Appendix of Deep Stochastic Processes via Functional Markov Transition Operator

A Proofs

A.1 Proof of proposition 4.1 (See page 4)

Proposition 4.1. *Let $F_\theta(\cdot; x, z) : \mathcal{Y} \mapsto \mathcal{Y}$ denote an invertible transformation between outputs, parameterized by the input and latent. If $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is a valid Stochastic Process (SP) (i.e. it satisfies Equation (1) and Equation (2)) and*

$$y^{(0)}_{1:n} \sim \{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}, \quad z \sim p_\theta(z), \quad y_i = F_\theta(y^{(0)}_i; x_i, z). \tag{6}$$

then $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ is also a valid SP.

Proof. Given an input x_i and a latent variable z, y_i is obtained by applying an invertible transformation F_θ to $y^{(0)}_i$. Therefore, we can represent $p(y_i|y^{(0)}_i; x_i, z)$ using the Dirac delta:

$$p(y_i|y^{(0)}_i; x_i, z) = \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \tag{16}$$

Furthermore, according to Equation (6), these transformations are independently applied given the latent variable z. So we have:

$$p(y_{1:n}|y^{(0)}_{1:n}; x_{1:n}, z) = \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \tag{17}$$

Hence

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) = \int p_\theta(z) p_\theta(y_{1:n}|y^{(0)}_{1:n}; x_{1:n}, z) \mathrm{d}z \tag{18}$$

$$= \int p_\theta(z) \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \mathrm{d}z \tag{19}$$

□

As per Proposition 4.3, $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$ is exchangeable and consistent. Furthermore, because $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is a valid SP, $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ is also a valid SP

A.2 Proof of proposition 4.2 (See page 4)

Proposition 4.2. *If the collection of marginals before transition $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ is consistent and exchangeable (as per Equations (1) and (2)) and the collection of marginal Markov transition operators (MMTOs) $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$ is also consistent and exchangeable (as per Equations (7) and (8)), then the collection of marginals after transition $\{p_{x_{1:n}}(y_{1:n})\}$ is also consistent and exchangeable, hence defining a valid SP.*

Proof. According to the definitions of consistency and exchangeability for both the collection of marginals $\{p_{x_{1:n}}(y^{(0)}_{1:n})\}_{x_{1:n}}$ and the collection of transition operators $\{p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n})\}_{x_{1:n}}$, we have the following (Equations (1), (2), (7) and (8)):

$$\int p_{x_{1:n}}(y^{(0)}_{1:n}) \mathrm{d}x_{m+1:n} = p_{x_{1:m}}(y^{(0)}_{1:m})$$

$$p_{\pi(x_{1:n})}(\pi(y^{(0)}_{1:n})) = p_{x_{1:n}}(y^{(0)}_{1:n})$$

$$\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) \, \mathrm{d}y_{m+1:n} = p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m})$$

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) = p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n}))$$

where $1 < m < n$.

The Markov transition on the function outputs are described by:

$$p_{x_{1:n}}(y_{1:n}) = \int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) dy^{(0)}_{1:n}$$

For any finite sequence $x_{1:n}$, we have

$$\begin{aligned}
\int p_{x_{1:n}}(y_{1:n}) \mathrm{d}y_{m+1:n} &= \int \Big(\int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) \mathrm{d}y^{(0)}_{1:n} \Big) \mathrm{d}y_{m+1:n} \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) \Big(\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) \mathrm{d}y_{m+1:n} \Big) \mathrm{d}y^{(0)}_{1:n} \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m}) \mathrm{d}y^{(0)}_{1:n} \\
&= \int \Big(\int p_{x_{1:n}}(y^{(0)}_{1:n}) \mathrm{d}y^{(0)}_{m+1:n} \Big) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m}) \mathrm{d}y^{(0)}_{1:m} \\
&= \int p_{x_{1:m}}(y^{(0)}_{1:m}) p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m}) \mathrm{d}y^{(0)}_{1:m} \\
&= \int p_{x_{1:m}}(y_{1:m}) \mathrm{d}y^{(0)}_{1:m} \qquad (20)
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
p_{\pi(x_{1:n})}(\pi(y_{1:n})) &= \int p_{\pi(x_{1:n})}(\pi(y^{(0)}_{1:n})) p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n})) \mathrm{d}\pi(y^{(0)}_{1:n}) \\
&= \int p_{x_{1:n}}(y^{(0)}_{1:n}) p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n}) \mathrm{d}y^{(0)}_{1:n} \\
&= p_{x_{1:n}}(y_{1:n}) \qquad (21)
\end{aligned}$$

