FLOW MATCHING NEURAL PROCESSES

Hussen Abu Hamad
Department of Computer Science
University of Haifa
hussen.abu.hamad@gmail.com

Dan Rosenbaum
Department of Computer Science
University of Haifa
danro@cs.haifa.ac.il

ABSTRACT

Neural processes (NPs) are a class of models that learn stochastic processes directly from data and can be used for inference, sampling, and conditional sampling. We introduce a new NP model, which is based on flow matching, a generative modeling paradigm that has demonstrated strong performance on various data modalities. Our model is simple to implement, is efficient in training and evaluation, and outperforms previous state-of-the-art methods on various benchmarks including synthetic 1D Gaussian processes data, 2D images, and real-world weather data.

1 INTRODUCTION

Recent advances in generative machine learning are primarily driven by models capable of leveraging context information to enhance their predictions. To be effective across varying levels of contextual information, these models must handle multiple degrees of conditional uncertainty and therefore accurately capture conditional distributions at multiple levels of abstraction.

The neural process framework introduced by Garnelo et al. (2018a;b) provides a principled approach to context-based predictions by formulating the problem as learning stochastic processes from data, essentially training generative models of functions. Similarly to Gaussian processes, these models focus on continuous data and can be used as priors over functions capable of generating samples of arbitrary points along the functions. These target points can be predicted unconditionally or conditioned on a context of observed points along the function. This formulation has made neural processes a popular approach for modeling complex functional data in various domains.

In recent years, many different models have been proposed within the neural process framework, differing in their architecture and stochastic mechanisms. Early models focused on stochastic latent variables and were trained by variational inference Garnelo et al. (2018b). More recent approaches such as Nguyen & Grover (2022) leverage transformer-based architectures Vaswani et al. (2017) in an autoregressive setup, achieving improved performance in many scenarios.

Despite these advancements, several challenges remain unresolved. With the exception of autoregressive approaches, models tend to underfit the training functions, failing to capture intricate structures. On the other hand, autoregressive models, while more expressive, require sequential sampling of the target points one at a time and are therefore expensive to sample from. Additionally, these models lack a global or hierarchical representation of the function and the uncertainty which can limit their applicability and harm their performance in some cases. For example, the first dimensions in the autoregressive order are constrained to simple distributions.

In this paper, we introduce FlowNP, a new neural process model based on *flow matching* - a recent generative modeling paradigm that is closely related to diffusion and score-based models and has demonstrated strong performance in images and other modalities Liu et al. (2022); Lipman et al. (2022); Albergo & Vanden-Eijnden (2022). Flow matching operates by continuously transforming a simple and tractable initial distribution into the target data distribution by flowing through a continuous path of intermediate probabilities.

Our model is built on a transformer architecture where the input tokens include both the observed context points and intermediate values of the target points as generated across the flow path. At each step the model outputs a velocity vector, which guides the sampler in updating the target point values

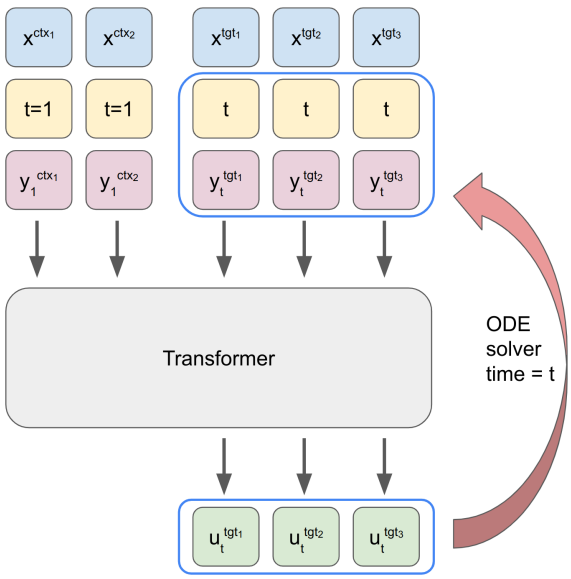

Figure 1: FlowNP: we model a probability flow of a stochastic process where at each step our model takes the values of the observed context points from the given function (ctx), and an intermediate value of the target points (tgt) at time t and predicts the velocities of the target points. With an ODE solver this can be used as a model of $p\left(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right)$ to generate samples or compute likelihoods.

for the subsequent step. This formulation allows FlowNP to capture complex structures effectively while enabling parallel sampling of all target points.

FlowNP offers several advantages over existing approaches. It is simple, efficient and capable of generating samples and computing likelihoods of any conditional distribution. In contrast to autoregressive models, it generates all target points in parallel, providing a more globally coherent and consistent representation of uncertainty. Extensive experimentation on standard NP benchmarks demonstrate that FlowNP consistently outperforms prior models, achieving state-of-the-art results across multiple datasets.

2 Background

2.1 Neural Processes

Neural processes (NP) Garnelo et al. (2018a;b), provide a framework for training generative models of functions, by modeling a *stochastic process* using a dataset of functions. Given such a dataset, a model is trained to predict an arbitrary *target* set of points along a function, based on a *context* set of observed points along the same function. More formally, if the functions are defined as $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$, models are trained to predict a conditional distribution over the target set:

$$p_\theta\left(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right) \tag{1}$$

where $x^{\texttt{ctx}} \in \mathcal{X}^M$ and $x^{\texttt{tgt}} \in \mathcal{X}^N$ are the positions of the M context points and N target points respectively, and $y^{\texttt{ctx}} \in \mathcal{Y}^M$ and $y^{\texttt{tgt}} \in \mathcal{Y}^N$ are the function evaluations of these points $y_i = f(x_i)$. Each of $\mathcal{X}$ and $\mathcal{Y}$ can be of arbitrary dimension. We put the context in '{}' brackets for better readability.

This approach of modeling stochastic processes follows from Kolmogorov's extension theorem (Øksendal, 2003) that states that a collection of distributions over sets is equivalent to a stochastic process if it meets two conditions: exchangeability and consistency.

