention block simply adds the output of Attn_D and Attn_N, followed by an element-wise non-linearity. Therefore, as a direct application of Lemma 2 the complete bi-dimensional attention block remains equivariant to Π_N and Π_D. □

C.3 NDP NOISE MODEL

By building on the equivariant properties of the bi-dimensional attention block, we prove the equivariance and invariance of the NDP's noise models, denoted by $\boldsymbol{\epsilon}_\theta^x$ and $\boldsymbol{\epsilon}_\theta^y$, respectively. We refer to Fig. 7 for their definition. In short, $\boldsymbol{\epsilon}_\theta^x$ consists of adding the output of several bi-dimensional attention blocks followed by dense layers operating on the last dimension. Similarly, $\boldsymbol{\epsilon}_\theta^y$ is constructed by summing the bi-dimensional attention blocks, but is followed by a summation over D before applying a final dense layer.

The following propositions hold:

Proposition 6. *The function* $\boldsymbol{\epsilon}_\theta^x$ *is equivariant to* Π_D *and* Π_N.

Proof. The output $\boldsymbol{\epsilon}_\theta^x$ is formed by element-wise summing the output of bi-dimensional attention layers. Directly applying Lemma 1 and Proposition 5 completes the proof. □

Proposition 7. *The function* $\boldsymbol{\epsilon}_\theta^y$ *is equivariant to* Π_N.

Proof. The summation over D does not affect the equivariance over Π_N from the bi-dimensional attention blocks as it can be cast as a row-wise operation. □

Proposition 8. *The function* $\boldsymbol{\epsilon}_\theta^y$ *is invariant to* Π_D.

Proof. Follows from the equivariance of the bi-dimensional blocks and [41, Thm. 7]. □

C.4 CONSISTENCY

Consistency[2] is achieved when marginalising over the conditional distribution of a random variable y^0, we recover its original distribution.

$$p(y^0) = \int p(y^0|y^1)p(y^1)\mathrm{d}y^1. \tag{16}$$

The reverse process of a diffusion model begins at T in a state of white noise, and progresses towards step 0. The features represented at T, denoted $\mathbf{y}_T := (y_T^0, y_T^1)$ (for simplicity in this proof we only pick two), are by construction a fully consistent stochastic process. We have $p(\mathbf{y}_T) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the consistency is trivially satisfied.

The progression from step t to $t-1$, involves using a deep neural network to estimate the noise present, and subtracting that noise from the signal. This procedure leaves us with a new probability distribution. Specifically, following Ho et al. [10], this is constructed as follows:

$$p_\theta(\boldsymbol{y}_{t-1} \mid \boldsymbol{y}_t, t) := \mathcal{N}\Big(\boldsymbol{y}_{t-1};\, \mu_\theta(\boldsymbol{y}_t), \beta_t^2 \mathbf{I}\Big). \tag{17}$$

[2] see also https://yanndubs.github.io/Neural-Process-Family/text/CNPF.html for a concise overview.

where the new mean function $\mu(\boldsymbol{y}_t)$ is a deterministic function, controlled by a neural network, and $\boldsymbol{y}_t$ represents the data vector at step t. Notice the conditional independence of y^0_{t-1} and y^1_{t-1} given $\boldsymbol{y}_t$ in the reverse process

$$\mathcal{N}\big(y^0_{t-1}; \mu^0_\theta(\begin{bmatrix} y^0_t \\ y^1_t \end{bmatrix}), \beta^2_t\big) \cdot \mathcal{N}\big(y^1_{t-1}; \mu^1_\theta(\begin{bmatrix} y^0_t \\ y^1_t \end{bmatrix}), \beta^2_t\big), \tag{18}$$

where we use $\mu^0_\theta(\cdot)$ and $\mu^1_\theta(\cdot)$ to select the first and second entry of the neural network output.

