Flow Matching Neural Processes

Hussen Abu Hamad
Department of Computer Science
University of Haifa

Dan Rosenbaum
Department of Computer Science
University of Haifa

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 based on flow matching, a generative modeling paradigm that has demonstrated strong performance on various data modalities. Following the NP training framework, the model provides amortized predictions of conditional distributions over any arbitrary points in the data. Compared to previous NP models, our model is simple to implement and can be used to sample from conditional distributions using an ODE solver, without requiring auxiliary conditioning methods. In addition, the model provides a controllable tradeoff between accuracy and running time via the number of steps in the ODE solver. We show that our model outperforms previous state-of-the-art neural process 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 (NP) framework introduced by Garnelo et al. [13, 14] 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, NP models 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 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 [14]. More recent approaches such as Nguyen & Grover [32] leverage transformer-based architectures [40] 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.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

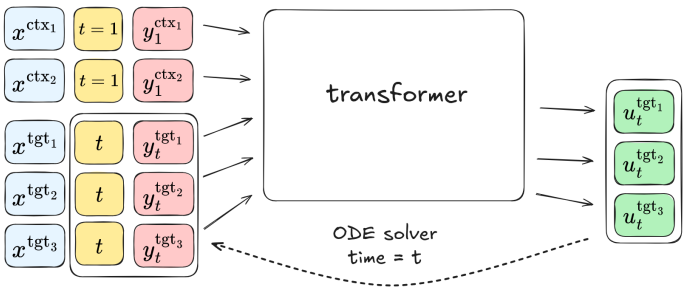

Figure 1: FlowNP: we model a probability flow of a stochastic process where at each step our model takes the values of observed context points from the given function (ctx), and an intermediate value of the target points (tgt) at time t and predicts the velocities of the target points. With an ODE solver 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.

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 [27, 25, 1]. 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. The approach enables both generating samples and computing log-likelihood over data by solving an ODE.

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 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: (1) it is simple to implement, train and use for sampling and inference, (2) it is capable of generating samples and computing likelihoods of any conditional distribution, (3) in contrast to autoregressive models, it generates all target points in parallel, providing a more globally coherent and consistent representation of uncertainty, and (4) inference time is controlled by the number of ODE steps rather than the number of target points. Extensive experimentation on standard NP benchmarks demonstrate that FlowNP consistently outperforms prior NP models, achieving state-of-the-art results across multiple datasets.

Our implementation is available at `https://github.com/danrsm/flowNP`.

2 Background

2.1 Neural Processes

Neural processes (NP) [13, 14], 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 $y_i = f(x_i)$. $\mathcal{X}$ and $\mathcal{Y}$ can be of arbitrary dimensions. The context is placed in '$\{\}$' brackets for better readability.

This approach of modeling stochastic processes follows from Kolmogorov's extension theorem [42] stating that stochastic processes can be defined via a collection of joint distributions over finite sets, $p_{x_{1:n}}(y_{1:n})$, if it meets two conditions: exchangeability and consistency.

Exchangeability. This condition requires that all joint distributions over the sets are equivariant with respect to permutations:

$$p_{x_{1:n}}(y_{1:n}) = p_{\pi(x_{1:n})}(\pi(y_{1:n})) \coloneqq p_{x_{\pi(1),\ldots,\pi(n)}}(y_{\pi(1),\ldots,\pi(n)}), \tag{2}$$

for any given permutation π. Models that directly predict conditional distributions such as in Eq. 1 should also be invariant to permutations in the context set.

Consistency. This condition requires that joint distributions are consistent with marginalizations:

$$p_{x_{1:m}}(y_{1:m}) = \int p_{x_{1:n}}(y_{1:n}) dy_{m+1:n}, \quad \forall 1 \leq m \leq n. \tag{3}$$

For models that directly predict conditional distributions as in Eq. 1, this translates to the following formulation of *marginal consistency*:

$$p_\theta\left(y_{1:m} | x_{1:m}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right) = \int p_\theta\left(y_{1:n} | x_{1:n}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right) dy_{m+1:n}, \quad \forall 1 \leq m \leq n, \tag{4}$$

and additionally, models should hold *conditional consistency*, i.e. the chain rule:

$$p_\theta\left(y_{s^1+s^2} | x_{s^1+s^2}, \{\}\right) = p_\theta\left(y_{s^1} | x_{s^1}, \{x_{s^2}, y_{s^2}\}\right) p_\theta\left(y_{s^2} | x_{s^2}, \{\}\right), \tag{5}$$

for any two sets of indexes s^1 and s^2. We note that while most NP models follow the exchangeability condition, they do not fully guarantee that the consistency conditions hold (see App. A).

2.2 Flow Matching

Flow matching [25, 27, 1, 2] 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 a data sample 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:

$$\mathcal{L}(\theta) = \mathbb{E}_{t, x_1, x_t} \left\| u^\theta(x_t, t) - u_t(x_t|x_1) \right\|^2 \tag{6}$$

It can be shown that 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.