Exchangeability. This condition requires that joint distributions over the sets are invariant to permutations. When using conditionals as in Eq. 1, this condition requires invariance to random permutations both for points in the target set and points in the context set. For example, the following equality should hold:

$$p_\theta(y_1, y_2|x_1, x_2, \{x_3, x_4, y_3, y_4\}) = p_\theta(y_2, y_1|x_2, x_1, \{x_4, x_3, y_4, y_3\}) \tag{2}$$

Consistency This condition requires that joint distributions are consistent with marginalizations. Since many NP models are designed to directly predict conditional distributions as in Eq. 1, this means two things. First, marginalizing over variables in the target set should be consistent with using the equivalently reduced target set. For example, the following equality should hold:

$$\int p_\theta(y_1, y_2|x_1, x_2, \{x_3, y_3\})dy_2 = p_\theta(y_1|x_1, \{x_3, y_3\}) \tag{3}$$

Additionaly, the condional distributions need to be consistent with different conditioning, i.e. they should obey the chain rule and Bayes' rule. For example:

$$p_\theta(y_1, y_2|x_1, x_2, \{\}) = p_\theta(y_1|x_1, \{x_2, y_2\}) \;\; p_\theta(y_2|x_2, \{\}) \tag{4}$$

2.2 Flow Matching

Flow matching Lipman et al. (2022); Liu et al. (2022); Albergo & Vanden-Eijnden (2022); Albergo et al. (2023) is a generative modeling paradigm that has recently achieved significant traction. It is closely related to diffusion modeling and its appeal stems from being both conceptually very simple and versatile.

The fundamental object in flow matching is a probability path $p_t(x)$ defined over a continuous time parameter $t \in [0, 1]$. This path defines a smooth transition between the two distributions $p_0(x)$ and $p_1(x)$. Most commonly, $p_0(x)$ corresponds to a simple tractable distribution such as a Gaussian, and $p_1(x)$ corresponds to a target distribution which is defined only through samples from the training data such as images.

In its simplest formulation, which we also adopt here, training is performed by generating samples of a conditional probability path $p_t(x_t|x_1)$ computed as an interpolation $tx_1 + (1 - t)x_0$ between an image x_1 and a noise sample x_0. Given this sample, a model is trained to predict a conditional velocity $u_t(x_t|x_1) = x_1 - x_0$, using a squared loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,x_1,x_t} \left\| u^\theta(x_t, t) - u_t(x_t|x_1) \right\|^2 \tag{5}$$

It can be shown that using the expectation over x_1 results in the model approximating the unconditional velocity $u_t(x_t) = \frac{d}{dt}x_t$ defining an ordinary differential equation (ODE) which can be used for both sampling and likelihood estimation.

In the case of sampling, this is achieved by first generating samples from the noise $p_0(x_0)$ and using an ODE solver to transform the sample into a sample from $p_1(x_1)$. Computing the likelihood of a given sample x_1 is done via the change of variable formula where the sample is transformed back from x_1 to a sample from the tractable noise distribution $p_0(x_0)$, then computing the likelihood of x_0 and correcting with an estimate of the Jacobian trace.

2.3 Related Work

Neural processes. A prominent approach to form distributions over continuous functions is by using Gaussian processes Rasmussen & Williams (2006). While they can be powerful, leveraging their capacity in modeling arbitrary functions is limited due to the complexity of training and evaluating them. To this end Neural processes were introduced, allowing the prior over functions to be learned from data in straightforward way. The first model in the NP class is the Conditional Neural Process (CNP) Garnelo et al. (2018a), which is deterministic and predicts an independent Gaussian distribution for each target point given the context. Later, the Neural Process (NP) Garnelo et al. (2018b) introduced a latent variable to allow capturing global uncertainty over the functions. Empirical evaluations of the approach was conducted by Le et al. (2018). Following, different extensions to the model were proposed. These include using attention mechanisms (ANP) Kim et al. (2019), translation invariance through convolutions Gordon et al. (2019), bootstrapping Lee et al. (2020), and more Bruinsma et al. (2021); Holderrieth et al. (2021); Ashman et al. (2024a;b).

Among the extensions of NP, a notable example is the Transformer Neural Process (TNP) Nguyen & Grover (2022) which treats the prediction of conditional distributions as sequence modeling. Of the three proposed variants in the TNP paper, the autoregressive model (TNP-A), implemented with a causal mask, achieves the highest scores on most common benchmarks, and here we refer to it

simply as TNP. Our model is implemented in a similar way to TNP, however, rather than predicting the next single target point in an autoregressive fashion, we use a transformer as a predictor of the flow matching velocity for all target points at once. We show that our model outperforms the TNP.

Diffusion of continuous spaces Following the success of diffusion models for data in discrete and finite spaces such as image grids Sohl-Dickstein et al. (2015); Song & Ermon (2019); Ho et al. (2020); Song et al. (2021) different works have investigated their extension to infinite and continuous spaces Kerrigan et al. (2022); Phillips et al. (2022); Pidstrigach et al. (2023); Xu et al. (2023); Bond-Taylor & Willcocks (2023); Mathieu et al. (2023). These approaches use diffusion to model joint distributions with some spatial structure. Given a joint distribution several approaches can be applied to predict conditional distributions. Pidstrigach et al. (2023) and Bond-Taylor & Willcocks (2023) use a guidance term computed based on the conditioning context and Kerrigan et al. (2022) and Dutordoir et al. (2023) use a replacement method Lugmayr et al. (2022) to generate conditional samples.

Using joint distributions over continuous spaces to predict conditional distributions can be seen as implementing neural processes using the definition of conditional distributions $p(x|y) = p(x, y)/p(y)$. The Neural Diffusion Process (NDP) Dutordoir et al. (2023) makes this connection explicitly and uses this method to evaluate the model on NP benchmarks. In comparison, our model uses a flow matching formulation, and is trained to capture both the joint distribution and arbitrary conditionals directly. We show that our model outperforms the NDP.

3 THE FLOWNP MODEL

Our goal is to implement a model of conditional distributions defined by any arbitrary target and context sets of points coming from an underlying function as formulated in Eq. 1.

