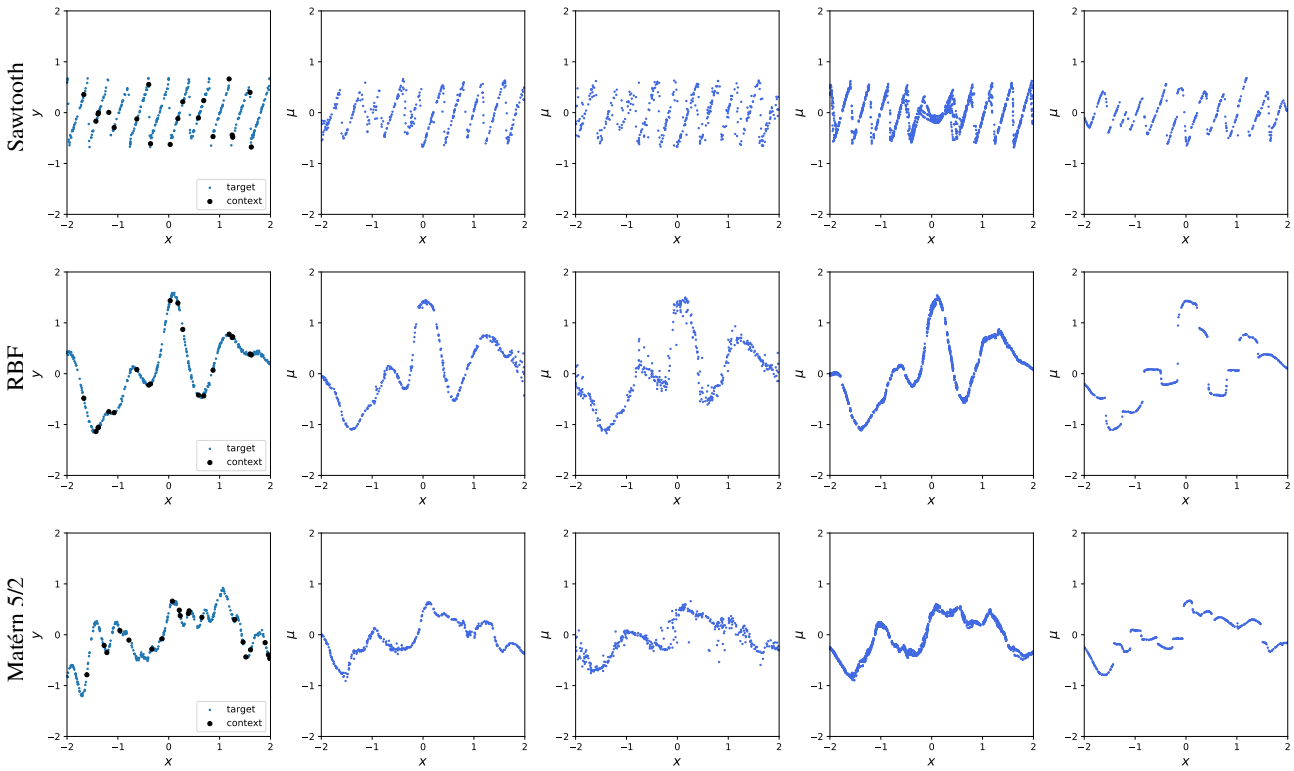
Adaptive Conditional Quantile Neural Processes (Supplementary material)

Peiman Mohseni[1] **Nick Duffield**[2] **Bani Mallick**[3] **Arman Hasanzadeh**[4]

[1]Computer Science and Engineering Department, Texas A&M University
[2]Electrical and Computer Engineering Department, Texas A&M University
[3]Statistics Department, Texas A&M University
[4]Google Cloud

1 ADDITIONAL RESULTS



a) Data b) ACQNP c) CQNP d) BNP e) CANP

Figure 1: Examples of predictions made by different methods. For A/CQNP, the mean of the compound predictive distribution, approximated with $N_\tau = 10$ samples, is plotted. For BNP and CANP, we plot the mean of the Gaussian predictive distribution as the predictions. For BNP, we plot the predictions obtained from 20 different sets of bootstrap contexts.

