GRAPH TRANSFORMER NEURAL PROCESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Processes (NPs) are a powerful class of model for forming predictive distributions. Rather than use an assumed prior over functions to form uncertainties—as is done with Gaussian Processes—NPs can meta-learn uncertainties for unseen tasks; however, in practice meta-learning uncertainties may require a great deal of data. To address this, we propose representing the inputs to the model as a graph and labelling the edges of the graph with similarities or differences between points in the context and target sets, allowing for invariant representations. We propose an architecture that can operate over such a graph and experimentally show that it achieves strong performance even when there is limited data available. We then apply our model on three real world regression tasks to demonstrate the advantages of representing the data as a graph.

1 INTRODUCTION

When tasked with a decision making problem in the real world, one common approach is to use collected data to learn a surrogate model that can inform action selection. This strategy is not without its challenges, however, there is uncertainty in the model's predictions due to inherent noise in the real-world system or a lack of data. Forming accurate predictive uncertainties is therefore essential, and indeed, intelligent decision making algorithms take advantage of such uncertainties (Snoek et al., 2012; Chua et al., 2018; Shyam et al., 2019; Mehta et al., 2022; Li et al., 2022a).

In this work, we learn these uncertainties using a type of model known as a Neural Process (NP) (Garnelo et al., 2018b). These models are neural networks which take a context set of previous observations as input and predict a distribution for a specified target set. Rather than assuming a prior over possible functions—as is done with a Gaussian Process (GP)—we instead use collected data to meta-learn (Hospedales et al., 2021) these uncertainties. This model class is powerful and has been used for many real-world applications such as neuroscience (Pakman et al., 2020; Cotton et al., 2020), astronomy (Čvorović-Hajdinjak et al., 2022; Park & Choi, 2021; Pondaven et al., 2022), and robotics (Chen et al., 2022; Li et al., 2022b; Yildirim & Ugur, 2022).

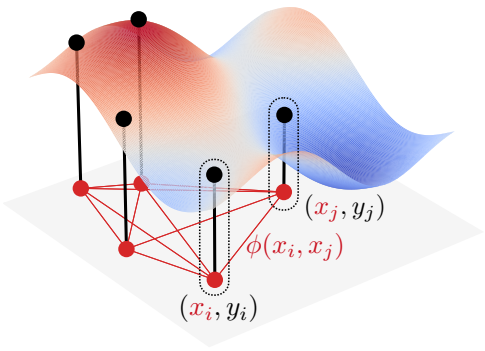

Figure 1: **A Visualization of the Data as a Graph**. The diagram depicts how one can construct a graph (in red) from observations from an unknown function (points in black) using a distance or similarity metric ϕ.

Most NP models operate directly on the raw data representations; however, this does not take advantage of invariances that may be in the data, resulting in the sample inefficiency. An important exception to this are Convolutional NPs (Gordon et al., 2019). These models enforce translational equivariance through so-called "convolutional deep sets". However, these models require forming a grid over the support of the data, which becomes infeasible with high dimensional input spaces.

To remedy this, we advocate for representing the condition and target set as a graph. Depending on the choice of edge labels, this representation can be invariant to translations or rotations, and at the same time, it can be extended to higher dimensional settings. We start this work by first describing how to formulate the context and target set into a graph. We then suggest an architecture that can operate over such a graph, which is a graph transformer architecture (Shi et al., 2020). In addition, we introduce a second novel architecture inspired by Gaussian elimination, but leave the description and analyses in appendix. Through synthetic experiments we show that our proposed model not only achieve strong results, but is dramatically more sample efficient. We then show how these advantages translate to real data with three experiments on a nuclear fusion application, a weather prediction application, and on a cheminformatics application.

2 METHOD

2.1 PRELIMINARIES

Let $\mathcal{X} \subset \mathbb{R}^{D_X}$ and $\mathcal{Y} \subset \mathbb{R}$ be input and output spaces, and let $f : \mathcal{X} \to \Delta(\mathcal{Y})$ be a random function, where $f \sim \mathcal{P}$ and $\Delta(\mathcal{Y})$ is the set of distributions over $\mathcal{Y}$. Sometimes $\mathcal{P}$ is known or assumed (e.g one could use a Gaussian Process prior (Williams & Rasmussen, 2005)); however, it may be the case that one only has access to data produced by the random function. Let $\mathcal{D}$ be such a dataset where

$$\mathcal{D} = \{\{(x_j, y_j \sim f_i(x_j)\}_{j=1}^{N_i}\}_{i=1}^{M}$$

and where $f_i \sim \mathcal{P}$. In other words, $\mathcal{D}$ is a dataset composed of M different function samples, and for the i^{th} sample, there are N_i function evaluations.