We implement our model using a transformer architecture Vaswani et al. (2017) that predicts the velocities of the target variables at time t in the probability path defined by the continuous flow. The model is depicted in Fig. 1. We perform full self-attention between all input tokens, whereby each of the tokens are updated using an attention operator on all other tokens. The tokens we feed to the transformer are divided to context tokens and target tokens.

Target tokens: These tokens represent the intermediate values of the N function points $y^{\texttt{tgt}}$ evaluated at the N target positions $x^{\texttt{tgt}}$. For the evaluation at time t in the probability path, each token is formed by concatenating a single target point position together with the time t and the intermediate value of the variable at that time.

$$\texttt{token}^{\texttt{tgt}_i} = \texttt{embed}([x^{\texttt{tgt}_i},\ t,\ y_t^{\texttt{tgt}_i}]) \tag{6}$$

Context tokens: These tokens represent the observed points along the function, on which the distribution is conditioned. They are formed in the same way as the target tokens, except that they always contain the true observed function evaluations $y_1^{\texttt{ctx}} = y^{\texttt{ctx}}$, and the time $t = 1$, equivalent to the data distribution $p_1(y)$.

$$\texttt{token}^{\texttt{ctx}_i} = \texttt{embed}([x^{\texttt{ctx}_i},\ 1,\ y_1^{\texttt{ctx}_i}]) \tag{7}$$

Output tokens: The output of our model consist of the output tokens corresponding to the input target tokens. Each of these tokens is projected to the original dimension of the function values $\texttt{dim}(\mathcal{Y})$ and used as the velocity vector at time t in the continuous probability path defined by the model.

3.1 TRAINING

We train our model following the standard NP training setup using a conditional flow matching approach. At each training step, a minibatch of functions is extracted from the training set, where for each function two random sets of points are used as the context and target sets. For each step we randomly define different sizes of these two sets. Given the context and targets sets, we train our model as a velocity predictor by generating a random sample from an intermediate distribution along

the continuous probability path $y_t^{\texttt{tgt}} \sim p_t(y_t^{\texttt{tgt}}|y_1^{\texttt{tgt}})$. This is achieved by interpolating the clean data with random noise:

$$y_t^{\texttt{tgt}} = ty_1^{\texttt{tgt}} + (1-t)y_0^{\texttt{tgt}},\ y_0^{\texttt{tgt}} \sim \mathcal{N}(0, NI). \tag{8}$$

The loss is computed using a square error of model's output and the conditional velocity of the target points $u_t(y_t^{\texttt{tgt}}|y_1^{\texttt{tgt}}) = y_1^{\texttt{tgt}} - y_0^{\texttt{tgt}}$. Since the velocity model also sees the clean values of the context points, it can be viewed as a standard flow matching model only with access to side information about the target variables. In summary, the training loss is computed by:

$$\mathcal{L}(\theta) = \mathbb{E}\|u^\theta(y_t^{\texttt{tgt}}, t, x^{\texttt{tgt}}, x^{\texttt{ctx}}, y_1^{\texttt{ctx}}) - (y_1^{\texttt{tgt}} - y_0^{\texttt{tgt}})\|^2 \tag{9}$$

where the expectation is over:

$$f \sim \mathcal{F}, \{x^{\texttt{tgt}}, y_1^{\texttt{tgt}}, x^{\texttt{ctx}}, y_1^{\texttt{ctx}}\} \sim f, t \sim \mathcal{U}, y_0^{\texttt{tgt}} \sim \mathcal{N}$$

The training process is presented in Alg. 1

3.2 Evaluation

Once the model is trained, it can be used both for generating samples and computing likelihoods over data. In order to use the model to generate samples, random values of the target set at time $t = 0$ are drawn from a Normal distribution $y_0^{\texttt{tgt}} \sim \mathcal{N}(0, NI)$ and then they are used as the initial condition when solving the ODE from $t = 0$ to $t = 1$. This can be done with various ODE solvers where each evaluation of the velocity u_t is performed by calling the model with the additional input of target positions and context positions and values, $u_t \approx u^\theta(y_t^{\texttt{tgt}}, t, x^{\texttt{tgt}}, x^{\texttt{ctx}}, y_1^{\texttt{ctx}})$. The sampling process is presented in Alg. 2.

In order to compute likelihoods over data, a similar ODE is solved in the reverse direction. Starting from the target values $y_1^{\texttt{tgt}}$ at $t = 1$, samples are transformed back to $t = 0$ and their likelihood is evaluated with the standard Normal distribution. To accommodate for the change of variable formula, the Jacobian is estimated across the probability path using the Hutchinson trace estimator.

Running time Our model is based on a transformer architecture, therefore it is interesting to analyze its running time compared to the TNP. First, the number of tokens used for predicting N target points given a context of M points in our model is $N + M$ compared to the TNP model which uses $2N + M$. Second, generating samples with TNP requires N evaluations as it is an autoregressive model. For our model the number of evaluation depends on the ODE solver and is a parameter that can be tuned according to the required accuracy, however it is independent of the number of target points N. The main disadvantage of our model compared to TNP is in evaluating likelihoods, where TNP requires only one model evaluation, and our model, similar to sampling, requires multiple evaluations as part of the ODE solution. Wall-time Comparisons for some of the experiments are reported in App. D.

3.3 Exchangeability and Consistency

In this section we discuss the properties of our model in relation to the requirements of the Kolmogorov extension theorem as presented in Sec. 2.1, namely the exchangeability and consistency properties.

Within a given conditional prediction task, $p(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\})$ our model is guaranteed to be invariant with respect to permutations of both the context and the target, and therefore complies with the exchangeability property. This is due to the transformer architecture and the treatment of tokens as sets rather than sequences. Specifically, all tokens undergo the exact same processing and we do not use positional encodings that rely on ordering.

On the other hand, the consistency property is not implied by the architecture of our model and therefore it is not guaranteed. However, even though it is not guaranteed by the model itself, the training paradigm of NPs promotes the consistency property. This is because for every training sample, the context and target sets are constructed randomly from a true underlying function, which is itself a sample from the stochastic process defined by the training set. Therefore the training objective is a Monte Carlo estimate of the ground truth stochastic process. We empirically show an example where our model learns to be consistent in App. D.