We compare A/CQNPs and baselines on data generated from three additional processes described in table 1 with the following choice of parameters:

- Sawtooth[Gordon et al., 2020]: $s \sim \mathcal{U}[-2,\ 2)$, $\alpha \sim \mathcal{U}[1, 2)$, $\omega \sim [1, 3)$, $\delta \sim \mathcal{U}[-2, 2)$, $K \sim \mathcal{U}[10, 20)$
- RBF: $s \sim \mathcal{U}[-2,\ 2)$, $\ell = 0.25$, $\sigma = 0.75$, $\delta = 0.02$

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

- Matérn 5/2: $s \sim \mathcal{U}[-2,\, 2)$, $\ell = 0.25$, $\sigma = 0.75$, $\delta = 0.02$

The results are provided in table 2. Figure 1 illustrates examples of predictions made by each method. Note that for A/CQNP, the predictions correspond to the mean of the conditional distribution $p(y \mid x)$, not the quantiles. The mean of the uncountable mixture of $\mathcal{A}L$ distributions can be computed as the following:

$$\begin{aligned}
\mathbb{E}_{\mathrm{y}}[p(\mathrm{y} \mid x)] &= \mathbb{E}_{\mathrm{y}}\left[\mathbb{E}_{\tau \sim \mathcal{U}(0,1)}\left[\alpha_\tau(x)\, \mathcal{A}L\left(\mathrm{y} \mid \mu_\tau(x), \sigma_\tau(x), \tau\right)\right]\right] \\
&= \mathbb{E}_{\tau \sim \mathcal{U}(0,1)}\left[\mathbb{E}_{\mathrm{y}}\left[\alpha_\tau(x)\, \mathcal{A}L\left(\mathrm{y} \mid \mu_\tau(x), \sigma_\tau(x), \tau\right)\right]\right] \\
&= \mathbb{E}_{\tau \sim \mathcal{U}(0,1)}\left[\alpha_\tau(x)\, \mathbb{E}_{\mathrm{y}}\left[\mathcal{A}L\left(\mathrm{y} \mid \mu_\tau(x), \sigma_\tau(x), \tau\right)\right]\right] \\
&= \mathbb{E}_{\tau \sim \mathcal{U}(0,1)}\left[\alpha_\tau(x)\left(\mu_\tau(x) + \frac{1-2\tau}{\tau(1-\tau)}\sigma_\tau(x)\right)\right].
\end{aligned} \tag{1}$$

Similar to section 3.2, we use Monte Carlo to approximate this expectation. For sawtooth, RBF, and Matérn 5/2, we used 100, 50, and 50 maximum context points, respectively.

Table 1: Synthetic processes used in unimodal 1D regression experiments.

Process	$g(s) = (g_x(s),\, g_y(s))$
Sawtooth	$g_x(s) = s\,,\; g_y(s) = \frac{\alpha}{2} - \frac{\alpha}{\pi}\sum_{k=1}^{K}(-1)^k \frac{\sin(2\pi k\omega(s+\delta))}{k}$
GP (RBF)	$g_x(s) = s\,,\; g_y \sim \mathcal{GP}(0, C)\,,\; C(x,x') = \sigma^2 \exp\left(-\frac{\|x-x'\|^2}{2\ell}\right) + \delta$
GP (Matérn 5/2)	$g_x(s) = s\,,\; g_y \sim \mathcal{GP}(0, C)\,,\; C(x,x') = \sigma^2(1 + \frac{\sqrt{5}d}{\ell} + \frac{5d^2}{3\ell^2}) + \delta, d = \|x-x'\|$

Table 2: Context and target log-likelihoods on synthetic 1D regression tasks (6 Seeds).

	Sawtooth		RBF		Matérn 5/2	
	context	target	context	target	context	target
CNP	$0.937_{\pm 0.023}$	$0.586_{\pm 0.038}$	$0.837_{\pm 0.058}$	$0.100_{\pm 0.023}$	$0.626_{\pm 0.056}$	$-0.183_{\pm 0.013}$
CANP	$1.191_{\pm 0.190}$	$0.341_{\pm 0.085}$	$\mathbf{1.269}_{\pm 0.083}$	$0.225_{\pm 0.052}$	$\mathbf{1.058}_{\pm 0.382}$	$\mathbf{-0.015}_{\pm 0.198}$
BNP	$0.884_{\pm 0.038}$	$0.769_{\pm 0.039}$	$1.121_{\pm 0.008}$	$\mathbf{0.339}_{\pm 0.009}$	$0.879_{\pm 0.018}$	$\mathbf{-0.048}_{\pm 0.018}$
CQNP(ours)	$\mathbf{1.229}_{\pm 0.031}$	$\mathbf{0.833}_{\pm 0.035}$	$0.947_{\pm 0.042}$	$0.083_{\pm 0.037}$	$0.515_{\pm 0.039}$	$-0.373_{\pm 0.041}$
ACQNP(ours)	$\mathbf{1.386}_{\pm 0.042}$	$\mathbf{1.026}_{\pm 0.039}$	$\mathbf{1.215}_{\pm 0.027}$	$\mathbf{0.254}_{\pm 0.020}$	$\mathbf{0.912}_{\pm 0.042}$	$-0.117_{\pm 0.025}$