This work focuses on learning a deep network that approximates the random function f using the dataset $\mathcal{D}$. Let p_θ be such a network where θ is the set of parameters for the network. Moreover, let p_θ be a so-called *Neural Process* (NP) (Garnelo et al., 2018b). The hallmark of these models is that they can produce a predictive distribution given an input x and a context set of previous observations.

Concretely, let $\mathbf{x} \in \mathcal{X}^N$ be a collection of N "points" in $\mathcal{X}$. Further, let the first $C \in \mathbb{Z}^+$ of these points, $\mathbf{x}_{1:C}$, belong to the context set, and let the remainder of the points, $\mathbf{x}_{C+1:N}$, be the target set. Along with this, the model also has access to the observations corresponding to the context set, $\mathbf{y}_{1:C} \in \mathcal{Y}^C$. The goal is for the model to predict a distribution for the target observations, $\mathbf{y}_{C+1:N}$, which usually takes the form of a multivariate normal distribution where target points are independent from each other. In summary, the model produces $\mu, \sigma = p_\theta(\mathbf{x}, \mathbf{y}_{1:C})$, where $\mu, \sigma \in \mathbb{R}^{N-C}$ are the mean and standard deviations of the predicted normal distribution. Lastly, for notational convenience, we let $p(\cdot|\mathbf{x}, \mathbf{y}_{1:C})$ be the pdf of the predicted distribution.

For most NPs (including the ones we will propose), the training objective is to maximize the log likelihood. That is, the objective function is

$$J(\theta) = \mathbb{E}_{C,\mathbf{x},\mathbf{y}}\left[\log p_\theta(\mathbf{y}_{C+1:N}|\mathbf{x}, \mathbf{y}_{1:C})\right] \tag{1}$$

where context size, $\mathbf{x}$, and $\mathbf{y}$ all have underlying distribution. In the rest of this subsection, we give a brief overview of different members of the NP family that are most salient for our work. For a more thorough review, see Jha et al. (2022).

Latent, Conditional, and Attentive Neural Processes Latent and conditional NPs (LNP[1] (Garnelo et al., 2018b) and CNP (Garnelo et al., 2018a), respectively) were the first proposed NP models. Both of these architectures use a `DeepSet` (Zaheer et al., 2017) over the context set to create an encoding which captures the relevant information of the context set. This encoding is then used, along with $\mathbf{x}_{C+1:N}$, to predict independent normal distributions for the targets. The key difference

[1] In the original paper, this model is simply called a "Neural Process". Following other works, we instead call it a Latent Neural Process to differentiate it from the broad class of models.

between the two is that the LNP assumes a latent distribution over the encoding and therefore uses a variational approximation. Kim et al. (2019) improves on these NP architectures by introducing attention (Vaswani et al., 2017) into the encoding scheme. In particular, they replace the `DeepSet` with self-attention and add cross-attention between the context and target sets.

Convolutional Neural Processes All of the NPs discussed up until now work directly on x representations. This is not the case, however, with the Convolutional Conditional NP (ConvCNP) (Gordon et al., 2019). Instead of a `DeepSet` architecture, the ConvCNP uses a `ConvDeepSet`, which operates over a grid covering $\mathbf{x}$. By operating over this grid rather than x representations, the ConvCNP is translation equivariant and more sample efficient. While operating over a grid is natural for image based domains, it has challenges when operating in the regression setting, especially when dealing with high dimensional $\mathcal{X}$. While there have been follow up works introducing new variants (Wang et al., 2021; Kawano et al., 2021), these methods are still bound to a grid.