Therefore, the collection of marginals $\{p_{x_{1:n}}(y_{1:n})\}_{x_{1:n}}$ are both consistent and exchangeable (Equations (20) and (21)), hence defining a valid SP.

□

A.3 Proof of proposition 4.3 (See page 4)

Proposition 4.3. MMTOs *in the form of Equation* (9) *are consistent and exchangeable.*

Proof. Recall that MMTOs in Equation (9) write as:

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) = \int p_\theta(z) \prod_{i=1}^{n} \delta(y_i - F_\theta(y^{(0)}_i; x_i, z)) \, \mathrm{d}z, \qquad (22)$$

It is a special case of a more general form:

$$p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) = \int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) \mathrm{d}z, \qquad (23)$$

Equation (23) becomes Equation (22) when $p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) = \delta(y_i - F_\theta(y^{(0)}_i; x_i, z))$ is a δ-distribution. Below we prove that MMTOs are consistent and exchangeable for the general form. We have

$$\begin{aligned}
\int p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) \mathrm{d}y_{m+1:n} &= \int \Big(\int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) \, \mathrm{d}z \Big) \mathrm{d}y_{m+1:n} \\
&= \int p_\theta(z) \prod_{i=1}^{m} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) \Big(\int \prod_{i=m+1}^{n} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) \mathrm{d}y_{m+1:n} \Big) \, \mathrm{d}z \\
&= \int p_\theta(z) \prod_{i=1}^{m} p_\theta(y_i \,|\, y^{(0)}_i, x_i, z) \, \mathrm{d}z \\
&= p_{x_{1:m}}(y_{1:m}|y^{(0)}_{1:m};\theta) \qquad (24)
\end{aligned}$$

and

$$
\begin{aligned}
p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n});\theta) &= \int p_\theta(z) \prod_{i=\pi(1)}^{\pi(n)} p_\theta(y_i \,|\, y_i^{(0)}, x_i, z)\mathrm{d}z \\
&= \int p_\theta(z) \prod_{i=1}^{n} p_\theta(y_i \,|\, y_i^{(0)}, x_i, z)\mathrm{d}z \\
&= p_{x_{1:n}}(y_{1:n}|y^{(0)}_{1:n};\theta) \qquad (25)
\end{aligned}
$$

Therefore, the collection $\{p_{\pi(x_{1:n})}(\pi(y_{1:n})|\pi(y^{(0)}_{1:n});\theta)\}_{x_{1:n}}$ is both consistent (Equation (24)) and exchangeable (Equation (25)). □

A.4 Derivation of Markov Neural Process marginal densities

T step Markov Neural Processes (MNPs) have marginal densities in the following form (as in Equation (10)):

$$
\begin{aligned}
p_{x_{1:n}}(y_{1:n};\theta) &= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y^{(0)}_{1:n}) \prod_{t=1}^{T}\prod_{i=1}^{n} p_\theta^{(t)}(y_i^{(t)} \,|\, y_i^{(t-1)}, x_i, z^{(t)})\mathrm{d}y_{1:n}^{(0:T-1)}\mathrm{d}z^{(1:T)} \\
&= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y^{(0)}_{1:n}) \prod_{t=1}^{T}\prod_{i=1}^{n} \left|\det \frac{\partial F_\theta^{(t)}(y_i^{(t-1)}; x_i, z^{(t)})}{\partial y_i^{(t-1)}}\right| \mathrm{d}z^{(1:T)}
\end{aligned}
$$