Table 1: Comparison of log-likelihood computed on the target set for various 1D GP datasets. For CNP, NP and ANP on RBF Matern-$\frac{5}{2}$ and Periodic we report results from Nguyen & Grover (2022) while for the rest we provide mean and standard deviation over five runs. FlowNP outperforms all other models across all datasets, except in two cases where TNP achieves comparable performance.

MODEL	RBF	MATERN-$\frac{5}{2}$	PERIODIC	FIXED-NOISY RBF	FIXED-NOISY MATERN-$\frac{5}{2}$
CNP	0.26 ± 0.02	0.04 ± 0.02	-1.40 ± 0.02	-1.00 ± 0.02	-1.09 ± 0.02
NP	0.27 ± 0.01	0.07 ± 0.01	-1.15 ± 0.04	-1.18 ± 0.02	-1.20 ± 0.01
ANP	0.81 ± 0.00	0.63 ± 0.00	-5.02 ± 0.21	-0.91 ± 0.02	-0.99 ± 0.02
TNP	1.65 ± 0.02	$\mathbf{1.29 \pm 0.02}$	-0.58 ± 0.01	0.68 ± 0.01	$\mathbf{0.30 \pm 0.01}$
NDP(REIMPLEMENTED)	0.90 ± 0.03	0.7 ± 0.03	-0.55 ± 0.00	0.53 ± 0.00	0.19 ± 0.02
FLOWNP (OURS)	$\mathbf{1.68 \pm 0.02}$	$\mathbf{1.29 \pm 0.02}$	$\mathbf{-0.50 \pm 0.01}$	$\mathbf{0.71 \pm 0.01}$	$\mathbf{0.30 \pm 0.01}$

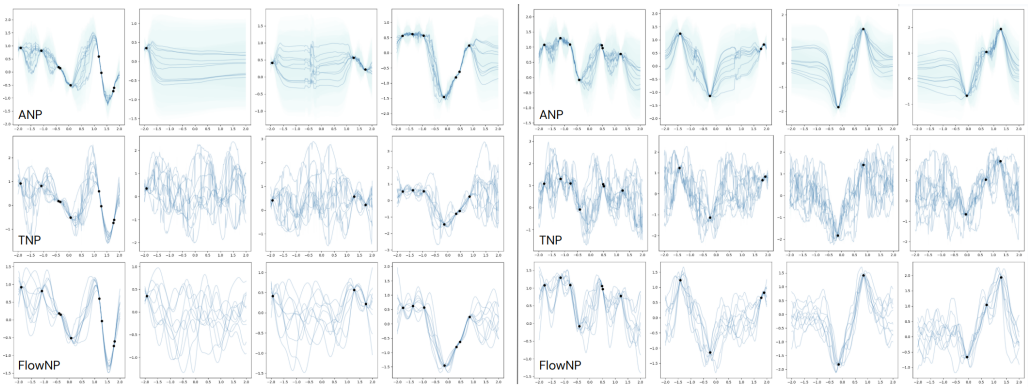

Figure 2: Samples from models trained on an RBF kernel (left) and a Matern-$\frac{5}{2}$ kernel (right). ANP captures the uncertainty mostly through its predicted local variance (shaded area) while TNP and FlowNP can generate coherent samples that cover the global uncertainty. In contrast to TNP that generates the samples point-by-point, FlowNP generates all points in parallel, resulting in smoother samples.

We note that also for previous models of NPs, consistency is not completely guaranteed and holds only in a limited sense. Specifically, while models like CNP, NP and TNP are consistent over marginalization (Eq. 3), they are not consistent over conditioning (Eq. 4). On the other hand models like NDP are consistent over conditioning and not over marginalization. For more discusison on this see Sec. A.

4 EXPERIMENTS

We implement the same FlowNP model for all experiments, differing only in the dimensions of the input $\texttt{dim}(\mathcal{X})$ and output $\texttt{dim}(\mathcal{Y})$. We use a transformer with 6 layers of full self attention, 128 hidden dimensions and 4 attention heads. We use sinusoidal positional encodings for the input x with 10 frequencies per dimension. We emphasize that the encoding is a function of x rather than the position in the sequence. For likelihood evaluation and sampling, we use an ODE solver based on the *midpoint* method with 100 steps implemented by Lipman et al. (2024). The transformer architecture and implementation of our model are based on Nguyen & Grover (2022). We provide the code of our model in the supplementary and we will open source the full repository including code of all models and experiments upon publication.

Baselines We compare to the baselines CNP Garnelo et al. (2018a), NP Garnelo et al. (2018b) and ANP Kim et al. (2019) using the implementation in Nguyen & Grover (2022) and Lee et al. (2020), to TNP Nguyen & Grover (2022) and to a reimplemented version of NDP Dutordoir et al. (2023) denoted by NDP(reimplemented). Both TNP and NDP models share our network architecture and differ only in the training objective and evaluation method. Specifically, TNP uses the same transformer as we do but is trained by autoregressive maximum likelihood using a causal mask. We use the code published by the authors. For the NDP baseline we reimplement the model with the

Table 2: Target log-likelihood mean $\pm$ 1 standard deviation for the EMNIST and CelebA datasets.

MODEL	EMNIST 0-9	EMNIST 10-46	CELEBA
CNP	0.73 ± 0.00	0.49 ± 0.01	2.15 ± 0.01
NP	0.79 ± 0.01	0.59 ± 0.01	2.48 ± 0.02
ANP	0.98 ± 0.00	0.89 ± 0.00	2.90 ± 0.00
TNP	2.06 ± 0.00	1.74 ± 0.01	4.13 ± 0.02
NDP(REIMPLEMENTED)	1.20 ± 0.01	1.09 ± 0.01	3.42 ± 0.02
FLOWNP (OURS)	$\mathbf{2.49 \pm 0.01}$	$\mathbf{2.41 \pm 0.02}$	$\mathbf{7.88 \pm 0.01}$

same transformer used in FlowNP, omitting the bi-directional attention used in the original NDP. Therefore, it differs from our model only by training on unconditional joint distributions and using a diffusion loss based on predicting the denoised data rather than the flow velocity. We evaluate the model likelihood by the probability flow ODE with the same ODE solver used by FlowNP.