For sampling, we first generate a sample from the noise $p_0(x_0)$ and then use an ODE solver to transform it into a sample from the data distribution $p_1(x_1)$. To compute the likelihood of a given sample x_1, we employ the change of variable formula. This involves transforming x_1 back to a corresponding noise sample from the tractable distribution $p_0(x_0)$, and then calculating the likelihood of x_0 while correcting for the flow's volume change 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 [36]. 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 a straightforward way. The first model in the NP class is the Conditional Neural Process (CNP) [13] which is deterministic and predicts an independent Gaussian distribution for each target point given the context. Later, the Neural Process (NP) [14] introduced a latent variable to allow capturing global uncertainty over the functions. Empirical evaluations of the approach was conducted by Le et al. [22]. Following, different extensions to the model were proposed. These include using attention mechanisms (ANP) [21], translation invariance through convolutions [15], bootstrapping [23], and more [8, 18, 3, 4].

Among the extensions of NP, a notable example is the Transformer Neural Process (TNP) [32] 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. Autoregressive-CNP [7] shows that models can be effectively deployed as autoregressive NPs even if they were not trained autoregressively. However, the autoregressive approach has limitations that we address here, namely, the potential complexity of generating samples point-by-point, and the failure to capture complex distributions in the first points in the autoregressive order. An interesting model that combines autoregressive modeling with diffusion [6] treats the full sampling process as a sequence. Our model is implemented with a transformer, like TNP, however, rather than predicting the next single target point in an autoregressive fashion, we use the transformer as a predictor of the flow matching velocity for all target points at once. We show that our model outperforms the TNP.

Diffusion on continuous spaces Following the success of diffusion models for data in discrete and finite spaces such as image grids [37, 38, 17, 39] different works have investigated their extension to infinite and continuous spaces [19, 34, 35, 41, 5, 30, 20]. These approaches use diffusion or flow matching to model joint distributions with some spatial structure. Given a joint distribution several approaches can be applied to predict conditional distributions. Pidstrigach et al. [35] and Bond-Taylor & Willcocks [5] use a guidance term computed based on the conditioning context and Kerrigan et al. [19] and Dutordoir et al. [11] use a replacement method [29] to generate conditional samples.

Using joint distributions over continuous spaces to predict conditional distributions can be seen as an implemention of neural processes using the definition of conditional distributions $p(x|y) = p(x, y)/p(y)$. The Neural Diffusion Process (NDP) [11] 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 joint distributions and arbitrary conditionals directly, thus amortizing the computation of conditioning. We show that our model outperforms the NDP on standard benchmarks, and that amortizing the conditioning enables directly generating conditional samples without needing auxiliary conditioning methods such as guidance or replacement.

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 [40] 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 $y^{\mathtt{tgt}}$ evaluated at the N target positions $x^{\mathtt{tgt}}$. 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.

$$\mathtt{token}^{\mathtt{tgt}_i} = \mathtt{embed}([x^{\mathtt{tgt}_i},\ t,\ y_t^{\mathtt{tgt}_i}]) \tag{7}$$

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 $y_1^{\mathtt{ctx}} = y^{\mathtt{ctx}}$, and the time $t = 1$, equivalent to the data distribution $p_1(y)$.

$$\mathtt{token}^{\mathtt{ctx}_i} = \mathtt{embed}([x^{\mathtt{ctx}_i},\ 1,\ y_1^{\mathtt{ctx}_i}]) \tag{8}$$

Output tokens: The output tokens corresponding to the input target tokens are each projected to the original dimension of the function values $\mathtt{dim}(\mathcal{Y})$ 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 target 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:

$$y_t^{\texttt{tgt}} = ty_1^{\texttt{tgt}} + (1-t)y_0^{\texttt{tgt}},\ y_0^{\texttt{tgt}} \sim \mathcal{N}(0, I). \tag{9}$$

The loss is computed using a squared error between the 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 model has access to the clean values of the context points, it effectively has side information about the velocity of target variables. In summary, the training loss is computed by:

$$\mathcal{L}(\theta) = \mathbb{E}\|u^\theta(y_t^{\texttt{tgt}}, t, x^{\texttt{tgt}}, x^{\texttt{ctx}}, y_1^{\texttt{ctx}}) - (y_1^{\texttt{tgt}} - y_0^{\texttt{tgt}})\|^2 \tag{10}$$

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 in the appendix.

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 $y_0^{\texttt{tgt}} \sim \mathcal{N}(0, I)$ and 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 the appendix.

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 volume, the change of variable formula is applied where 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 $2N + M$ in order to comply with causal masking. Second, generating samples with TNP requires N evaluations as it is an autoregressive model. For our model the number of evaluations 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-clock time comparisons for some of the experiments are reported in Fig. 2 and Fig. 3.

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 both the marginal and conditional consistency properties. This is because in 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 note that also in previous NP models, consistency is not fully guaranteed and holds only in a limited sense. Specifically, while models like CNP, NP and TNP are consistent over marginalization (Eq. 4), they are not consistent over conditioning (Eq. 5). On the other hand models like NDP are consistent over conditioning and not over marginalization. For more discusison on this see App. A.

4 Experiments

Our experiments are conducted on three distinct data domains: synthetic 1D Gaussian processes, image data from EMNIST [10] and CelebA [28], and real-world weather prediction data ERA5 [16].