Autoregressive Neural Processes and Transformers Most NPs assume conditional independence over the target set given the context set. That is, the output of these networks are the mean and standard deviation parameterizing an independent normal distribution for each $\mathbf{y}_{C+1:N}$. While there are works which predict multivariate normal distributions over the target set (Bruinsma et al., 2021; Markou et al., 2022; Nguyen & Grover, 2022), recent work has shown that models that are trained assuming conditional independence can predict the joint distribution in an auto-regressive fashion to produce competitive results (Bruinsma et al., 2023; Nguyen & Grover, 2022). Concretely, one can form the joint distribution using the chain rule:

$$p_\theta(\mathbf{y}_{C+1:N}|\mathbf{x}, \mathbf{y}_{1:C}) = \prod_{i=C}^{N-1} p_\theta(\mathbf{y}_{i+1}|\mathbf{x}_{1:i+1}, \mathbf{y}_{1:i})$$

where $\mathbf{y}_i$ is the i^{th} observation in $\mathbf{y}$. As noted by Bruinsma et al. (2023), using these NPs in "autoregressive mode" results in highly expressive model at the cost of coherence. In this work, we will default to the autoregressive mode for evaluation.

An architecture of particular interest for this work is the transformer (Vaswani et al., 2017). Previous works have shown that transformers are excellent at meta-learning for reinforcement learning (Melo, 2022), drug discovery (Chen & Bajorath, 2023), and—most related to this work—for approximating Bayesian inference (Müller et al., 2021). Nguyen & Grover (2022) use the transformer architecture to create the Transformer NP (TNP). This model frames the context and target sets as sequences, and then uses a transformer architecture to directly predict normal distributions for each target. Because our proposed architectures are variants of the transformer, we briefly review self-attention.

Consider N embeddings $z_1, z_2, \ldots, z_N \in \mathbb{R}^d$. We use d to denote the size of embeddings throughout this work. Fix an $i \in [1, N]$. To start, three linear projections of z_i are made. They are known as the key, query, and value vectors and are denoted by $k_i, q_i, v_i \in \mathbb{R}^d$, respectively. We denote the attention operator for the ith embedding as $\texttt{Attention}_i$ and define it as the following:

$$\texttt{Attention}_i(z_1, \ldots, z_N) = \sum_{n=1}^{N} \alpha_{i,n} v_n$$

$$\text{where} \quad \alpha_{i,j} = \frac{\exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_j\rangle\right)}{\sum_{n=1}^{N} \exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_n\rangle\right)}$$

In practice, transformers use multi-headed self-attention, repeating the mechanism H times, concatenating the results, and applying a final linear projection.

2.2 Framing the Data as a Graph

In this subsection, we argue for representing the context and target sets as a graph. We define a graph as $\mathcal{G} := (V, E)$, where V is the vertex set and E is the set of edges between the vertices. Here, vertex $\nu_i \in V$, where $i \in \{1, \ldots, N\}$, corresponds to $\mathbf{x}_i$. We will often refer to $\{\nu_i\}_{i=1}^{C}$ as context vertices and $\{\nu_i\}_{i=C+1}^{N}$ as target vertices. In a slight abuse of notation, we consider E to be a tensor in $\mathbb{R}^{N \times N \times d}$, where $E_{i,j} \in \mathbb{R}^d$ is the label for the edge between ν_i and ν_j, i is the row of the tensor, and j is the column of the tensor. A diagram of such a graph can be seen in Figure 1.

Although the exact formation of the graph is architecture dependent, the key idea is that $E_{i,j}$ should capture information from $\phi(\mathbf{x}_i, \mathbf{x}_j)$, where ϕ can be a distance metric or a kernel function. This representation is powerful because one can encode different properties depending on the choice of ϕ. For example, when $\phi(x, x') = \|x - x'\|_2$, the representation is both translation and rotation invariant.

We propose an NP architecture operating on such graphs. Our main model is a graph transformer (Shi et al., 2020) that evolves vertex features. A second, more expressive model focusing on evolving edge features (GEENP) is described in Appendix A and B.2; however, it underperforms the main model unless large amounts of data are available and is significantly more computationally expensive, especially for large N.

Graph Transformer Neural Process To start, we describe a model that takes advantage of the graph transformer architecture (Shi et al., 2020). Accordingly, we refer to this model as the Graph Transformer Neural Process (GTNP). To start, the graph is initialized by setting $E_{i,j} = f_\theta^E(\phi(\mathbf{x}_i, \mathbf{x}_j))$ and initializing vertex features to $\nu_i^0 = f_\theta^V(\mathbf{y}_i)$ when $i \leq C$ and $\nu_i^0 = f_\theta^V(\mathbf{0})$ otherwise; here, the super-script on ν_i indicates how many graph transformer block transformations the vertex encoding has undergone. Both f_θ^V and f_θ^E are fully-connected neural networks.