4.1 SYNTHETIC 1D FUNCTIONS

We start with the standard benchmarks of NPs generated from synthetic 1D Gaussian processes (GP). We follow the evaluation protocols of both Lee et al. (2020) and Bruinsma et al. (2021), which are used respectively in the TNP Nguyen & Grover (2022) and NDP Dutordoir et al. (2023) papers. In the first Lee et al. (2020), we generate data from three different GP kernels: RBF (also called squared exponential - SE), Matern-$\frac{5}{2}$ and Periodic. For each kernel, functions are generated using parameters which are randomly chosen for each sampled function separately. Evaluation is done on held-out data using a random number of context and target points sampled uniformly between 3 and 47 where the total number is constrained to be equal or less than 50. In the second protocol Bruinsma et al. (2021) we use two GP kernels: RBF and Matern-$\frac{5}{2}$, using a *fixed* set of parameters, and additional Gaussian observation noise with variance 0.05^2. Evaluation is done on held-out data with the same parameters, a uniformly random context size between 1 and 10, and a fixed number of 50 target points. These datasets are denoted by Fixed-Noisy RBF and Fixed-Noisy Matern-$\frac{5}{2}$. Likelihood for the latent variable models NP and ANP are computed with variational importance sampling Burda et al. (2015) using 50 samples, and for the NDP and FlowNP using the midpoint ODE solver with 100 steps.

Results are shown in Table 1. We run all experiments 5 times with different random seeds and evaluation sets and report the mean and standard deviation. For CNP ANP and NP on RBF, Matern-$\frac{5}{2}$ and Periodic we report the results from Nguyen & Grover (2022). Our model, FlowNP, outperforms all other models in all the benchmarks except for Matern-$\frac{5}{2}$ and Fixed-Noisy Matern-$\frac{5}{2}$ where FlowNP and TNP perform similarly. We also show samples for models trained on the RBF and Matern kernels with fixed parameters in Fig. 2. We compare ANP: a latent variable model trained with variational inference; TNP: an autoregressive model; and our proposed model, FlowNP. ANP largely fails to capture the global uncertainty with the latent variable, and tends to attribute large variances to local uncertainty, depicted as the shaded area corresponding to the predicted variance of each point. In contrast, TNP and FlowNP generate plausible samples that cover the global uncertainty. However while TNP generates samples in an autoregressive manner sometimes resulting in a jumpy behavior, FlowNP generates all points in parallel mostly resulting in smoother samples. For a visualiztion of the sampling process see Fig. 5 in the appendix.

4.2 IMAGES

We follow with experiments on two image datasets, EMNIST Cohen et al. (2017) and CelebA Liu et al. (2018), where the input $\mathcal{X}$ is 2-dimensional and represents the spacial position in the image, and the output $\mathcal{Y}$ is either a 1D grayscale value or a 3D RGB value of the pixels. For EMNIST, which contains images of characters, we train on the 10 first classes (digits 0-9) and evaluate both the in-distribution performance, and the out-of-distribution performance on the rest of the characters. Since the background in the images is effectively constant, the likelihood can grow arbitrarily and therefore the predicted variance needs to be lower bounded. Since we cannot bound the variance in a straightforward way, in order to make a fair comparison with the baseline models that do bound the variance we add small noise to the data with variance 0.01^2. For the TNP baseline we optimize the variance bound. Results are shown in Table 2. FlowNP outperforms all other models both for the

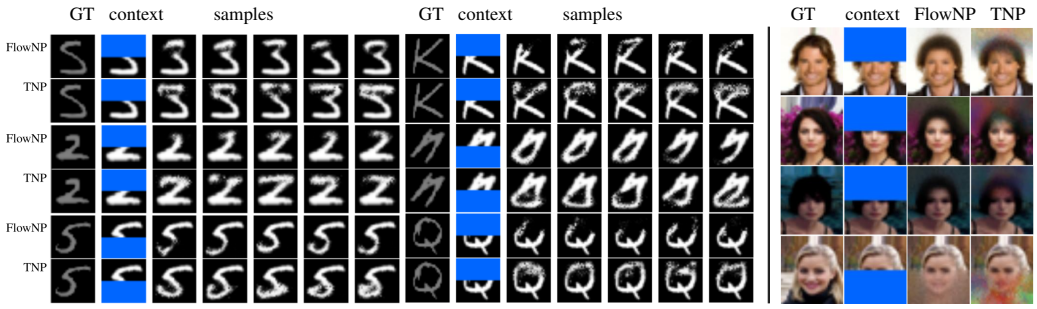

Figure 3: Conditional samples generated by FlowNP and TNP for in-distribution EMNIST, out-of-distribution EMNIST and CelebA. We show 5 samples for EMNIST and the average of 30 samples for CelebA. Samples from FlowNP are generally sharper, more coherent and more diverse.

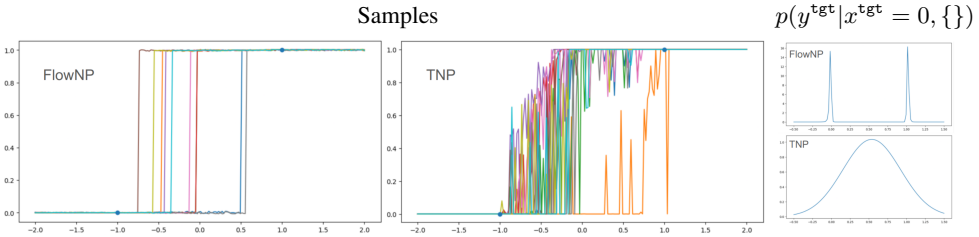

Figure 4: Analysis of FlowNP vs. TNP for a random step function. Samples from FlowNP capture the sharp transition occurring in random positions, while TNP cannot model this function as each step in the autoregressive prediction is Gaussian. On the right: the marginal distribution of y predicted for a single point where $x = 0$ further demonstrates this and highlights FlowNP's capacity to capture multimodal distributions.

in-distribution and out-of-distribution evaluations. We show samples generated by FlowNP and TNP of three in-distribution and out-of-distribution images, using half of the image as context. Samples from FlowNP are slightly sharper and more diverse, and are generated by four times less evaluation steps compared to TNP.