2 IMPLEMENTATION DETAILS

This section provides a detailed description of different methods' implementation mentioned throughout the paper. All the implementations are based on PyTorch [Paszke et al., 2019]. For CNP and CANP, we closely followed the official implementation[1]. Note that the implementation of CANP is identical to ANP, but without the latent path shown in figure 2 of Kim et al. [2019]. For BNP, however, we borrowed the implementation provided by the authors [2], and thus use their terminology in describing the network architecture. Nonetheless, neural networks used in all models are instances of multi-layer perceptrons (MLPs) with ReLU activations and the only differences are regarding their depth and width. To specify the architecture of MLPs, we use the following notation:

$$[d_{h_0}] \times [d_{h_1}, \ldots, d_{h_{n-1}}] \times [d_{h_n}],$$

where d_{h_0} and d_{h_n} denote the dimension of the network's input and output respectively. Furthermore, $[d_{h_1}, \ldots, d_{h_{n-1}}]$ shows that the network has $n-1$ hidden layers with d_{h_i} as the width of i-th hidden layer. In all the experiments, Adam [Kingma and Ba, 2015] is used for optimizing the objective function. Other than the learning rate and the ℓ_2 regularizer, rest of the hyper-parameters used with Adam are set to the default values in Pytorch.

[1] https://github.com/deepmind/neural-processes

[2] https://github.com/juho-lee/bnp

2.1 SYNTHETIC DATA

For synthetic data, each model is trained for 10^5 iterations with 128 sampled functions per batch. During training, the tasks are generated at the moment, i.e. the training data is not fixed across different models and seeds. However, for evaluation, we generate and save 5×10^3 batches, each containing 16 curves. This data is later used to evaluate all the models. Note that in our implementation, N_{total} is the same across all observations in each batch. More precisely, for a batch $\mathcal{E} = \{\mathcal{E}_k\}_{k=1}^{n_b}$ with n_b as the batch size, all $\mathcal{E}_k$s contain the same number of data points. However, N_{total} is not necessarily the same between two different batches as explained in section 4.1. The same setup holds for N_{context}.

2.1.1 CNP

Table 3 shows the encoder and decoder architectures used in the implementation of CNPs for experiments on synthetic data. Table 4 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

Table 3: Architectural details of CNPs for experiments on synthetic data.

Benchmark	Encoder	Decoder
Sawtooth RBF Matérn 5/2 Double Sine Circle Lissajous	$[2] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128] \times [2]$

Table 4: Hyper-parameters and GPU devices used for training and testing CNPs on synthetic data.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
Sawtooth	5×10^{-4}	0	Quadro RTX 6000	NVIDIA A100
RBF	5×10^{-4}	0	NVIDIA A100	Quadro RTX 6000
Matérn 5/2	5×10^{-4}	0	NVIDIA A100	Quadro RTX 6000
Double Sine	5×10^{-4}	0	NVIDIA A100	NVIDIA A100
Circle	5×10^{-4}	10^{-5}	Quadro RTX 6000	NVIDIA A100
Lissajous	5×10^{-4}	0	Quadro RTX 6000	NVIDIA A100

2.1.2 CANP

Table 5 contains details on the MLP architectures used for modeling the encoder and decoder modules in CANPs. Note that instead of passing raw context and target inputs as keys and queries to the attention modules, we first pass them through separate MLPs, namely key encoder and query encoder, and then apply the attention to the obtained embeddings. Here we work with the same 8-headed attention [Vaswani et al., 2017] mechanism used in the official implementation. Table 6 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

Table 5: Architectural details of CANPs for experiments on synthetic data.