We implement a single FlowNP model architecture across all experiments in the main paper, adapting only the input dimensions $\texttt{dim}(\mathcal{X})$ and output dimensions $\texttt{dim}(\mathcal{Y})$. We use a transformer with 6 layers of full self attention, 128 hidden dimensions and 4 attention heads. We use sinusoidal encodings for the input x and flow time t with 10 frequencies per dimension, except for the ERA5 experiments where we use 40 frequencies per dimension. We emphasize that the encoding is a function of x rather than the position in the sequence. For likelihood evaluation we use an ODE solver based on the midpoint method with 100 steps implemented by Lipman et al. [26], and for sampling we use the Euler method with 100 steps. See Alg. 1 and Alg. 2 in the appendix for the training and sampling algorithms respectively. The transformer architecture and implementation of our model are based on Nguyen & Grover [32]. All training, inference and sampling are performed with an NVIDIA RTX4090 GPU. The implementation of all experiments and models is available at `https://github.com/danrsm/flowNP`.

In the appendix we provide more examples of larger models trained on the CelebA image data and discuss a modified sampling method that incorporates small noise within the sampling process.

Baselines We compare to the baselines CNP [13], NP [14], ANP [21] and TNP [32] using the implementation in Nguyen & Grover [32] and Lee et al. [23]. In order to make a fair comparison of the main conceptual differences between the methods, we base our model on the transformer architecture in TNP, and reimplement NDP [11] using the same architecture. Therefore, 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 as implemented by the authors, and for the NDP baseline we implement the model with the same transformer used in FlowNP and TNP, omitting the bi-directional attention used in the original NDP. Therefore, NDP differs from our model only by training on unconditional joint distributions, using a linear variance-preserving noise schedule and applying a diffusion loss based on predicting the denoised data rather than the flow velocity. In order to generate conditional samples from NDP we use a guidance-based method, and we evaluate the NDP likelihood by the probability flow ODE with the same ODE solver used for FlowNP. We note that for benchmarks that also appear on the original NDP paper (namely Fixed-Noisy RBF and Fixed-Noisy Matern) our implementation of NDP results in better log-likelihood than the original one.

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. [23] and Bruinsma et al. [8], which are used respectively in the TNP [32] and NDP [11] papers. In the first [23], 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 [8] 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

Table 1: Comparison of log-likelihood computed on the target set for various 1D GP datasets. We provide mean and standard deviation over five runs. FlowNP consistently achieves state-of-the-art performance, outperforming or matching all other models across all datasets.

MODEL	RBF	MATERN-$\frac{5}{2}$	PERIODIC	FIXED-NOISY RBF	FIXED-NOISY MATERN-$\frac{5}{2}$
CNP	0.31 $\pm$ 0.04	0.12 $\pm$ 0.02	-0.63 $\pm$ 0.01	-1.00 $\pm$ 0.02	-1.09 $\pm$ 0.01
NP	0.31 $\pm$ 0.04	0.13 $\pm$ 0.05	-0.61 $\pm$ 0.01	-1.13 $\pm$ 0.02	-1.18 $\pm$ 0.01
ANP	1.10 $\pm$ 0.04	0.85 $\pm$ 0.04	-0.89 $\pm$ 0.04	-0.87 $\pm$ 0.03	-0.98 $\pm$ 0.02
TNP	1.65 $\pm$ 0.02	1.29 $\pm$ 0.02	-0.58 $\pm$ 0.01	0.68 $\pm$ 0.01	0.30 $\pm$ 0.01
NDP	1.20 $\pm$ 0.04	0.94 $\pm$ 0.03	-0.54 $\pm$ 0.01	0.71 $\pm$ 0.01	0.27 $\pm$ 0.02
FLOWNP (OURS)	1.69 $\pm$ 0.02	1.30 $\pm$ 0.02	-0.50 $\pm$ 0.01	0.71 $\pm$ 0.01	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, NDP and FlowNP can generate coherent samples that cover the global uncertainty. In contrast to NDP, FlowNP generates conditional samples directly without needing auxiliary conditioning methods such as guidance. In contrast to TNP that generates the samples point-by-point, FlowNP generates all points in parallel, resulting in faster and smoother sampling.

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 [9] using 200 samples, and for the NDP and FlowNP using the midpoint ODE solver with 100 steps.

Results are summarized in Table 1, where for each dataset we report the mean and standard deviation of five repeated experiments, each using different random seeds and evaluation sets. Note that the performance of baseline models is improved compared to the previously reported results [32, 8] due to our hyperparameter tuning (see App. B). Our model, FlowNP, consistently outperforms all other models while in some cases achieving comparable results to the next best models, TNP and NDP.

Table 2: Target log-likelihood mean $\pm$ 1 standard deviation for the EMNIST and CelebA image datasets and the ERA5 weather dataset. FlowNP consistently outperforms all othe models.