Proof. Given the sequence of latent variables $z^{(1:n)}$, our model becomes normalising flows on the finite sequence of function outputs $y_{1:n}$, with a prior distribution $p_{x_{1:n}}(y^{(0)}_{1:n})$, and the invertible transformation at each step $F_\theta^{(t)}(\cdot; x_i, z^{(t)})$. According to Papamakarios et al. [43], Rezende and Mohamed [49], we have

$$
\begin{aligned}
p_{x_{1:n}}(y_{1:n}|z^{(1:T)};\theta) &= p_{x_{1:n}}(y^{(T)}_{1:n}|z^{(1:T)};\theta) \\
&= p_{x_{1:n}}(y^{(T-1)}_{1:n}|z^{(1:T-1)};\theta) \prod_{i=1}^{n} \left|\det \frac{\partial F_\theta^{(T)}(y_i^{(T-1)}; x_i, z^{(T)})}{\partial y_i^{(T-1)}}\right| \\
&= \cdots \\
&= p_{x_{1:n}}(y^{(0)}_{1:n}) \prod_{t=1}^{T}\prod_{i=1}^{n} \left|\det \frac{\partial F_\theta^{(t)}(y_i^{(t-1)}; x_i, z^{(t)})}{\partial y_i^{(t-1)}}\right|
\end{aligned}
$$

The marginal density $p_{x_{1:n}}(y_{1:n};\theta)$ can be computed as:

$$
\begin{aligned}
p_{x_{1:n}}(y_{1:n};\theta) &= \int p_\theta(z^{(1:T)}) p_{x_{1:n}}(y_{1:n}|z^{(1:T)};\theta)\mathrm{d}z^{(1:T)} \\
&= \int p_\theta(z^{(1:T)};\theta) p_{x_{1:n}}(y^{(0)}_{1:n}) \prod_{t=1}^{T}\prod_{i=1}^{n} \left|\det \frac{\partial F_\theta^{(t)}(y_i^{(t-1)}; x_i, z^{(t)})}{\partial y_i^{(t-1)}}\right| \mathrm{d}z^{(1:T)} \qquad (26)
\end{aligned}
$$

□

B Implementation details

B.1 Data

B.1.1 1D synthetic data

Gaussian Process (GP) samples Three 1D function datasets were created, each comprising samples from Gaussian processes (GPs) with different kernel functions: RBF (length scale 0.25), Matern-2.5 (length scale 0.5), and Exp-Sine-Squared (length scale 0.5 and periodicity 0.5). Observation noise

variances were set at 0.0001 for RBF and Matern kernels, and 0.001 for the Exp-Sine-Squared kernel. Function inputs spanned from -2.0 to 2.0. To accelerate sampling, identical input locations were employed for every 20 function instances. For this dataset, context size varies randomly from 2 to 50.

Monotonic functions Our generation of monotonic functions starts by sampling $N \sim \text{Poisson}(5.0)$ to determine the number of interpolation nodes. We then sample $N+1$ increments $X_{\text{increments}}$ sampled from a Dirichlet distribution. These increments are increased by 0.01 to avoid excessively small values, and are then normalized such that their sum is 4.0. The final X values for interpolation nodes are obtained by adding -2.0 to the cumulative sum of these increments so that these X values are within the range $[-2.0, 2.0]$. For each X value, a corresponding Y value is sampled from a Gamma distribution $Y \sim \text{Gamma}(2, 1)$. The cumulative sum of Y values ensures monotonicity. A PCHIP interpolator [20] is then created using these interpolation nodes (X and Y values) to generate function outputs. Given the functions, we randomly sample 128 X values and compute their corresponding function values. Note that these X values are now used to evaluate the functions, rather than serving as interpolation nodes. The function values are normalized to the range $[-1.0, 1.0]$. Finally, Gaussian observation noise with a standard deviation of 0.01 is added to these function values. For this dataset, context size varies randomly from 2 to 20.

Convex functions To create a dataset of convex functions, we compute integrals of the monotonic functions previously created. These convex functions are then randomly shifted and rescaled to increase diversity. The function values are normalized to the range $[-1.0, 1.0]$. Finally, Gaussian observation noise with a standard deviation of 0.01 is added to these function values. Context sizes varied randomly from 2 to 20.

