A. Proof

Theorem 1. *Given a set of observations* $\mathbf{x} = \{x_i\}_{i=1}^N$ *from an infinitely exchangeable process, denote the observed and unobserved part as* $\mathbf{x}_o = \{x_i^{(o_i)}\}_{i=1}^N$ *and* $\mathbf{x}_u = \{x_i^{(u_i)}\}_{i=1}^N$ *respectively. Then the arbitrary conditional distribution* $p(\mathbf{x}_u \mid \mathbf{x}_o)$ *can be decomposed as follows:*

$$p(\mathbf{x}_u \mid \mathbf{x}_o) = \int \prod_{i=1}^N p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) p(\theta \mid \mathbf{x}_o) d\theta.$$

Proof. According to the definition, we have $u_i, o_i \subseteq \{1, \dots, d\}$. Define $m_i = u_i \cup o_i$ and $\mathbf{x}_m = \mathbf{x}_u \cup \mathbf{x}_o = \{x_i^{(m_i)}\}_{i=1}^N$, where $\mathbf{x}_m$ represents the set of features that contains both observed and unobserved dimensions.

From the De Finetti's theorem, we can derive the following equations:

$$\begin{aligned}
p(\mathbf{x}_m) &= \int \prod_{i=1}^N p(x_i^{(m_i)} \mid \theta) p(\theta) d\theta \\
&= \int \prod_{i=1}^N \left[p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) p(x_i^{(o_i)} \mid \theta) \right] p(\theta) d\theta \\
&= \int \prod_{i=1}^N p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) \prod_{i=1}^N p(x_i^{(o_i)} \mid \theta) p(\theta) d\theta \\
&= \int \prod_{i=1}^N p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) p(\mathbf{x}_o \mid \theta) p(\theta) d\theta \\
&= \int \prod_{i=1}^N p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) p(\theta \mid \mathbf{x}_o) p(\mathbf{x}_o) d\theta
\end{aligned}$$

The key step is the penultimate equation, where the De Finetti's theorem is applied for $\mathbf{x}_o$ in reverse direction. That is, given the same latent code, we assume set elements of $\mathbf{x}_o$ are conditionally i.i.d.. Since $\mathbf{x}_o$ contains subsets of features from $\mathbf{x}_m$, the above assumption holds.

Dividing both side of the equation by $p(\mathbf{x}_o)$ gives

$$p(\mathbf{x}_u \mid \mathbf{x}_o) = \int \prod_{i=1}^N p(x_i^{(u_i)} \mid x_i^{(o_i)}, \theta) p(\theta \mid \mathbf{x}_o) d\theta$$

□

B. Models

As shown in Fig. 1, our model mainly contains 4 parts: posterior, prior, permutation equivariant embedding and the generator. The prior can be further divided as base distribution (B) and the flow transformations (Q). Here, we describe the specific architectures used for each component respectively. Please see Table B.1 for details. Note we did not tune the network architecture heavily. Further improvement is expected by tuning the network for each task separately.

For simplicity, the posterior and prior are mostly constructed by processing each set element independently then pooling across the set. A Gaussian distribution is then derived from the permutation invariant embedding. For point clouds, we use set transformer to incorporate dependencies among points. For set of functions (each function is represented as a set of (input, target) pairs), we first use set transformer to get an permutation invariant embedding for each function. Then, we take the average pooling over the set as the embedding for the set. The prior utilizes a normalizing flow to increase the flexibility, which is implemented as a stack of invertible transformations over the samples. For images, we use a multi-scale ACFlow as the generator, which is similar to the original model used in (Li et al., 2020a). We modify it to a conditional version so that both the base distribution and the transformations are conditioned on the given embeddings. For point cloud, we use the conditional ACFlow with an autoregressive base likelihood.

C. Experiments

C.1. Image Inpainting

For image inpainting, we evaluate on both MNIST and Omniglot datasets. We construct the sets by randomly selecting 10 images from a certain class. The observed part for each image is a 10×10 square placed at random positions.

The baseline TRC (Wang et al., 2017) is based on tensor ring completion. We use their official implementation [1] and cross validate the hyperparameters on our datasets. The set of images with size $[32, 32, 10]$ are treated as a 3-order tensor and reshaped into a 10-order tensor of size $[4, 2, 2, 2, 4, 2, 2, 2, 2, 5]$. The tensor is then completed by alternating least square method with a specified tensor ring rank (TR-Rank). We found the TR-Rank of 9 works best for our datasets.