MODEL	EMNIST 0-9	EMNIST 10-46	CELEBA	ERA5
CNP	1.27 ± 0.04	0.73 ± 0.04	2.10 ± 0.01	4.06 ± 0.06
NP	1.17 ± 0.03	0.92 ± 0.02	2.53 ± 0.03	3.35 ± 0.08
ANP	1.35 ± 0.02	1.18 ± 0.01	3.24 ± 0.01	7.76 ± 0.08
TNP	2.08 ± 0.02	1.80 ± 0.05	3.95 ± 0.01	11.32 ± 0.09
NDP	1.58 ± 0.02	1.47 ± 0.01	4.28 ± 0.04	6.76 ± 0.07
FLOWNP (OURS)	2.50 ± 0.01	2.42 ± 0.01	6.37 ± 0.01	12.79 ± 0.03

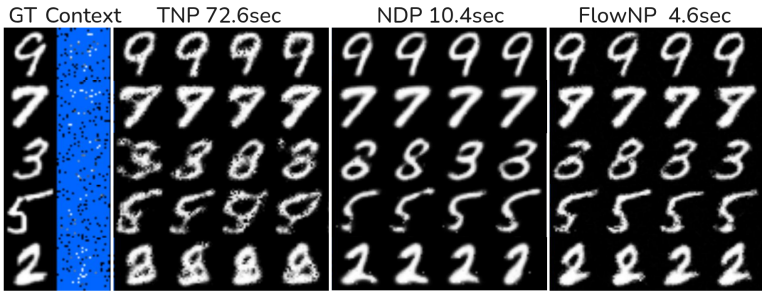

Figure 3: Conditional EMNIST samples generated by TNP, NDP and FlowNP that were trained only on subsets of pixels. For each model we show 4 samples. FlowNP generates sharp and diverse samples faster than other models.

Qualitative results are provided in Fig. 2, showing samples from models trained on the RBF and Matern kernels with fixed parameters. We compare ANP: a latent variable model trained with variational inference; TNP: an autoregressive model; NDP: a diffusion model of joint distributions; 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, NDP 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, NDP and FlowNP generate all points in parallel mostly resulting in smoother samples. An advantage of FlowNP over NDP is that the conditional samples are generated directly from the model outpus rather than relying on a guidance method as used for NDP. For each method we note the wall-clock time for generating single samples, showing that FlowNP runs faster than TNP and NDP. For a visualiztion of the sampling process see Fig. 6 in the appendix.

4.2 Images

We follow with experiments on two image datasets, EMNIST [10] and CelebA [28], where the input $\mathcal{X}$ is 2-dimensional and represents the spatial position in the image, and the output $\mathcal{Y}$ is either a 1D grayscale value or a 3D RGB value of the pixels. The NP models are trained on random subsets of image 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 image data is generated from a discrete representation of the pixels, the likelihood of continuous models can grow arbitrarily (see e.g. App. B. in [24]). Therefore we add small noise with variance 0.01^2 to the data.

Results are provided in Table 2. FlowNP outperforms all other models both for the in-distribution evaluation of EMNIST and CelebA, and for the out-of-distribution evaluation of EMNIST classes 10-46. We show samples generated by TNP, NDP and FlowNP of EMNIST images, using 70 observed pixels as context. Samples from FlowNP are sharp and diverse, and are generated by seven times less network evaluation and using less tokens compared to TNP. FlowNP samples also do not require computing guidance signals for conditioning like NDP. This leads to faster sampling.

In the appendix we show samples of CelebA images (Fig. 7) using a larger FlowNP model based on DiT-B [33] and compare to samples that are generated by adding small noise during the sampling process (App. C).

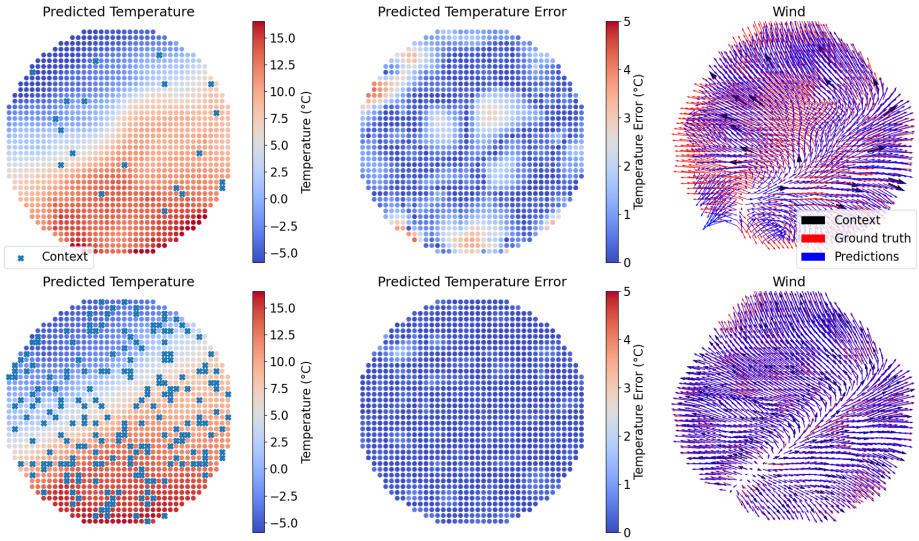

Figure 4: Visualization of temperature and wind prediction using conditional samples generated by FlowNP on two held-out data points from ERA5. The model generates coherent samples based on the context points. As more context points are given (bottom row) predictions become more accurate.