Stochastic differential equations samples We create a dataset of 1D functions, each of which represents a solution to a Stochastic Differential Equation (SDE). This SDE is defined by the drift function $f(x, t) = -(a + x \cdot b^2) \cdot (1 - x^2)$ and the diffusion function $g(x, t) = b \cdot (1 - x^2)$, with constants a and b both set to 0.1. The function sets up a time span that includes 128 uniformly distributed points within the range of $[-5.0, 5.0]$. We then uniformly sample an initial condition, x_0, between 0.2 and 0.6. We use the `sdeint.stratKP2iS` function from the `sdeint` library to generate a solution to the SDE. This solution forms a 1D function that depicts a trajectory of the SDE across the defined time span, originating from the initial condition x_0. Lastly, we randomly alter the context sizes between 2 and 50.

For all the aforementioned datasets, we use the following set sizes: 50000 for the training set, 5000 for the validation set, and 5000 for the test set.

B.1.2 Geological data

We generate the GeoFluvial dataset using the `meanderpy` [54] package. We first run simulations using the numerical model of meandering in `meanderpy` with default parameters except for channel depth which we change from 16m to 6m. The resulting simulations correspond to images of shape (800, 4000). We then extract 3 random non-overlapping crops of shape (700, 700) from these images, which are resized to (128, 128) and are used as data. We ran $\lceil 25000/3 \rceil$ simulations, resulting in a dataset of 25,000 images.

B.2 Model architectures and hyperparameters

Permutation equivariant/invariant neural networks on sets We implement two versions of neural modules which operate on sets, both of which preserve the permutation symmetry of the sets. They are known as deep sets [61] and set transformers [36]. To obtain an invariant representation, we used sum pooling for deep sets, and pooling by multi-head attention (PMA) [36] for set transformers. Our experiments primarily employed set transformers, chosen for their stronger ability to model interactions between datapoints. However, for the wheel contextual bandit experiments, the context set often expanded to tens of thousands. To circumvent memory issues, we use deep sets in this instance. For set transformers, we stack two layers of set attention blocks (SABs) with a hidden dimension of 64 and 4 heads. This is followed by a single layer of PMA, which was subsequently followed by a linear map. In the case of using deep sets, we use a shared instance-wise Multi-Layer Perceptron (MLP) that has two hidden layers and a hidden dimension of 64. This MLP processes the concatenation of function inputs and outputs. Following this, we add a sum aggregation which is then succeeded by a single-layer MLP with ReLU (Rectified Linear Unit) activation.

Conditional normalising flows The instance-wise invertible transformation $F_\theta^{(t)}$ at each time step t is parameterised as a rational quadratic spline flow [16]. Note that we do not share parameters among iterations. To condition on an input x_i and a latent variable z, we use a Multi-Layer Perceptron (MLP) which takes in x_i, z and produces the parameters for configuring a one-layer spline flow with 10 bins.

Multi-Layer Perceptrons (MLPs) Except for the deep sets, we use MLPs to parameterise continuous functions in two places. Firstly, we use it as a conditioning component in conditional normalising flows $F_\theta^{(t)}$ (as we mentioned above). It has two hidden layers and a hidden dimension of 128. Secondly, we use it to parameterise $\mu_\phi^{(t)}$ and $\sigma_\phi^{(t)}$ in the inference model (see Section 4.3). It has two hidden layers and a hidden dimension of 64.

We adopt the same pretraining approach as Garnelo et al. [22] for the contextual bandits problem, pretraining the model on 64 wheel problems $\{\delta_i\}_{i=1}^{64}$, where $\delta_i \sim \mathcal{U}(0, 1)$. Each wheel δ_i has a context size ranging from 10 to 512 and a target size varying between 1 and 50. Here each data point is a tuple (X, a, r). Because the context size can grow to tens of thousands during test time, for computational efficiency, we use deep sets to implement SETENCODER and a linear flow rather than a spline flow to parameterise $F_\theta^{(t)}$.