Results for CelebA are shown in Table 2 on the right. FlowNP outperforms all other models. We show image completion results (Fig. 3) where images are computed by averaging 30 samples. In comparison to TNP, FlowNP results in sharper and more coherent samples.

4.3 More Results, Analysis and Ablations

We provide results of more experiments on real-world dynamics using meteorological data in the appendix (App. C). Furthermore, we provide analysis of FlowNP compared to the autoregressive approach of TNP (Fig. 4) and ablations compared to NDP. For further discussion see App. D.

5 Discussion

We presented FlowNP, a new model which implements neural processes using flow matching. This model is simple, efficient and outperforms previous neural processes models. We showed results on standard benchmarks such as 1D GP data and 2D images as well as real-world weather prediction data. We find that modeling conditionals is favorable over modeling the joint and that using a flow matching objective is favorable over diffusion. Compared to TNP, FlowNP can efficiently capture non-Gaussian and multimodal distributions.

Limitations: The main limitation we find with this model is the iterative sampling that can be costly compared to feed forward models such as ANP. Compared to TNP, in most cases FlowNP would need less iterations however for likelihood computation and in cases where point-by-point prediction is needed TNP could be more efficient.

REFERENCES

Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

Matthew Ashman, Cristiana Diaconu, Junhyuck Kim, Lakee Sivaraya, Stratis Markou, James Requeima, Wessel P Bruinsma, and Richard E Turner. Translation equivariant transformer neural processes. *International Conference on Machine Learning*, 2024a.

Matthew Ashman, Cristiana Diaconu, Adrian Weller, Wessel Bruinsma, and Richard E Turner. Approximately equivariant neural processes. *Advances in neural information processing systems*, 2024b.

Sam Bond-Taylor and Chris G Willcocks. ∞-diff: Infinite resolution diffusion with subsampled mollified states. *arXiv preprint arXiv:2303.18242*, 2023.

Wessel P Bruinsma, James Requeima, Andrew YK Foong, Jonathan Gordon, and Richard E Turner. The gaussian neural process. *arXiv preprint arXiv:2101.03606*, 2021.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Vincent Dutordoir, Alan Saul, Zoubin Ghahramani, and Fergus Simpson. Neural diffusion processes. In *International Conference on Machine Learning*, pp. 8990–9012. PMLR, 2023.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pp. 1704–1713. PMLR, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. In *International Conference on Learning Representations*, 2019.

Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, et al. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

Peter Holderrieth, Michael J Hutchinson, and Yee Whye Teh. Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes. In *International Conference on Machine Learning*, pp. 4297–4307. PMLR, 2021.

Gavin Kerrigan, Justin Ley, and Padhraic Smyth. Diffusion generative models in infinite dimensions. *arXiv preprint arXiv:2212.00886*, 2022.

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.

Tuan Anh Le, Hyunjik Kim, Marta Garnelo, Dan Rosenbaum, Jonathan Schwarz, and Yee Whye Teh. Empirical evaluation of neural process objectives. In *NeurIPS workshop on Bayesian Deep Learning*, volume 4, 2018.

Juho Lee, Yoonho Lee, Jungtaek Kim, Eunho Yang, Sung Ju Hwang, and Yee Whye Teh. Bootstrapping neural processes. *Advances in neural information processing systems*, 33:6606–6615, 2020.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code, 2024. URL `https://arxiv.org/abs/2412.06264`.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11461–11471, 2022.

Emile Mathieu, Vincent Dutordoir, Michael J. Hutchinson, Valentin De Bortoli, Yee Whye Teh, and Richard E. Turner. Geometric Neural Diffusion Processes, 2023. URL `http://arxiv.org/abs/2307.05431`.

Radford M Neal. Regression and classification using gaussian process priors. *Bayesian statistics*, 6: 475, 1998.

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. 2022.

Angus Phillips, Thomas Seror, Michael Hutchinson, Valentin De Bortoli, Arnaud Doucet, and Emile Mathieu. Spectral Diffusion Processes, 2022. URL `http://arxiv.org/abs/2209.14125`.

Jakiw Pidstrigach, Youssef Marzouk, Sebastian Reich, and Sven Wang. Infinite-dimensional diffusion models. *arXiv preprint arXiv:2302.10130*, 2023.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Jin Xu, Emilien Dupont, Kaspar Märtens, Tom Rainforth, and Yee Whye Teh. Deep Stochastic Processes via Functional Markov Transition Operators, 2023. URL `http://arxiv.org/abs/2305.15574`.

Bernt Øksendal. Stochastic Differential Equations. 2003.

A Consistency of Prior Models

We discuss the consistency property of different models in more detail. The original CNP and NP models use architectures that ensure that given a context, all target points are predicted independently.

$$p(y_1, y2|x_1, x_2, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\} = p(y_1|x_1, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\ p(y_2|x_2, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\} \tag{10}$$

This implies that consistency over marginalization as in Eq. 3 holds. However, nothing in the model structure guarantees consistency over conditioning rule as in Eq. 4. This is because computing the model conditioned on different contexts (e.g. the term on the left hand side and the first term on the right hand side of Eq. 4) cannot be guaranteed to lead to consistent results that comply with an underlying joint distribution.

TNP, which is based on a transformer performs best when it is used as an autoregressive model. In that case it does not comply neither with exchangeability nor consistency, since even when using clever masking to make the model invariant to the context, the joint distribution still depends on the ordering of the autoregression in the target points. While the authors of TNP propose different variants of the models that make it exchangeable and consistent with marginalization, they lead to a significant drop in performance, and are still not guaranteed to be consistent in terms of conditioning.

The reason that NDP is consistent over the chain rule, is that it only models joint distributions and therefore computes conditionals using the conditional definition. However the full self-attention architecture leads to predictions of target points that cannot be separated into independent factors and therefore cannot ensure consistency over marginalizations.

B Training and Sampling

The algortithms for training and sampling are provided in Alg. 1 and Alg. 2. Fig. 5 provides a demonstration of the sampling process transforming random noise to samples from the conditional distribution of $p(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\})$.