4.3 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 [16]. We follow the setup in Holderrieth et al. [18] and extract data from a circular region with a 520km radius centered around Memphis, USA. Each sample represents a weather snapshot at a single point in time during the winter months in the years between 1980 to 2018 and consists of 1245 grid points of multiple meteorological variables: temperature, pressure, and two wind components (eastward and northward). We divide the years to 34K training samples and 17.5K evaluation samples. Training is performed using the NP setup where $\mathcal{X}$ represents the 2D longitude and latitude, $\mathcal{Y}$ represents the 4D meteorological variables and random subsets out of the 1245 points are used as the context and target sets.

The performance of FlowNP and the baselines are compared in Table 2, where the target set log-likelihood is reported for each model. FlowNP outperforms all other models.

For a qualitative evaluation, Figure 4 provides conditional samples of temperature and wind directions generated by FlowNP on the ERA5 weather data. The figure demonstrates how prediction quality scales with the size of the context set: the top row, using fewer context points, shows predictions with a greater degree of error compared to the bottom row, which is conditioned on a larger context set.

5 Analysis and Discussion

We provide analysis of FlowNP compared to the autoregressive approach of TNP and on different differentiating aspects compared to NDP. For more analysis on running times, different ODE solvers and the number of ODE steps for log-likelihood computation see App. D.

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 [31] and Dutordoir et al. [11] 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. 5 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. 5 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. 4).

Table 3: Target log-likelihood for ablations on three aspects differentiating FlowNP from NDP.

Model	network output	noise schedule	conditioning	RBF	EMNIST
NDP	clean: y_1	linear-vp: $\alpha_t = t, \beta_t = \sqrt{1-t^2}$	unconditional	1.20±0.04	1.58±0.02
diffusion:clean	clean: y_1	linear-vp: $\alpha_t = t, \beta_t = \sqrt{1-t^2}$	conditional	1.38±0.01	1.64±0.01
diffusion:noise	noise: y_0	linear-vp: $\alpha_t = t, \beta_t = \sqrt{1-t^2}$	conditional	1.33±0.04	1.56±0.01
flow:lin-vp	velocity: $y_1 - y_0$	linear-vp: $\alpha_t = t, \beta_t = \sqrt{1-t^2}$	conditional	0.41±0.02	0.48±0.01
flow:poly2	velocity: $y_1 - y_0$	polynomial-2: $\alpha_t = t^2, \beta_t = (1-t)^2$	conditional	1.08±0.02	1.60±0.02
flow:cosine	velocity: $y_1 - y_0$	cosine: $\alpha_t = \sin(\frac{1}{2}\pi t), \beta_t = \cos(\frac{1}{2}\pi t)$	conditional	1.22±0.02	1.80±0.01
flow:joint	velocity: $y_1 - y_0$	cond-ot: $\alpha_t = t, \beta_t = 1 - t$	unconditional	1.73±0.04	2.54±0.03
FlowNP	velocity: $y_1 - y_0$	cond-ot: $\alpha_t = t, \beta_t = 1 - t$	conditional	1.69±0.02	2.50±0.01

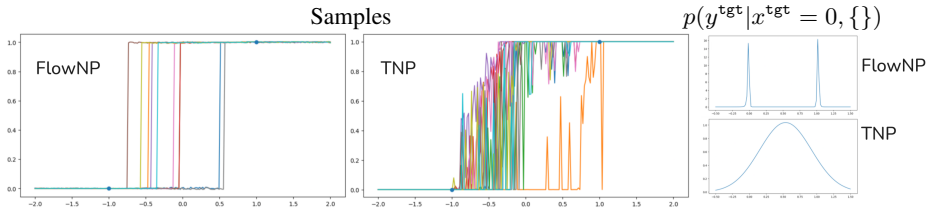

Figure 5: FlowNP vs. TNP for a random step function. FlowNP samples capture the sharp transition occurring in random positions, while TNP cannot model this function as each step in the autoregressive prediction is Gaussian. On the right: the marginal distribution of y predicted for a single point where $x = 0$ further demonstrates this and highlights FlowNP's capacity to capture multimodal distributions.

FlowNP vs. NDP We conduct an ablation study, summarized in Table 3, to investigate three key aspects differentiating FlowNP from the NDP baseline:

1. The output of the network at each step (clean target data y_1, noise y_0, or flow velocity $y_1 - y_0$)
2. The noise schedule used during training $y_t = \alpha_t y_1 + \beta_t y_0$
3. Amortizing context conditioning vs. modeling joint distributions only and computing conditional likelihoods through the joint/context ratio:

$$p\left(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right) = p_\theta\left(y^{\texttt{tgt}+\texttt{ctx}}|x^{\texttt{tgt}+\texttt{ctx}}\right) / p_\theta\left(y^{\texttt{ctx}}|x^{\texttt{ctx}}\right)$$

Our results show that using the flow velocity as the network output and conditional optimal-transport noise schedule (cond-ot) yields superior performance across the models. We note that training FlowNP only on joint distributions (flow:joint) and using it to compute conditional likelihood through the joint/context ratio, achieves a slightly higher likelihood than a FlowNP model that is trained conditioned on contexts directly. This is perhaps expected as a model trained unconditionally dedicates its entire capacity to modeling the joint data distribution, a less demanding task than amortizing the conditioning on all possible context sets. However, this slightly higher likelihood comes at a cost of requiring auxiliary methods to generate conditional samples. In contrast, conditional training of FlowNP provides slightly lower maximum log-likelihood but is inherently capable of direct, auxiliary-free conditional sampling.