Benchmark	Context Encoder	Key Encoder	Query Encoder	Decoder
Sawtooth RBF Matérn 5/2 Double Sine Circle Lissajous	$[2] \times [128, 128, 128] \times [128]$	$[1] \times [128] \times [128]$	$[1] \times [128] \times [128]$	$[129] \times [128, 128, 128] \times [2]$

Table 6: Hyper-parameters and GPU devices used for training and testing CANPs on synthetic data.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
Sawtooth	10^{-4}	0	Quadro RTX 6000	NVIDIA A100
RBF	10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
Matérn 5/2	10^{-4}	0	Tesla T4	NVIDIA A100
Double Sine	10^{-4}	10^{-5}	NVIDIA A100	NVIDIA A100
Circle	10^{-4}	10^{-5}	Quadro RTX 6000	NVIDIA A100
Lissajous	10^{-4}	10^{-5}	Quadro RTX 6000	NVIDIA A100

2.1.3 BNP

The architecture details for different components of BNPs including the encoder, adaptation layer, and decoder are provided in table 7. For all the benchmarks, we use $k = 4$ and $k = 50$ bootstrap contexts for training and testing, respectively. The choice of optimization hyperparameters along with the GPU devices used for training and testing are included in table 8.

Table 7: Architectural details of BNPs for experiments on synthetic data.

Benchmark	d_x	d_y	d_h	l_{pre}	l_{post}	l_{dec}
Sawtooth RBF Matérn 5/2 Double Sine Circle Lissajous	1	1	128	5	3	5

Table 8: Hyper-parameters and GPU devices used for training and testing BNPs on synthetic data.

Benchmark	Learning rate	L2 regularizer	Scheduler	GPU (Training)	GPU (Testing)
Sawtooth	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000
RBF	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000
Matérn 5/2	5×10^{-4}	0	cosine annealing	Tesla T4	NVIDIA A100
Double Sine	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000
Circle	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000
Lissajous	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000

2.1.4 CQNP

The encoder and decoder architectures used for implementing CQNPs in different benchmarks are shown in table 9. Table 10 summarizes the choice of hyperparameters along with the GPU devices used for training and testing.

Table 9: Architectural details of CQNPs for experiments on synthetic data.

Benchmark	Context Encoder	Decoder
Sawtooth RBF Matérn 5/2 Double Sine Circle Lissajous	$[2] \times [128, 128, 128] \times [128]$	$[130] \times [128, 128, 128] \times [3]$

Table 10: Hyper-parameters and GPU devices used for training and testing CQNPs on synthetic data.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
Sawtooth	5×10^{-4}	10^{-5}	50	100	Quadro RTX 6000	NVIDIA A100
RBF	10^{-3}	10^{-5}	50	100	Quadro RTX 6000	Quadro RTX 6000
Matérn 5/2	5×10^{-3}	10^{-5}	50	100	Tesla T4	Tesla T4
Double Sine	10^{-3}	10^{-5}	50	100	NVIDIA A100	Quadro RTX 6000
Circle	10^{-3}	0	50	100	NVIDIA A100	NVIDIA A100
Lissajous	10^{-3}	10^{-5}	50	100	NVIDIA A100	NVIDIA A100

2.1.5 ACQNP

Compared to CQNP, ACQNP has an additional component named the adaptation layer which takes in the raw sample u together with context representation and target input and maps them to a new set of quantile levels τ that we eventually approximate. Note that this is different from the adaptation layer used in BNPs. Also, we apply a sigmoid function to the final outputs of the adaptation layer to make sure that they correspond to valid quantile levels. The depth and width of the MLPs used for parameterizing the encoder, adaptation layer, and decoder in ACQNPs are presented in table 11. We summarize the choice of hyperparameters along with the GPU models used for training and testing in table 12.

Table 11: Architectural details of ACQNPs for experiments on synthetic data.