Starting from the RHS of Eq. (16)

$$\int p(y^0_{t-1} \mid y^1_{t-1})\, p(y^1_{t-1}) \mathrm{d}y^1_{t-1} \tag{19}$$

$$= \int_{y^1_{t-1}} \left(\int_{\boldsymbol{y}_t} p(\boldsymbol{y}_t) p(y^0_{t-1} \mid y^1_{t-1}, \boldsymbol{y}_t) \mathrm{d}\boldsymbol{y}_t \right) p(y^1_{t-1}) \mathrm{d}y^1_{t-1} \tag{20}$$

$$= \int_{y^1_{t-1}} \left(\int_{\boldsymbol{y}_t} p(\boldsymbol{y}_t) p(y^0_{t-1} \mid \boldsymbol{y}_t) \mathrm{d}\boldsymbol{y}_t \right) p(y^1_{t-1}) \mathrm{d}y^1_{t-1} \tag{21}$$

$$= \int_{\boldsymbol{y}_t} p(\boldsymbol{y}_t) p(y^0_{t-1} \mid \boldsymbol{y}_t) \mathrm{d}\boldsymbol{y}_t = p(y^0_{t-1}). \tag{22}$$

In Eq. (20) we marginalised over the previous function values $\boldsymbol{y}_t$. This allowed us to make use of the conditional independence between y^0_{t-1} and y^1_{t-1} given $\boldsymbol{y}_t$ in Eq. (21). Finally, the random variable y^1_{t-1} integrates to 1, which leads to the LHS of Eq. (16). This argument can be repeated for each step, until we reach $\boldsymbol{y}_0$. The NDP therefore offers samples drawn from a consistent set of conditional distributions.

Summarising Consistency of the NDP samples is ensured because *the neural network plays no role in the conditioning step*. It is therefore infeasible for it to be responsible for generating inconsistent draws. By contrast, if we used a generic neural network to generate the conditional probability distributions itself, there is no such guarantee to this consistency.

D Additional Information on the Experiments

D.1 Experimental setup

Here we provide more detailed information describing how the numerical experiments were conducted.

All experiments share the same model architecture illustrated in Figure 3, there are however a number of model parameters that must be chosen. An L1 (i.e., Mean Absolute Error, MAE) loss function was used throughout. We use five bi-dimensional attention blocks, each consisting of multi-head self-attention blocks [35] containing a representation dimensionality of $H = 64$ and 8 heads. Each experiment used either 500 or $1,000$ diffusion steps, where we find larger values produce more accurate samples at the expense of computation time. Following Nichol et al. [24] we use a cosine-based scheduling of β_t during training. The Adam optimiser is used throughout. Our learning rate follows a cosine-decay function, with a 20 epochs linear learning rate warm-up to a maximum learning rate of $\eta = 0.001$ before decaying. All NDP models were trained for 250 epochs with the exception of the lengthscale marginalisation experiment, which was trained for 500 epochs. Each epoch contained 4096 example training $(\boldsymbol{y}_0, \boldsymbol{x}_0)$ pairs. Training data was provided in batches of 32, with each batch containing data with the same kernel hyperparameters but different realisations of prior GP samples. The complete configuration for each experiment is given in Table 2

Experiments were conducted on a 32-core machine and utilised a single Tesla V100-PCIE-32GB GPU. Training of each model used in the experiments takes no longer than 30 minutes.

Time embedding The diffusion step t is a crucial input of the neural network noise estimator as the model needs to be able to differentiate between noise added at the start or the end of the process.

Table 2: Experiment configuration and training time. Bold values indicate a deviation from the fiducial values used in the regression experiments.

Experiment	Regression	Hyperparameter marginalisation	Step	High dim BO	1D opt
Epochs	250	**500**	250	250	250
Total samples seen	1024k	**2048k**	1024k	1024k	1024k
Batch size	32	32	32	32	32
Loss	L1	L1	L1	L1	L1
LR decay	cosine	cosine	cosine	cosine	cosine
LR init	0.001	0.001	0.001	0.001	0.001
LR warmup epochs	20	20	20	20	20
Num blocks	5	5	5	5	5
Representation dim (H)	64	64	64	64	64
Num heads	8	8	8	8	8
Num timesteps (T)	500	**1000**	**1000**	500	**2000**
Num data per batch (N)	100	100	100	**256**	100
Deterministic inputs	True	True	True	True	**False**
Training time (mins)	17	33	16	21	16