Conclusion 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. Compared to NDP, we find that using a flow matching objective is favorable over diffusion and that modeling conditionals enables direct computation both for conditional likelihood and conditional sampling. Compared to TNP, FlowNP can efficiently capture non-Gaussian and multimodal distributions and generate samples for a set of points in parallel.

Limitations The main limitation of FlowNP is the iterative sampling and likelihood computation. One possible approach to improve this could be based on methods like shortcut models [12] which explicitly train the model to require less ODE solver steps.

References

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

[2] 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.

[3] 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*, 2024.

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

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

[6] Lorenzo Bonito, James Requeima, Aliaksandra Shysheya, and Richard E. Turner. Diffusion-Augmented Neural Processes. 2023-11-16. doi: 10.48550/arXiv.2311.09848. URL `http://arxiv.org/abs/2311.09848`.

[7] Wessel Bruinsma, Stratis Markou, James Requeima, Andrew Y. K. Foong, Tom Andersson, Anna Vaughan, Anthony Buonomo, Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=OAsXFPBfTBh`.

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

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

[10] 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.

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

[12] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models, 2025. URL `https://arxiv.org/abs/2410.12557`.

[13] 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, 2018.

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

[15] 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.

[16] 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.

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

[18] 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.

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

[20] Gavin Kerrigan, Giosue Migliorini, and Padhraic Smyth. Functional flow matching. In *The 27th International Conference on AI and Statistics (AISTATS)*, 2024.

[21] 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.

[22] 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.

[23] 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.

[24] Marten Lienen, Marcel Kollovieh, and Stephan Günnemann. Generative modeling with bayesian sample inference. 2025. URL `https://arxiv.org/abs/2502.07580`.

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

[26] 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`.

[27] 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.

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

[29] 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.

[30] 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`.

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

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

[33] William Peebles and Saining Xie. Scalable diffusion models with transformers.

[34] 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`.

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

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

[37] 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.

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

[39] 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.

[40] 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.

[41] 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`.

[42] Bernt Øksendal. Stochastic Differential Equations. 2003.

A Consistency of NP 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. This implies that consistency over marginalization (Eq. 4) holds. However, the model structure does not guarantee consistency over conditioning (Eq. 5) because computing the model conditioned on different contexts 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 order, the joint distribution of the target 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 conditioning is that it only models joint distributions and therefore computes conditionals using the conditional definition:

$$p\left(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\}\right) = p_\theta\left(y^{\texttt{tgt}+\texttt{ctx}}|x^{\texttt{tgt}+\texttt{ctx}}\right) / p_\theta\left(y^{\texttt{ctx}}|x^{\texttt{ctx}}\right)$$

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 algorithms for training and sampling are provided in Alg. 1 and Alg. 2.

Fig. 6 provides a demonstration of the sampling process for 1D GP data by visualizing the transformation from random noise to samples from the conditional distribution of $p(y^{\texttt{tgt}}|x^{\texttt{tgt}}, \{x^{\texttt{ctx}}, y^{\texttt{ctx}}\})$.

The hyperparameters of the baselines are tuned to improve their performance. Namely, we find that for CNP and TNP no bound on the output variance is required, while for the models trained by variational inference, NP and ANP, we set the output variance bound to 0.05^2 and use 8 importance samples for training. The implementation in `https://github.com/danrsm/flowNP` contains the configurations of all the models.

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
  $y_0^{\texttt{tgt}} \sim \mathcal{N}(0, I)$ — Sample noise
  $y_t^{\texttt{tgt}} \leftarrow t y^{\texttt{tgt}} + (1-t) y_0^{\texttt{tgt}}$
  $\texttt{tokens}^{\texttt{ctx}} = \{\texttt{embed}^\theta([x^{\texttt{ctx}},\ 1,\ y^{\texttt{ctx}}])\}$ — 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

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
$y^{\texttt{tgt}} \sim \mathcal{N}(0, I)$ — 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
  $\hat{u}_t \leftarrow u^{\theta}([\texttt{tokens}^{\texttt{ctx}}, \texttt{tokens}^{\texttt{tgt}}])$
  $y^{\texttt{tgt}} \leftarrow y^{\texttt{tgt}} + \delta \hat{u}_t$
end for
output: predicted sample $y^{\texttt{tgt}}$.