Benchmark	Context Encoder	Adaptor	Decoder
Sawtooth	$[2] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [3]$
RBF	$[2] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [3]$
Matérn 5/2	$[2] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [3]$
Double Sine Circle Lissajous	$[2] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [3]$

Table 12: Hyper-parameters and GPU devices used for training and testing ACQNPs on synthetic data.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
Sawtooth	5×10^{-4}	0	50	100	Quadro RTX 6000	NVIDIA A100
RBF	10^{-3}	0	50	100	NVIDIA A100	Quadro RTX 6000
Matérn 5/2	10^{-3}	10^{-5}	50	100	Tesla T4	NVIDIA A100
Double Sine	10^{-3}	10^{-5}	50	100	NVIDIA A100	NVIDIA A100
Circle	10^{-3}	10^{-5}	50	100	Quadro RTX 6000	NVIDIA A100
Lissajous	10^{-3}	10^{-5}	50	100	NVIDIA A100	NVIDIA A100

2.2 SPEED-FLOW

For the speed-flow data, 75% of each lane's observations (≈ 988) are randomly selected for training and the rest are held out for testing. The batch size for both training and testing is 2 since we have data from 2 lanes. For the final evaluation of each method, we take the context and target sets to be the training and testing data, respectively. For training, we generate and save 10^4 copies of the training data with random partitioning to context and target sets. This means that we fix the training curves as well as the evaluation data across all models as the dataset is quite small and does not require a lot of memory for storage.

2.2.1 CNP

Table 13 shows the encoder and decoder architectures used for implementing CNPs in different benchmarks. Table 14 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

Table 13: Architectural details of CNPs for experiments on speed-flow data.

Benchmark	Encoder	Decoder
Speed-Flow	$[2] \times [64, 64] \times [64]$	$[65] \times [64, 64] \times [2]$

Table 14: Hyper-parameters and GPU devices used for training and testing CNPs on speed-flow data.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
Speed-Flow	10^{-4}	10^{-5}	Tesla T4	Tesla T4

2.2.2 CANP

Table 15 contains details on the MLP architectures used for modeling the encoder and decoder modules in CANPs. Note that instead of passing raw context and target inputs as keys and queries to the attention modules, we first pass them through separate MLPs, namely the key encoder and query encoder, and then apply the attention mechanism to the obtained embedding. Here we work with the same 8-headed attention mechanism used in the official implementation. Table 16 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

Table 15: Architectural details of CANPs for experiments on speed-flow data.

Benchmark	Context Encoder	Key Encoder	Query Encoder	Decoder
Speed-Flow	$[2] \times [64, 64] \times [64]$	$[1] \times [64] \times [64]$	$[1] \times [64] \times [64]$	$[65] \times [64, 64] \times [2]$

Table 16: Hyper-parameters and GPU devices used for training and testing CANPs on speed-flow data.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
Speed-Flow	10^{-4}	10^{-5}	Tesla T4	Tesla T4

2.2.3 BNP

The architecture details for different components of the BNPs including the encoder, adaptation layer, and decoder are provided in table 17. For all the benchmarks, we use $k = 4$ and $k = 50$ bootstrap contexts for training and testing, respectively. The choice of optimization hyperparameters along with the GPU devices used for training and testing are included in table 18.

Table 17: Architectural details of BNPs for experiments on speed-flow data.

Benchmark	d_x	d_y	d_h	l_{pre}	l_{post}	l_{dec}
Speed-Flow	1	1	64	4	3	4

Table 18: Hyper-parameters and GPU devices used for training and testing BNPs on speed-flow data.

Benchmark	Learning rate	L2 regularizer	Scheduler	GPU (Training)	GPU (Testing)
Speed-Flow	5×10^{-4}	10^{-5}	None	Tesla T4	Tesla T4

2.2.4 CQNP

The encoder and decoder architectures used for implementing CQNPs in different benchmarks are shown in table 19. Table 20 summarizes the choice of hyperparameters along with the GPU devices used for training and testing.

Table 19: Architectural details of CQNPs for experiments on speed-flow data.

Benchmark	Context Encoder	Decoder
Speed-Flow	$[2] \times [64, 64] \times [64]$	$[66] \times [64, 64] \times [3]$

Table 20: Hyper-parameters and GPU devices used for training and testing CQNPs on speed-flow data.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
Speed-Flow	5×10^{-3}	10^{-5}	100	50	Tesla T4	Quadro RTX 6000

2.2.5 ACQNP

Compared to CQNP, ACQNP has an additional component named the adaptation layer which takes in the raw sample of u together with context representation and target inputs and maps them to a new set of quantile levels τ that we eventually approximate. Note that this is different from the adaptation layer used in BNPs. Also, we apply a sigmoid function to the outputs of the adaptation layer to make sure that they correspond to valid quantile levels. The depth and width of the MLPs used for parameterizing the encoder, adaptation layer, and decoder in ACQNPs are presented in table 21. We summarize the choice of hyperparameters along with the GPU models used in training and testing in table 22.