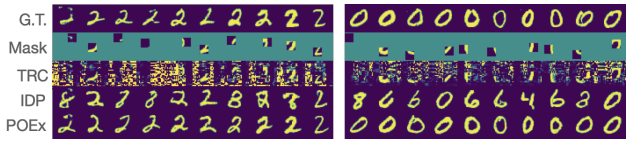
Figure C.1 presents several additional examples for inpainting a set of images. We can see the TRC fails to recover any meaningful structures, IDP fails to infer the right classes, while our POEx model always generates the characters from the specified classes.

At the request of the reviewers, we also provide a comparison to Neural Process. We would like to emphasize that our experimental setting is different from NP. NP models each image independently, while POEx models a set of images jointly. Conceptually NP is similar to the IDP baseline, but IDP learns a normalizing flow-based generative model for $p(x_i^{(u_i)} \mid x_i^{(o_i)})$, which performs better than NP based on our past experience. Figure C.2 shows several imputation

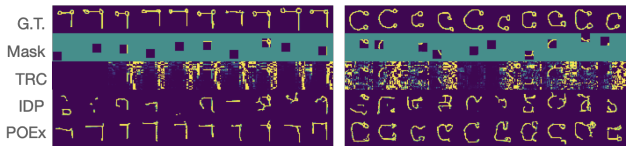
[1] https://github.com/wangwenqi1990/TensorRingCompletion

Table B.1. Network architectures

Data Type	Components	Architecture
Set of Images	Posterior	[Conv(128,3,1), Conv(128,3,1), MaxP(2,2)]×4 + FC(256) + SetAvgPool → Gaussian(128)
	Prior (B)	[Conv(128,3,1), Conv(128,3,1), MaxP(2,2)]×4 + FC(256) + SetAvgPool → Gaussian(128)
	Prior (Q)	[Linear, LeakyReLU, Affine Coupling, Permute]×4
	Peq Embed	[Conv(64,3,1), Conv(64,3,1), MaxP(2,2)]×2 + [DecomAttn, ResBlock]×4 + [DeConv(64,3,2), Conv(64,3,1)]×2
	Generator	Conditional ACFlow(multi-scale)
Point Clouds	Posterior	[SetTransformer(256)]×4 + SetAvgPool + FC(512) → Gaussian(256)
	Prior (B)	[SetTransformer(256)]×4 + SetAvgPool + FC(512) → Gaussian(256)
	Prior (Q)	[Linear, LeakyReLU, Affine Coupling, Permute]×4
	Peq Embed	[SetTransformer(256)]×4
	Generator	Conditional ACFlow
Set of Functions	Posterior	[SetTransformer(256)]×4 + SetAvgPool + SetAvgPool + FC(512) → Gaussian(256)
	Prior (B)	[SetTransformer(256)]×4 + SetAvgPool + SetAvgPool + FC(512) → Gaussian(256)
	Prior (Q)	[Linear, LeakyReLU, Affine Coupling, Permute]×4
	Peq Embed	SelfAttention + CrossAttention
	Generator	[FC(256)]×4 + FC(2) → Gaussian(1)



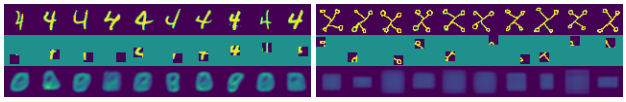
(a) MNIST



(b) Omniglot

Figure C.1. Inpaint the missing values for a set of images.

results from convolutional NP (Gordon et al., 2020), which is trained with the recommended hyperparameter in the official implementation. We can see it has difficulty accurately recovering the underlying classes given the limited context.



(a) MNIST (b) Omniglot

Figure C.2. Imputation results from Convolutional NP.

C.2. Image Set Expansion

To expand a set, we modify the distribution of the masks so that some elements are fully observed and others are fully unobserved. We randomly select the number of observed elements during training. During test, our model can expand the set to arbitrary size.

C.3. Few-shot Learning

Since our model can expand sets for even unseen categories, we utilize it to augment the support set for few-shot learning. Given a N-way-K-shot training set, we use the K exemplars from each class to generate M novel instances, thus change the problem to N-way-(M+K)-shot classification. We train the MAML with fully connected networks using the official implementation [2].