Following Vaswani et al. [35] we use a cyclic 128-dimensional encoding vector for each step

$$t \mapsto [\sin(10^{\frac{0\times4}{63}}t), \sin(10^{\frac{1\times4}{63}}t), \ldots, \sin(10^{\frac{64\times4}{63}}t), \cos(10^{\frac{0\times4}{63}}t), \ldots, \cos(10^{\frac{64\times4}{63}}t)] \in \mathbb{R}^{128}$$

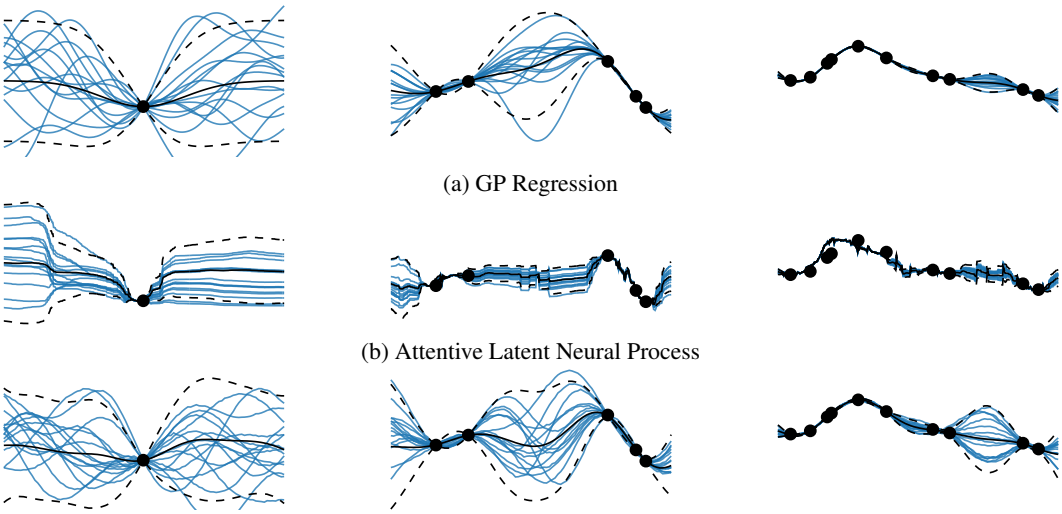
Data In our experiments, we provide training data to the models in the form of unique prior samples from a ground truth GP model. The input for each prior sample, $\boldsymbol{x}_0$, may be randomly sampled or deterministically spaced across the relevant input range. For all but the last experiment we deterministically space $\boldsymbol{x}_0$ during training and inference in the hypercube $[-1, 1]^D$, where D is the dimensionality of the input. For $D \leq 2$ the spacing of points is is a simple linear spacing (a grid for $D = 2$), in $D > 2$ we sample $\boldsymbol{x}_0$ according to a Halton sequence in the hypercube. The corresponding output $\boldsymbol{y}_0$ is sampled from a GP prior $\boldsymbol{y}_0|\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0}, k_\psi(\boldsymbol{x}_0, \boldsymbol{x}_0) + \sigma^2\big)$ where the kernel is a stationary kernel (Matern$\frac{3}{2}$, Matérn-$5/2$ or exponentiated quadratic, indicated in each experiment) with hyperparameters $\psi = [\boldsymbol{\ell}, \sigma_k^2]$ corresponding to the automatic relevance determination (ARD) lengthscales and kernel variance respectively. Kernel hyperparameters are fixed for all experiments except for the hyperparameter marginalisation experiment where we allow the model to observe data from a variety of different lengthscales during training. The noise variance is fixed to $\sigma^2 = 10^{-6}$ and the kernel variance is fixed to $\sigma_k^2 = 1.0$ throughout.

D.2 One-Dimensional Conditional samples

In the experiment, we use GPflow [34] for the GP regression model using a kernel that matches the training data: a squared exponential with lengthscale set to 0.2. The other baseline, namely the Attentive Latent NP, is a pretrained model which was trained on a dataset with the same configuration. The ALNP model originates from the reference implementation of Dubois et al. [3]. Figure 8 shows additional conditional samples.

D.3 High-dimensional Bayesian Optimisation

In this experiment, we perform Bayesian optimisation on the Hartmann 3D and 6D objectives. We re-scale, without loss of generality, the inputs of the objectives such that the search space is $[-1, 1]^D$ for both. We refer to https://www.sfu.ca/~ssurjano/hart3.html and https://www.sfu.ca/~ssurjano/hart6.html for more details about Hartmann 3D and 6D, respectively.