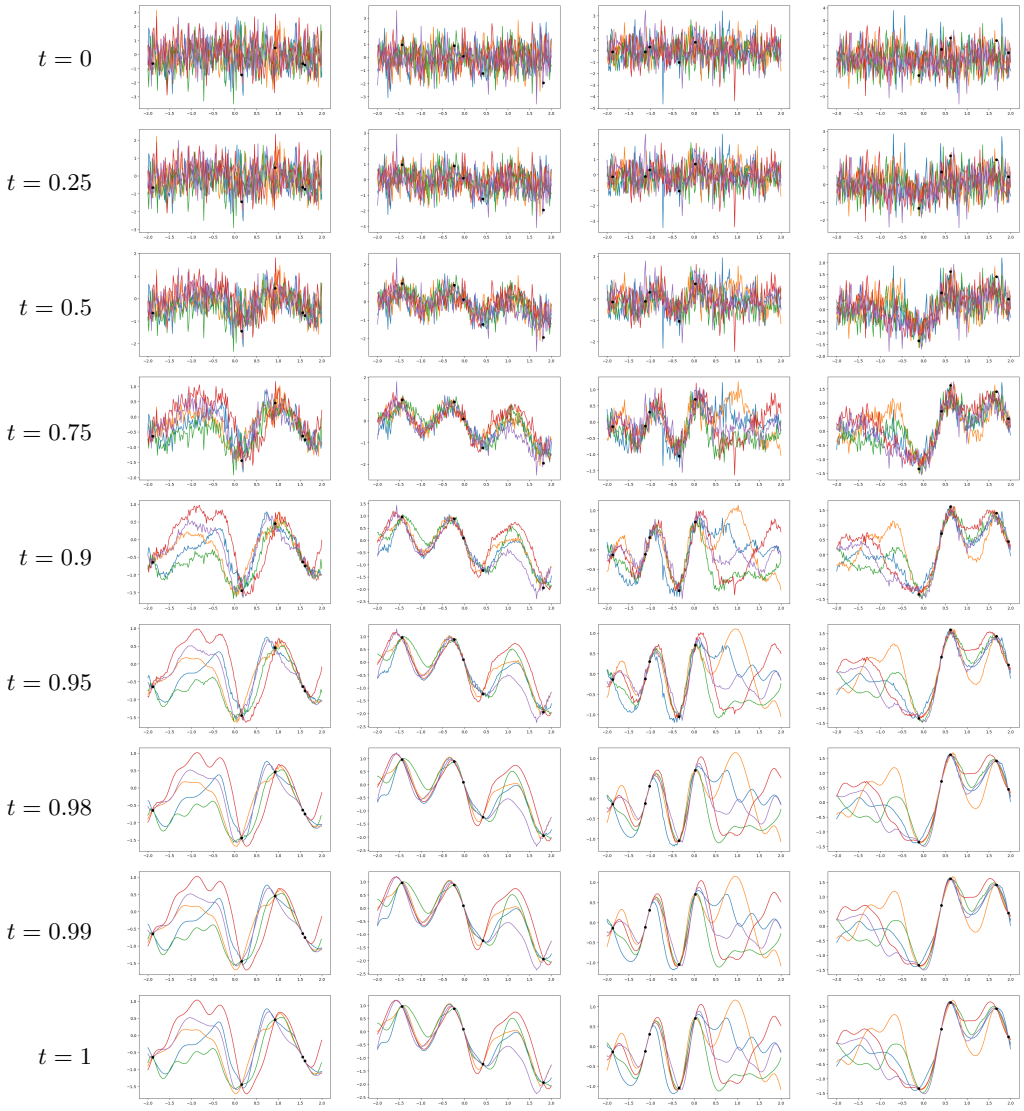

Figure 6: A visualization of the sampling process, showing the intermediate values $y_t^{\texttt{tgt}}$ for different time steps. Each column is conditioned on a different context and each color represents a different random sample generated by FlowNP.