After this intialization, the vertices of the graph are updated via a number of graph transformer blocks. These blocks closely match the structure of those in the GPT-2 model Radford et al. (2019) (where each instead of token embeddings we have vertex embeddings); however, attention is replaced with graph attention in order to fold in edge information. For each edge, $E_{i,j}$, key and value linear projections are made, which we denote as $\lambda_{i,j}, \beta_{i,j} \in \mathbb{R}^d$, respectively. The graph attention operator for vertex i is defined as

$$\texttt{GraphAttention}_i(V, E) = \sum_{n=1}^{N} \alpha_{i,n} \left(v_n + \beta_{i,n}\right)$$

$$\text{where} \quad \alpha_{i,j} = \frac{\exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_j + \lambda_{i,j}\rangle\right)}{\sum_{n=1}^{N} \exp\left(\frac{1}{\sqrt{d}}\langle q_i, k_n + \lambda_{i,n}\rangle\right)}$$

In practice we use a multi-headed version of graph attention. Additionally, we also use a masking scheme that prevents all vertices from attending to a target vertex. After L blocks, the vertices are used for predicting the mean and standard deviation of a normal, i.e. $\mu_i, \sigma_i = g_\theta\left(\nu_i^L\right)$, where g_θ is another fully-connected network. See Figure 2 for a diagram of this architecture. In practice, we use the masking trick in Nguyen & Grover (2022) to train the GTNP autoregressively (i.e. akin to TNP-A in Nguyen & Grover (2022)). We detail this as well as a thorough description of the architecture in Appendix B.1.

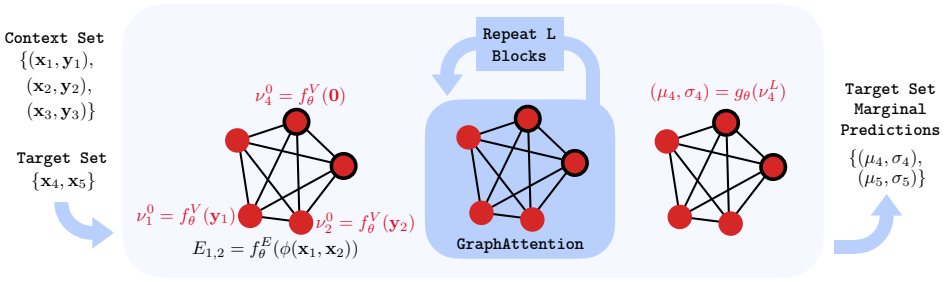

Figure 2: **An Illustration of the GTNP Architecture.** Data in the context and target set are first formed into a graph. The vertex features of the graph are then transformed via a graph transformer network (Shi et al., 2020). Lastly, mean and standard deviations are predicted using the final vertex features.

3 Experiments

In this section, we experimentally answer the following questions:

- How expressive is GTNP compared to other NPs? How robust are each of these NPs to translational shifts at test time? (Section 3.1)

- How sample efficient is GTNP compared to other NPs? (Section 3.2)
- How does GTNP perform on real world datasets? (Sections 3.3 and 3.5)

For comparison, we use the attentive versions of LNP and CNP (we refer to these as AttnLNP and AttnCNP, respectively), ConvCNP, and the autoregressive version of TNP (i.e. TNP-A). The implementation for AttnLNP, AttnCNP, and TNP were taken from the official TNP GitHub repository[2], and we use the same hyperparameters as in Nguyen & Grover (2022) for each of these models. For ConvCNP, we use a repository made by one of the authors of Gordon et al. (2019) [3]. We use the same parameters as their regression experiments and choose the "XL" version of their architecture, which leverages a U-Net (Ronneberger et al., 2015). Unfortunately, we were unable to find any implementations extending the "off-the-grid" version of ConvCNP for any dimensions higher than 1D. While higher dimensional versions of the algorithm are theoretically possible, forming a grid with high enough fidelity quickly becomes computationally taxing as the dimensionality increases.