Algorithm 1 Training

input: dataset of functions $\mathcal{F}$
repeat
  $f_i \sim \mathcal{F}$ — Sample a batch of functions, here shown for 1
  $M, N \sim \mathcal{U}$ — Random context and target sizes
  $x^{\texttt{ctx}} \sim \mathcal{X}^M, x^{\texttt{tgt}} \sim \mathcal{X}^N$ — Sample positions
  $y^{\texttt{ctx}} \leftarrow f_i(x^{\texttt{ctx}}), y^{\texttt{tgt}} \leftarrow f_i(x^{\texttt{tgt}})$
  $t \sim \mathcal{U}[0, 1]$ — Sample flow time
  $y_0^{\texttt{tgt}} \sim \mathcal{N}(0, NI)$ — Sample noise
  $y_t^{\texttt{tgt}} \leftarrow ty^{\texttt{tgt}} + (1-t)y_0^{\texttt{tgt}}$
  $\texttt{tokens}^{\texttt{ctx}} = \{\texttt{embed}^\theta([x^{\texttt{ctx}},\ 1,\ y^{\texttt{ctx}}])\}$ — M tokens
  $\texttt{tokens}^{\texttt{tgt}} = \{\texttt{embed}^\theta([x^{\texttt{tgt}},\ t,\ y_t^{\texttt{tgt}}])\}$ — N tokens
  $\hat{u}_t \leftarrow u^\theta([\texttt{tokens}^{\texttt{ctx}}, \texttt{tokens}^{\texttt{tgt}}])$
  $\mathcal{L}(\theta) = \|\hat{u}_t - (y^{\texttt{tgt}} - y_0^{\texttt{tgt}})\|^2$
  $\theta \leftarrow \texttt{update}\left(\frac{\partial}{\partial\theta}\mathcal{L}(\theta)\right)$
until convergence
output: trained parameters θ.

Algorithm 2 Sampling with a midpoint ODE solver

input: context $x^{\texttt{ctx}}, y^{\texttt{ctx}}$ and target positions $x^{\texttt{tgt}}$
$\texttt{tokens}^{\texttt{ctx}} = \{\texttt{embed}^{\theta}([x^{\texttt{ctx}},\ 1,\ y^{\texttt{ctx}}])\}$ — M tokens
$y^{\texttt{tgt}} \sim \mathcal{N}(0, NI)$ — Random initialization
for $t = 0$ **to** 1 in $\delta = 1/\text{n_steps}$ increments **do**
  $\texttt{tokens}^{\texttt{tgt}} = \{\texttt{embed}^{\theta}([x^{\texttt{tgt}},\ t,\ y^{\texttt{tgt}}])\}$ — N tokens
  $\hat{u}_t \leftarrow u^{\theta}([\texttt{tokens}^{\texttt{ctx}}, \texttt{tokens}^{\texttt{tgt}}])$
  $y^{\texttt{tgt}}_{\texttt{mid}} \leftarrow y^{\texttt{tgt}} + \frac{\delta}{2}\hat{u}_t$
  $\texttt{tokens}^{\texttt{tgt}}_{\texttt{mid}} = \{\texttt{embed}^{\theta}([x^{\texttt{tgt}},\ t + \frac{\delta}{2},\ y^{\texttt{tgt}}_{\texttt{mid}}])\}$
  $\hat{u}_{t_{\texttt{mid}}} \leftarrow u^{\theta}([\texttt{tokens}^{\texttt{ctx}}, \texttt{tokens}^{\texttt{tgt}}_{\texttt{mid}}])$
  $y^{\texttt{tgt}} \leftarrow y^{\texttt{tgt}} + \delta\hat{u}_{t_{\texttt{mid}}}$
end for
output: predicted sample $y^{\texttt{tgt}}$.