C.4. Point Cloud Completion

For point cloud completion, we build the dataset by sampling 256 points from the observed part and 1792 points from the occluded part, thus there are 2048 points in total. PCN is trained with a multi-scale architecture, where the given point cloud is first expanded to 512 points and then expanded further to 2048 points. We use EMD loss and CD loss for the coarse and fine outputs respectively.

C.5. Point Cloud Upsampling

Point cloud upsampling uses the ModelNet40 dataset. We leave 10 categories out that are not used during training to evaluate the generalization ability of our model. We uniformaly sample 2048 points as the target. During training, an arbitrary subset is taken as input. PUNet is not built for upsampling arbitrary sized point cloud, therefore we subsample 256 points as input. PUNet is trained to optimize the EMD loss and a repulsion loss as in their original work. For comparison, we evaluate our POEx model and PUNet for upsampling 256 points.

C.6. Point Cloud Compression

One advantage of our POEx model is that the likelihood can be used to guide the subset selection. To evaluate the selection quality, we build a biased point cloud by sampling the center points with higher probability. Specifically, 2048 points are selected from the original point cloud with probability $\text{softmax}(\|x-m\|^2/T^2)$, where x and m are the coordinates of each candidate points and the center respectively. We use $T = 0.1$ for our experiments. Given 2048 non-uniformly sampled points, our goal is to sample 256

[2] https://github.com/cbfinn/maml

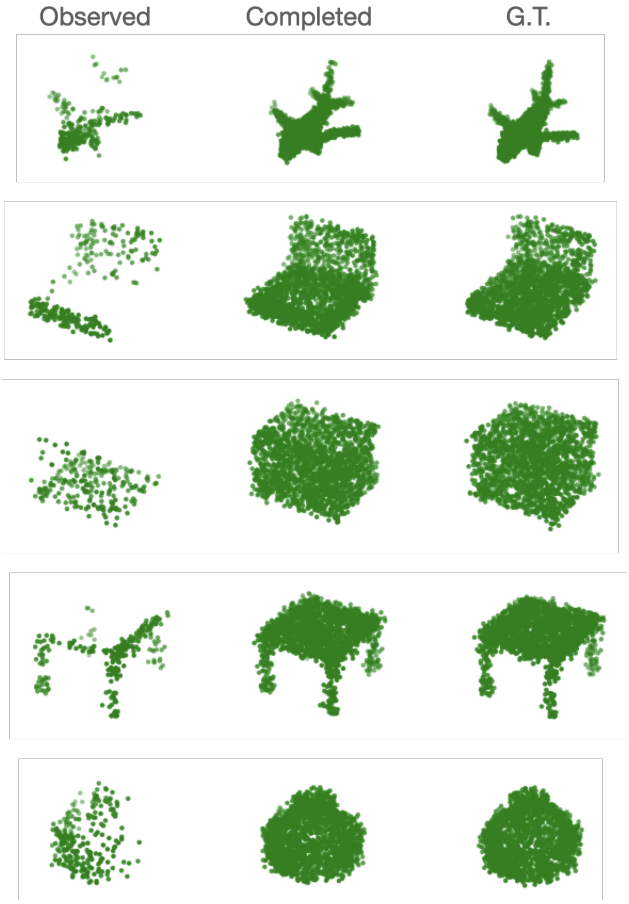


Figure C.3. Additional examples for point cloud completion.