Table 21: Architectural details of ACQNPs for experiments on speed-flow data.

Benchmark	Context Encoder	Adaptor	Decoder
Speed-Flow	$[2] \times [64, 64] \times [64]$	$[65] \times [64, 64] \times [1]$	$[66] \times [64, 64] \times [3]$

Table 22: Hyper-parameters and GPU devices used for training and testing ACQNPs on speed-flow data.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
Speed-Flow	5×10^{-3}	10^{-5}	100	50	Tesla T4	Quadro RTX 6000

2.3 IMAGE COMPLETION

In image completion experiments on MNIST, Fashion-MNIST, SVHN, and Omniglot, we use the default train/test split of the data. For FreyFace, however, we randomly select 75% of the images for training and keep the rest for testing. Similar to the experiments on synthetic data, the partitioning of image pixels to context and target sets is done randomly and during training, i.e. context and target sets are not fixed across different models and seeds. For evaluation, however, we saved the generated batches from test images. In the case of FreyFace, we repeat this process 4 more times so that each test image has 5 copies with different context/target splits in the stored evaluation batches. Note that the number of context points across different tasks in a batch is the same, but might change from one batch to another. Obviously, the union of context and target sets which comprises all pixels of an image is the same in all cases as the image size is fixed in each dataset. All the models were trained for 100 epochs with 16 images per batch. The same batch size is used for testing.

2.3.1 CNP

Table 23 shows the encoder and decoder architectures used for implementing CNPs in different benchmarks. Table 24 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

2.3.2 CANP

Table 25 contains details on the MLP architectures used for modeling the encoder and decoder modules in CANPs. Note that instead of passing raw context and target inputs as keys and queries to the attention modules, we first pass them through separate MLPs, namely the key encoder and query encoder, and then apply attention to the obtained embedding. Here we

Table 23: Architectural details of CNPs for image completion tasks.

Benchmark	Encoder	Decoder
MNIST Fashion MNIST Omniglot FreyFace	$[3] \times [128, 128, 128] \times [128]$	$[130] \times [128, 128, 128] \times [2]$
SVHN	$[5] \times [128, 128, 128] \times [128]$	$[130] \times [128, 128, 128] \times [6]$

Table 24: Hyper-parameters and GPU devices used for training and testing CNPs on image completion tasks.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
MNIST	5×10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
Fashion MNIST	5×10^{-4}	0	Tesla T4	Quadro RTX 6000
Omniglot	5×10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
FreyFace	5×10^{-4}	0	Quadro RTX 6000	NVIDIA A100
SVHN	5×10^{-4}	0	Tesla T4	Quadro RTX 6000

work with the same 8-headed attention mechanism used in the official implementation. Table 26 summarizes the choice of optimization hyperparameters along with the GPU devices used for training and testing.

Table 25: Architectural details of CANPs for image completion tasks.

Benchmark	Context Encoder	Key Encoder	Query Encoder	Decoder
MNIST Fashion MNIST Omniglot FreyFace	$[3] \times [128, 128, 128] \times [128]$	$[1] \times [128] \times [128]$	$[1] \times [128] \times [128]$	$[130] \times [128, 128, 128] \times [2]$
SVHN	$[5] \times [128, 128, 128] \times [128]$	$[2] \times [128] \times [128]$	$[2] \times [128] \times [128]$	$[130] \times [128, 128, 128] \times [6]$

Table 26: Hyper-parameters and GPU devices used for training and testing CANPs on image completion tasks.

Benchmark	Learning rate	L2 regularizer	GPU (Training)	GPU (Testing)
MNIST	5×10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
Fashion MNIST	5×10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
Omniglot	5×10^{-4}	0	Quadro RTX 6000	Quadro RTX 6000
FreyFace	5×10^{-4}	10^{-5}	Quadro RTX 6000	NVIDIA A100
SVHN	5×10^{-4}	0	Quadro RTX 6000	NVIDIA A100

2.3.3 BNP

The architecture details for different components of the BNPs including the encoder, adaptation layer, and decoder are provided in table 27. For all the benchmarks, we use $k = 4$ and $k = 50$ bootstrap contexts for training and testing, respectively. The choice of optimization hyperparameters along with the GPU devices used for training and testing are included in table 28.