(a) GP Regression

(b) Attentive Latent Neural Process

(c) Neural Diffusion Process (ours)

Figure 8: **1D regression:** The blue curves are posterior samples from different probabilistic models. We also plot the empirical mean and two standard deviations of the samples in black. From left to right we increase the number of data points (black dots) and notice how the process gets closer to the true underlying function.

The baseline models, GPR and Random, originate from Trieste [1] —a TensorFlow/GPflow based Bayesian Optimisation Python package. We compare them against two NDP models: Fixed and Marginalised. The Fixed NDP model is trained on Matérn-5/2 samples with a fixed lengthscale set to 0.5 along all dimensions. The Marginalised NDP model is trained on Matérn-5/2 samples originating from different lengthscales, where the lengthscales are drawn from a log-Normal prior $\log \mathcal{N}(0, 1)$.

D.4 Comparison to Neural Processes

Neural processes [8; 7] use an encoder and decoder architecture. The encoder is a neural network which operates on a dataset $\mathcal{D} = (X, Y)$ to output a dataset representation. Using this representation, a decoder predicts the function output at a test location. For a NP this is given by

$$p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}) = \prod_n \mathcal{N}(y_n^* \mid dec_\mu(x_n^*, enc(\mathcal{D})), dec_{\sigma^2}(x^*, enc(\mathcal{D})))$$

A latent NP works similarly but parameterises a latent variable z, over which we marginalise to make a prediction:

$$p(\boldsymbol{y}^* \mid \boldsymbol{x}^*, \mathcal{D}) = \int_z \prod_n \mathcal{N}\left(y_n^* \mid dec_\mu(x_n^*, z), dec_{\sigma^2}(x_n^*, z)\right) p(z \mid enc(\mathcal{D}))\, \mathrm{d}z.$$

An attentive Neural processes (ANPs) [12] uses an attention mechanism [35] to parameterise the encoder and decoder networks. While ANPs perform better than plain (L)NPs, they suffer from two key weaknesses. Firstly, their predictions result in jittery functions as a result of shifting attention patterns. This behaviour is well-known and can also be observed in the top row of Fig. 10 in the supplementary. Secondly, ANP do not encode dimensionality invariance in their architecture, which we show is critical for tabular data applications.

Figure 10 shows example fits for all the different models. We used `https://github.com/wesselb/neuralprocesses` for the NP baselines and evaluation framework.

D.5 Global optimisation

E Code

Full source code will be made publicly available upon acceptance.

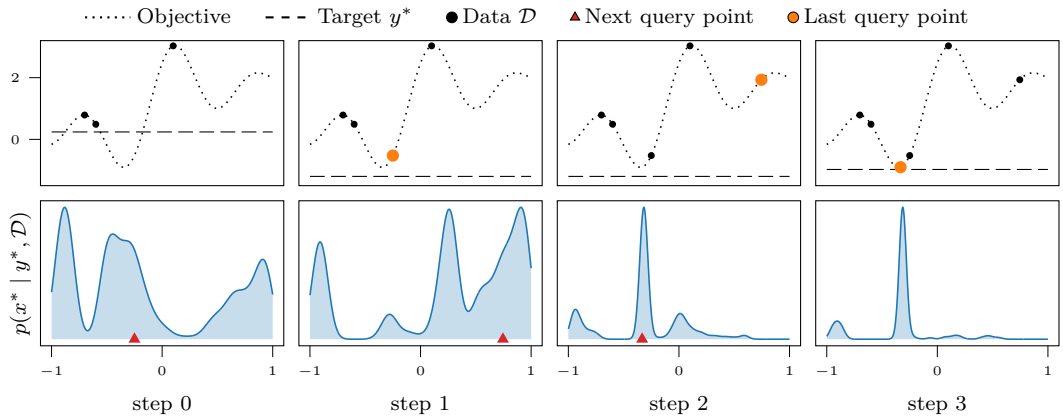


Figure 9: Global optimisation by diffusing the input locations. We condition the model at each step on a target y^* value (dashed line), and use the NDP to sample from $p(x^* \mid y^*, \mathcal{D})$. The panels in the upper row illustrate the progression of the optimisation. Bottom row shows the distribution $p(x^* \mid y^*, \mathcal{D})$ and the red triangles mark the selected query point.