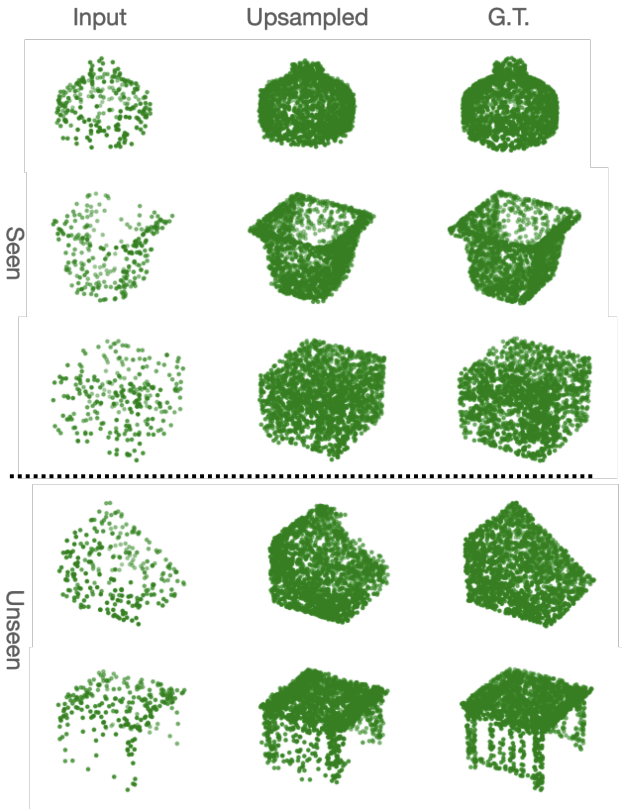


Figure C.4. Additional examples for point cloud upsampling.

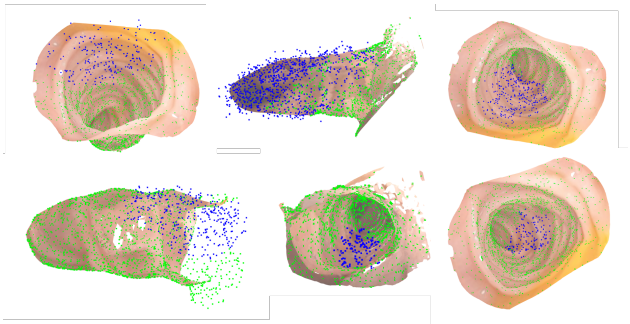


Figure C.5. Additional examples for imputing colonoscopy point cloud.

points from it to represent the underlying geometry. We propose a sequential selection strategy, where one point is selected per step based on its conditional likelihood. The one with lowest likelihood given the current selected points is selected at each step. Baseline approaches include uniform sampling, where 256 points are uniformly sampled. A k-means based sampling method group the given points into 256 clusters and we select one point from each cluster that is closest to its cluster center. The farthest point sampling algorithm also proceeds sequentially, where the point that is farthest from the current selected points is selected at each step. To quantitatively evaluate the selection quality we reconstruct the selected subset to 2048 points and compare it to a uniformly sampled point cloud. If the selected subset represents the geometry well, the upsampled point cloud should be close to the uniformly sampled one.

C.7. Colonoscopy Point Cloud Imputation

We uniformly sample 2048 points from the reconstructed colonoscopy meshes as our training data. During training, points inside a random ball are viewed as unobserved blind spot and our model is trained to impute those blind spots. To provide guidance for the imputation, we divide the space into small cubes and condition our model on the cube coordinates. That is, points inside the cube are indexed by the corresponding cube coordinates. To train the conditional POEx model, the indexes and the point coordinates are concatenated together as inputs. Figure C.5 presents several examples of the imputed point clouds.

C.8. Neural Processes over Images

The conditional version of POEx can be interpreted as a neural process. We evaluate our model on the ShapeNet dataset, where images of a object viewed from different angles are used as the target variables. Given the context that contains images from several random views, the neural processes are expected to generate novel images for arbitrary view points. We split the dataset into seen and unseen

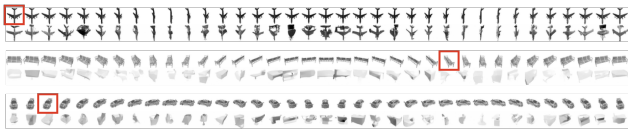


Figure C.6. Neural Process sampling from conditional BRUNO. Red coxes indicate the given context. First row: ground truth. Second row: generation.

categories and train our POEx model and the conditional BRUNO only on seen categories. Images are indexed with the continuous angles, which we translate to their sin and cos values. Figure C.6 shows several examples of generated images from conditional BRUNO. We can see the generated images do not always match with the provided context.

C.9. Video Inpainting

We evaluate our model for video inpainting with two datasets. 10 frames are randomly selected from a video to construct a set. The frames are indexed by their timestamps. We normalize them to the range of [0,1] and further calculate a 32-dimensional positional embedding similar to (Vaswani et al., 2017). To simulate occluded pixels, we put a 16×16 square at random position, pixels inside the square are considered missing. For occlusion removal, since we do not have the ground truth values for occluded pixels, those pixels are excluded for training and evaluation. We run TCC (Huang et al., 2016) using the official code [3] and their recommended hyperparameters. For group mean imputation, if a certain pixel is missing in all frames, we impute it using the global mean of all observed pixels. Figure C.7 and C.8 present additional examples for these two datasets. GMI works well if the movement is negligible, but it fails when the object moves too much. It also struggles if certain pixels are missing in all frames. In the second example of Fig. C.7, TCC fails to run since the missing rate is too high for certain frames and the optical flow cannot be properly estimated. For quantitative evaluation, we report the PSNR and SSIM scores between the ground truth and the imputed images. We calculate PSNR only for the missing part and the SSIM for the whole image.