2.3.4 CQNP

The encoder and decoder architectures used for implementing CQNPs in different benchmarks are shown in table 29. Table 30 summarizes the choice of hyperparameters along with the GPU devices used for training and testing.

Table 27: Architectural details of BNPs for image completion tasks.

Benchmark	d_x	d_y	d_h	l_{pre}	l_{post}	l_{dec}
MNIST Fashion MNIST Omniglot FreyFace	2	1	128	5	3	5
SVHN	2	3	128	5	3	5

Table 28: Hyper-parameters and GPU devices used for training and testing BNPs on image completion tasks.

Benchmark	Learning rate	L2 regularizer	Scheduler	GPU (Training)	GPU (Testing)
MNIST	5×10^{-4}	0	cosine annealing	NVIDIA A100	Quadro RTX 6000
Fashion MNIST	5×10^{-4}	0	cosine annealing	NVIDIA A100	Quadro RTX 6000
Omniglot	5×10^{-4}	0	cosine annealing	Tesla T4	Quadro RTX 6000
FreyFace	5×10^{-4}	0	cosine annealing	Quadro RTX 6000	Quadro RTX 6000
SVHN	5×10^{-4}	0	cosine annealing	NVIDIA A100	Quadro RTX 6000

Table 29: Architectural details of CQNPs for image completion tasks.

Benchmark	Context Encoder	Decoder
MNIST Fashion MNIST Omniglot FreyFace	$[3] \times [128, 128, 128] \times [128]$	$[131] \times [128, 128, 128] \times [3]$
SVHN	$[5] \times [128, 128, 128] \times [128]$	$[131] \times [128, 128, 128] \times [9]$

Table 30: Hyper-parameters and GPU devices used for training and testing CQNPs on image completion tasks.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
MNIST	5×10^{-4}	0	25	50	NVIDIA A100	NVIDIA A100
Fashion MNIST	10^{-3}	10^{-5}	25	50	NVIDIA A100	NVIDIA A100
Omniglot	10^{-3}	10^{-5}	25	50	Quadro RTX 6000	NVIDIA A100
FreyFace	10^{-3}	0	25	50	Quadro RTX 6000	NVIDIA A100
SVHN	10^{-3}	10^{-5}	25	50	Quadro RTX 6000	NVIDIA A100

2.3.5 ACQNP

Compared to CQNP, ACQNP has an additional component named the adaptation layer which takes in the raw sample of u together with context representation and target inputs and maps them to a new set of quantile levels τ that we eventually approximate. Note that this is different from the adaptation layer used in BNPs. Also, we apply a sigmoid function to the outputs of the adaptation layer to make sure that they correspond to valid quantile levels. The depth and width of the MLPs used for parameterizing the encoder, adaptation layer, and decoder in ACQNPs are presented in table 31. We summarize the choice of hyperparameters along with the GPU models used in training and testing in table 32.

Table 31: Architectural details of ACQNPs for image completion tasks.

Benchmark	Context Encoder	Adaptor	Decoder
MNIST Fashion MNIST Omniglot FreyFace	$[3] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [3]$
SVHN	$[5] \times [128, 128, 128] \times [128]$	$[129] \times [128, 128, 128, 128, 128] \times [1]$	$[130] \times [128, 128, 128] \times [9]$

Table 32: Hyper-parameters and GPU devices used for training and testing ACQNPs on image completion tasks.

Benchmark	Learning rate	L2 regularizer	N_τ(Training)	N_τ(Testing)	GPU (Training)	GPU (Testing)
MNIST	10^{-3}	10^{-5}	25	50	NVIDIA A100	NVIDIA A100
Fashion MNIST	10^{-3}	10^{-5}	25	50	NVIDIA A100	NVIDIA A100
Omniglot	10^{-3}	10^{-5}	25	50	NVIDIA A100	NVIDIA A100
FreyFace	10^{-3}	10^{-5}	25	50	NVIDIA A100	Tesla T4
SVHN	10^{-3}	10^{-5}	25	50	NVIDIA A100	NVIDIA A100

References

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional conditional neural processes. In *International Conference on Learning Representations*, 2020.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.

Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.