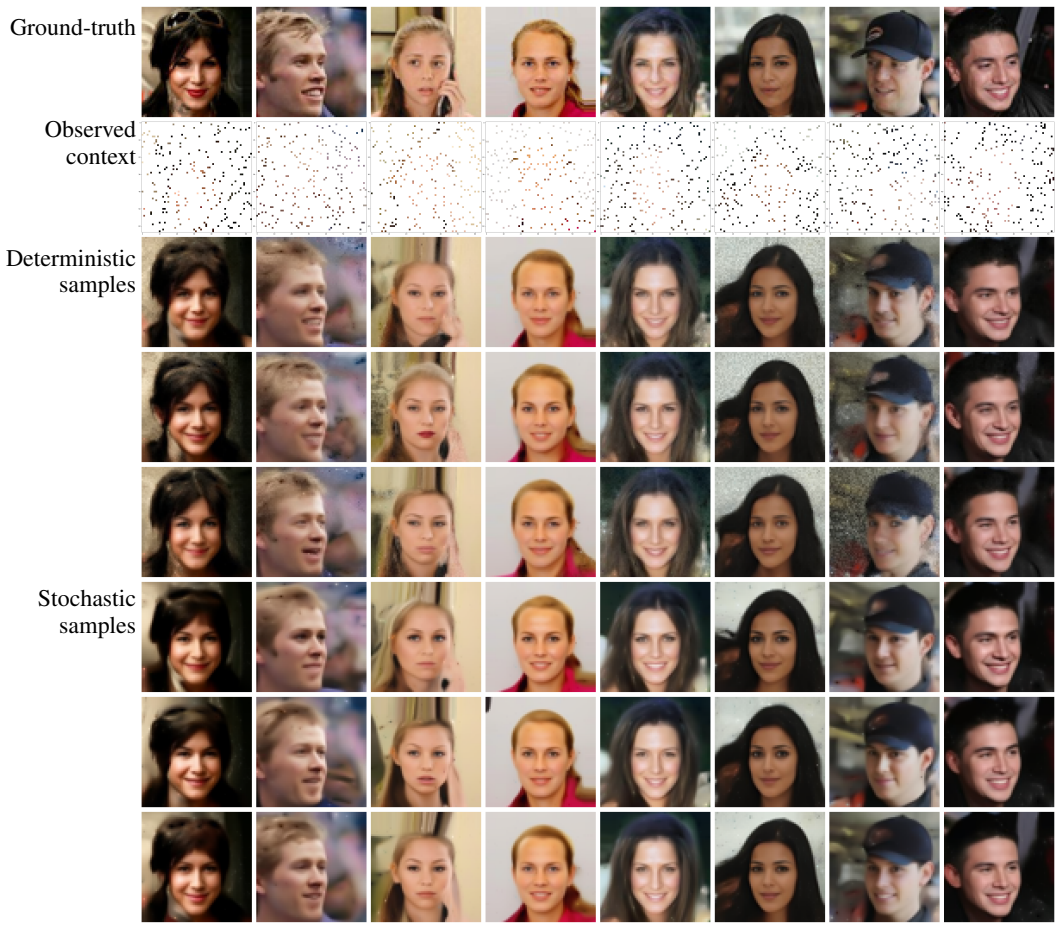

Figure 7: Generating conditional samples of CelebA data using the FlowNP model trained only on subsets of pixels.

C CelebA Images

We train a larger FlowNP model using the DiT-B architecture [33] on CelebA images with a resolution of 64×64. The model consists of 12 transformer layers with 768 hidden dimensions and 12 attention heads. We train this model using the same approach as in the main paper, by sampling random subsets of context pixels and target pixels every time an image is drawn. We the use this model to generate samples conditioned on a context of 5% observed pixels as shown in Fig. 7.

In some cases, we find that adding small levels of noise and scaling the predicted velocity during the sampling process results in samples that look more coherent and less noisy. We implement this by replacing the update of $y^{\texttt{tgt}}$ in each step of Alg. 2 to the following:

$$y^{\texttt{tgt}} \leftarrow y^{\texttt{tgt}} + \delta\alpha_t\hat{u}_t + \delta\sigma_t\nu \tag{11}$$

Where $\nu \sim \mathcal{N}(0, I)$ is random noise, α_t are values greater than 1 and σ_t are small values. In our implementation we use $\alpha_t = 1 + t(1-t)$ and $\sigma_t = 0.2t^2(1-t)^2$.

A comparison between generated samples using Alg. 2 (Deterministic samples) and samples generated with the noise modification (Stochastic samples) is provided in Fig. 7. While the results are very similar, it can be seen that the deterministic samples sometimes appear noisier.

This approach did not make any difference for the GP data which suggests that it is effective when the capacity of the model is limited relative to the complexity of the distribution.

ODE-solver	log-likelihood
Euler	1.7941
Midpoint	1.6890
RK4	1.6856
Dopri5	1.6857

Table 4: Comparing different ODE solvers on the RBF GP data.

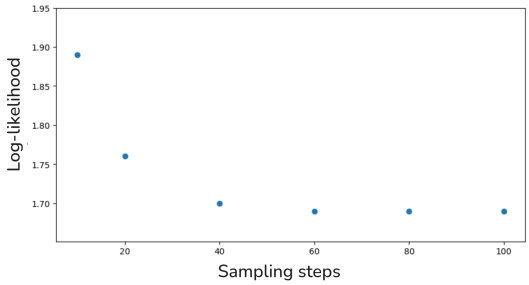

Figure 8: Log-likelihood as a function of number of ODE steps, computed using the midpoint ODE solver on the RBF GP data.

D Analysis

ODE solver We compute the likelihood using several different ODE solvers for the FlowNP model on the RBF experiment. The results, presented in Table 4, show that except for the Euler method which overestimates the likelihood, all other solvers compute very similar values.

Number of ODE steps We run an ablation on sampling and log-likelihood evaluation for different numbers of steps in the ODE solver using the midpoint method. For GP-RBF data, the mean of 5 random seeds with different number of steps is presented in Fig. 8. The std of for all is 0.014. This shows that the evaluation plateaus after T=60 steps. We observe similar behavior for sample quality.

Running time Comparing the sampling time between FlowNP, NDP 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 0.2sec for FlowNP, 0.5sec for NDP and 0.8sec for TNP. The time to generate 1 EMNIST sample with 748 target points is 4.6sec for FlowNP, 10.4sec for NDP and 72.6sec for TNP.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All our claims are either evident from the the model construction or empirically evaluated in the experiments (performance and running time).

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations throughout the paper and in a dedicated paragraph in the dicussion.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not contain theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We specify all details required for reproducing the results. Most of our experiments are based on standard open source benchmarks and we also provide code for the models.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code for the models that can be used with open sourced code for the experiments. Upon publication we will open source our full repository.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We specify all details of our experiments. Additionally most of them are based on open sourced experiments.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide error bars for our main experiments computed from 5 runs with different training random seeds and evaluation sets.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide details on the resources used and execution time.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper deals with modeling distribution of abstract functions and as such does not raise ethical concerns.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: The paper deals with modeling distributions of abstract functions and as such does not have societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper deals with modeling distributions of abstract functions and as such does not have risks of misuse.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All prior work and code used is properly cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We specify all details of our model and provide code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.