C.10. Set of Functions

Generalizing the concept of partial sets to infinite dimensional set elements, we can utilize POEx to model a set of functions. Each function is represented as a set of (input, target) pairs. We first evaluate on the multi-task Gaussian processes. Following (Bonilla et al., 2008), we directly

[3] https://github.com/amjltc295/Temporally-Coherent-Completion-of-Dynamic-Video

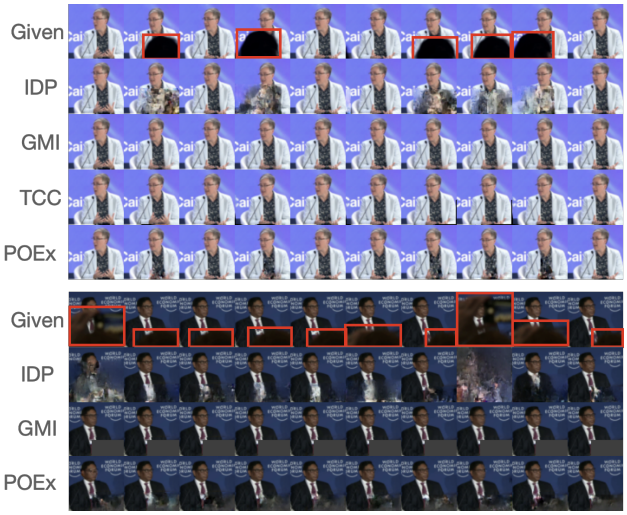


Figure C.7. Additional examples for occlusion removal.

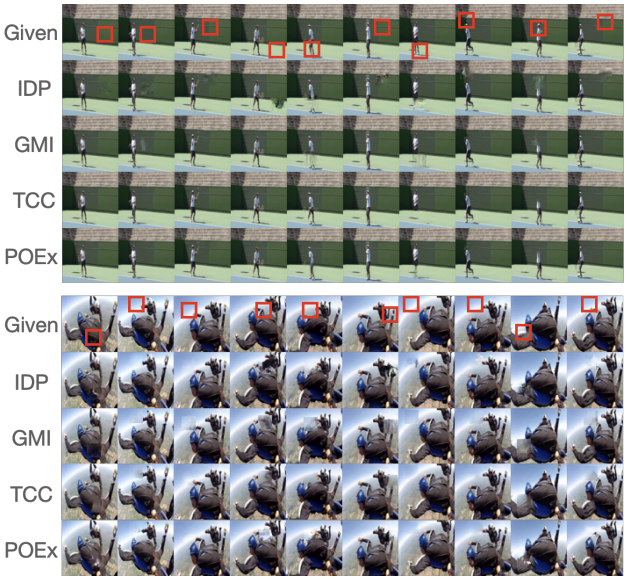


Figure C.8. Additional examples for inpainting Youtube videos.

specify the correlations among functions:

$$\langle f_l(x), f_k(x')\rangle = K^f_{lk} k^x(x, x'), \quad y_{il} = \mathcal{N}(f_l(x_i), \sigma_l^2),$$

where K^f is a positive semi-definitive matrix that specifies the inter-task similarities, k^x is a covariance function over inputs. To model N functions, K^f is a $N \times N$ matrix and K^f_{lk} is the element in row l and column k. Here, we assume the tasks are permutation equivariant, that is, every two tasks have the same correlation:

$$K^f_{lk} = \begin{cases} c, l \neq k \\ 1, l = k \end{cases}$$

Similar to Neural processes, we use a Gaussian kernel for k^x. During training, we generate synthetic data from 5 functions with $c = 0.9$. We sample at most 100 points from each function and select at most 10 points as context. The sampled points are then transformed by shifting or reversing to obtain 5 target functions. Our POEx model is trained by conditioning on a one-hot function identifier.

Following Neural Processes, we also build a set of functions from a set of images. Given a set of MNIST images from the same class, we view each image as a function between the pixel index and its value. During training, we sample an arbitrary subset from each image as context.