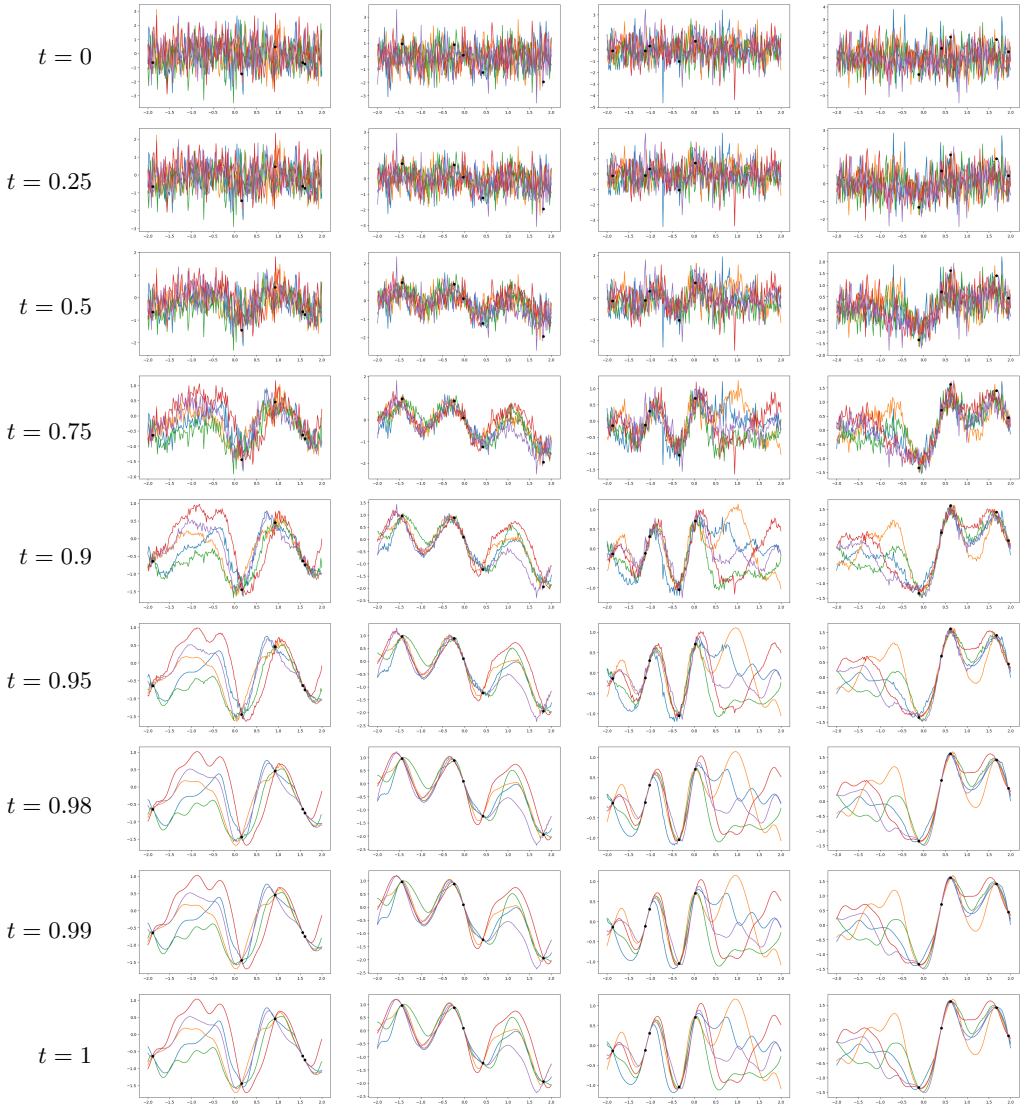

Figure 5: A visualization of the sampling process, showing the intermediate values $y_t^{\texttt{tgt}}$ for different time steps. Each column is conditioned on a different context and each color represents a different random sample generated by FlowNP.

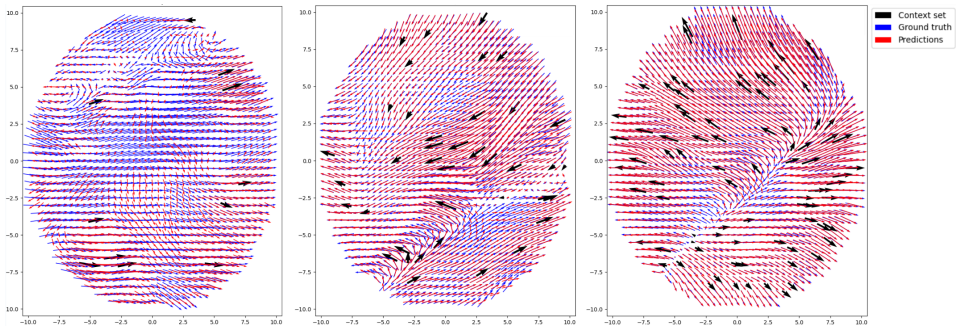

Figure 6: Visualization of wind prediction using conditional samples generated by FlowNP on three held-out data points from ERA5. The model generates coherent samples based on the context points. Results are more accurate as more context points are given (right).

MODEL	ERA5
TNP	7.9446
FLOWNP (OURS)	**11.6083**

Table 3: Target log-likelihood for the ERA5 dataset. Predictions are 4D and consist of temperature, pressure, and 2D wind direction.

C WEATHER DATA

To assess the performance of the FlowNP model in real-world dynamic and multi-dimensional tasks, we use meteorological data from the ERA5 global dataset Hersbach et al. (2020). We follow the setup in Holderrieth et al. (2021) and extract data from a circular region centered around Memphis, USA. Each sample represents a weather snapshot at a single point in time and consists of multiple meteorological variables: temperature, pressure, and two wind components (eastward and northward). We use about 34K training samples and train using the NP setup where $\mathcal{X}$ represents the 2D longitude and latitude, $\mathcal{Y}$ represents the 4D meteorological variables and random subsets are used as the context and target sets. A quantitative comparison of the target set log-likelihood between FlowNP and TNP is reported in Table 3 and conditional samples of wind directions predicted by FlowNP are presented in Fig. 6.

MODEL	RBF	EMNIST
NDP(REIMPLEMENTED)	0.90	1.20
ABL1: DENOISER	1.06	1.24
ABL2: JOINT	1.37	2.44
FLOWNP (OURS)	1.68	2.49

Table 4: Target log-likelihood for ablations connecting our model to NDP. ABL1 is similar to FlowNP but trained with a denoiser objective over a variance preserving probablity path. ABL2 is similar to FLowNP but is trained to predict only joint distributions rather than conditionals.

D ANALYSIS AND ABLATION

FlowNP vs. TNP We consider the 1D step function, where the output changes from $y = 0$ to $y = 1$ at a random point of x. This function was used in Neal (1998) and Dutordoir et al. (2023) as an example that cannot be captured by Gaussian processes. Here we show that an autoregressive transformer-based model like TNP also cannot capture this function accurately. In Fig. 4 we show samples from TNP and FlowNP. While TNP samples are noisy in the transition area, FlowNP samples are sharp. The reason is that even though TNP can model complex distributions via the autoregression, it is constraind to Gaussians at every step. This is evident when looking at the marginal distribution at $x = 0$, shown on the right of Fig. 4 for both models. Comparing the marginal distribution of a single point with samples from the entire function indicates that, even without a formal guarantee, FlowNP's training approximately preserves the consistency property (Eq. 3).

Running time Comparing the sampling time between FlowNP and TNP using an NVIDIA RTX4090 GPU and 100 sampling steps for FlowNP, the time to generate 1 sample in the GP experiment with 200 target points is 0.3sec for FlowNP and 0.8sec for TNP, and 1 EMNIST sample with 748 target points is 4.1sec for FlowNP and 72.6sec for TNP.

Number of Samples We run an ablation on sampling and log-likelihood evaluation for a different number of ODE steps. For GP-RBF data, the mean of 5 random seeds with number of steps (10, 20, 40, 60, 80, 100) are the following (the std of for all is 0.014): 1.89, 1.76, 1.70, 1.69, 1.69, 1.69. This shows that the evaluation plateaus after T=60 steps. We observe similar behavior for sample quality.

FlowNP vs. NDP We conduct an ablation to test the two aspects on which the models differ: (1) Modeling the joint distribution vs. modeling random conditionals and (2) the objectives of diffusion with a variance preserving schedule vs. conditional flow matching. The results are shown in Table. 4 suggesting that both aspects contribute to FlowNP's performance.