PREPROCESSING METHOD

The NDP's noise model starts by rearranging the inputs $\boldsymbol{x}_t \in \mathbb{R}^{N \times D}$ and $\boldsymbol{y}_t \in \mathbb{R}^N$ to be of shape `[N, D, 2]`, as highlited in the red box in Figs. 3 and 7. Below we show the code which is responsible for this preprocessing:

```
import tensorflow as tf

def preprocess(x: tf.Tensor, y: tf.Tensor) -> tf.Tensor:
    """
    Transform inputs to split out the x dimensions for dimension-
    agnostic processing.

    :param x: [B, N, D]
    :param y: [B, N, 1]
    :return: [B, N, D, 2]
    """
    D = tf.shape(x)[-1]
    x = tf.expand_dims(x, axis=-1)
    y = tf.repeat(tf.expand_dims(y, axis=-1), D, axis=2)
    return tf.concat([x, y], axis=-1)
```

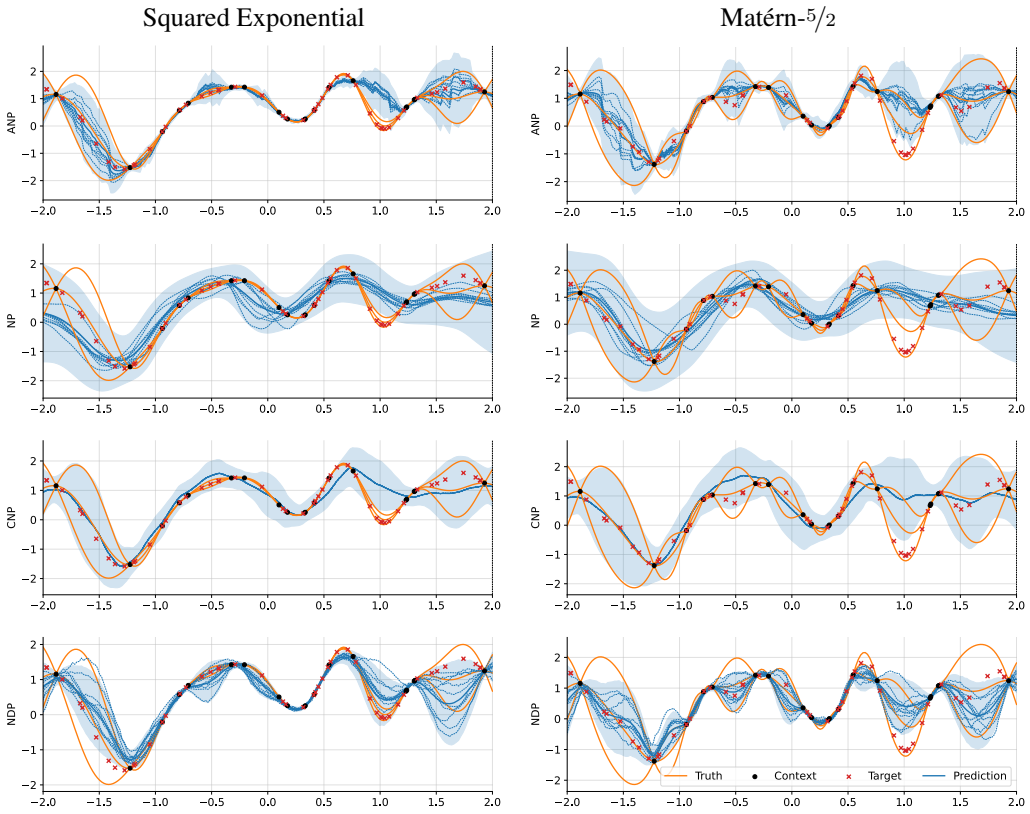


Figure 10: Model predictions on 1D datasets. Black dots: context points (conditioning points). Red crosses: Test/target points. Blue lines: sampels of the model, mean and 2 std. dev. Orange: mean and 2 std dev from the true underlying GP model.