For optimisation, we use Adam [31] optimiser with a learning rate of 0.0001. We use a batch size of 100 for 1D synthetic data and a batch size of 20 for the geological data. In experiments, we find that it is often beneficial to encode x_i with Fourier features [55], and we use 80 frequencies randomly sampled from a standard normal.

For more details, please also see our reference implementation at https://github.com/jinxu06/mnp.

B.3 Computational costs and resources

In our current implementation,the invertible transformations $F_\theta^{(t)}$ in the generative model and the mean/variance function $\mu_\phi^{(t)}\sigma_\phi^{(t)}$ in the inference model do not share parameters across iterations. However, we do share the SETENCODER across iterations. Consequently, with our MNP, both memory usage and computing time increase linearly with the number of steps. If the SETENCODER employs set transformers, the computational cost becomes $O(m^2)$, where m stands for the context size. This cost, however, can be decreased to $O(mk)$ by replacing set attention blocks (SABs) with induced set attention blocks (ISABs), where k denotes the number of inducing points. If deep sets are used to implement SETENCODER, the computational cost reduces to $O(m)$, despite the inference model being less expressive. The computatiional advantage of using deep sets becomes prominent when we have a large number of observed datapoints in the context. For instance, in the contextual bandits experiments (in Section 6.2), our model needs to process 80K datapoints, which is intractable for a transformer-based NP on a single GPU.

To further enhance scalability, one might consider sharing some parameters across transition steps. Additionally, in the case of space complexity, as discussed in Section 7, we can explore continuous-time Markov transitions, i.e., stochastic differential equations in function space. Such an approach would potentially require less memory by leveraging adjoint methods.

Training MNP is indeed resource-intensive; however, inference in MNP simply requires a forward pass. On a single GeForce GTX 1080 GPU card, a standard 7-step MNP takes approximately one day to train for $200k$ steps on 1D functions. If we use 20 latent samples to evaluate the marginal log-likelihood using the IWAE objective [6], inference runs typically in a few seconds for a batch of 100 functions.

C Ablation studies

The better performance of MNPs compared to standard Neural Processes (NPs) can roughly be attributed to two reasons: Firstly, instead of directly constructing the SP as in NPs, MNPs iteratively transform from a trivial SP gradually to a more expressive one. Secondly, MNPs employ more expressive invertible transformations on function outputs, parameterized by normalizing flows. To verify that multiple steps are indeed advantageous, we evaluated various models based on their estimated marginal log-likelihood on GP data using an RBF kernel. The results are as follows:

- MNP, 1 step, spline flow: 0.941
- MNP, 2 steps, spline flow: 1.950
- MNP, 3 steps, spline flow: 2.266
- MNP, 1 step, spline flow, latent dimension $\times 2$: 0.850
- MNP, 1 step, spline flow, latent dimension $\times 2$, MLP hidden dimension $\times 2$: 1.476
- MNP, 1 step, linear transformation: 0.990
- MNP, 2 steps, linear transformation: 1.977
- MNP, 3 steps, linear transformation: 2.140

From these results, we observe that performance generally improves with an increase in the number of steps. Using higher-dimensional latent variables or wider neural networks in models with only one step doesn't achieve the performance seen in models with multiple steps. Furthermore, replacing the spline flow with a linear transformation (keeping in mind that stacking multiple linear transformations remains linear) still showed a consistent performance improvement with an increase in steps.

D Broader impacts

Our work presents MNPs, a novel approach to construct SP using neural parameterised Markov transition operators. The broader impacts of this work have many aspects. Firstly, the proposed models are more flexible and expressive than traditional SP models and NPs, enabling them to handle more complex patterns that arise in many applications. Moreover, the exchangeability and consistency in MNPs could improve the robustness and reliability of neural SP models. This could lead to more trustworthy systems, which is a critical aspect in high-stakes applications. However, as with any machine learning system, there are potential risks. For example, the complexity of these models could exacerbate issues related to interpretability and transparency, making it more difficult for humans to understand and control their behaviour.