For our proposed graph NPs, we try to match the hyperparameters of TNP as closely as possible. In particular, we use 6 blocks, $H = 4$ attention heads, and an embedding size of $d = 64$ for each architecture. We use the following for ϕ,

$$\phi(x, x') = W \bigoplus_{h=1}^{H} \|x \otimes w_h - x' \otimes w_h\|_2$$

where $\oplus$ representation concatenation, $\otimes$ represents the Hadamard product, and $W \in \mathbb{R}^{d \times H}$ and $w_h \in \mathbb{R}^{D_x}$ are learnable parameters. Note that this ϕ results in a translation invariant representation.

Table 1: **Infinite Dataset Results**. The reported metric average joint log likelihood over the test set. We use five seeds to report the average and the standard error.

Dimension	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP
1D	12.92 ± 0.03	12.62 ± 0.03	20.64 ± 0.08	21.34 ± 0.02	$\mathbf{21.80 \pm 0.01}$
2D	-4.43 ± 0.02	-4.69 ± 0.02	—	-4.16 ± 0.55	$\mathbf{-2.99 \pm 0.01}$
4D	-9.37 ± 0.01	-9.40 ± 0.00	—	-9.86 ± 0.00	$\mathbf{-9.25 \pm 0.00}$

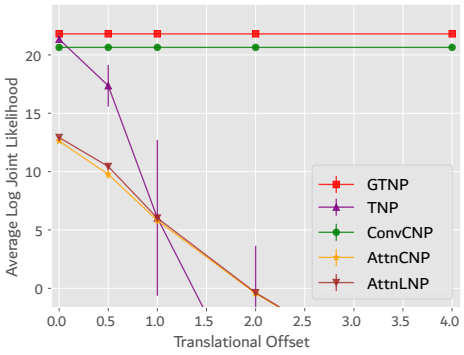

Figure 3: **Log Likelihood vs. Translational Offset in the 1D Infinite Data Case.** The y-axis shows the average log joint likelihood over the test set and the x-axis shows the amount of translational shift on the inputs.

3.1 Synthetic Experiments in the Infinite Data Regime

We start by repeating the experiment first done in Garnelo et al. (2018a) in which the NP is trained from data generated from a hierarchical, 1D GP. The GP has an RBF kernel parameterized by length-scale $\ell \sim \mathcal{U}(0.1, 0.6)$ and scale $\sigma_\kappa \sim \mathcal{U}(0.1, 1.0)$. Each point in $\mathbf{x}$ is drawn independently from $\mathcal{U}(-2.0, 2.0)$, and we draw N and C uniformly from $[6, 50]$ and $[3, 47]$, respectively. We refer to this regime as the "infinite data regime" because there is no fixed dataset $\mathcal{D}$. Instead, because $\mathcal{P}$ is

[2] https://github.com/tung-nd/TNP-pytorch/tree/master
[3] https://github.com/cambridge-mlg/convcnp

Table 2: **Finite Dataset Results**. The metric is a standardized score where a score of 100 means that the NP has exactly reproduced the GP, and a score of 0 means that the NP produces a worse log likelihood than simply predicting the prior (see Appendix C.4 for the exact definition of this metric). We use five seeds to report the average and the standard error.

Dataset	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP
1D 1K	4.88 ± 1.74	-2.72 ± 4.20	34.26 ± 5.38	-2.62 ± 1.86	$\mathbf{61.90 \pm 1.92}$
1D 10K	66.83 ± 0.75	63.94 ± 1.06	65.48 ± 4.52	82.01 ± 0.58	$\mathbf{83.73 \pm 0.59}$
1D 100K	71.76 ± 0.74	71.92 ± 0.68	93.04 ± 0.35	92.95 ± 0.38	$\mathbf{93.86 \pm 0.23}$
2D 1K	-2.56 ± 1.48	-2.80 ± 1.79	—	-8.78 ± 7.38	$\mathbf{56.94 \pm 0.64}$
2D 10K	61.18 ± 0.96	57.81 ± 2.24	—	63.92 ± 0.42	$\mathbf{73.49 \pm 1.06}$
2D 100K	66.04 ± 2.10	68.33 ± 1.07	—	82.33 ± 0.35	$\mathbf{82.92 \pm 0.34}$
4D 1K	-31.04 ± 15.97	-41.94 ± 19.01	—	-9.24 ± 1.22	$\mathbf{-6.10 \pm 2.91}$
4D 10K	24.74 ± 7.62	-43.79 ± 29.08	—	-2.59 ± 2.31	$\mathbf{78.90 \pm 1.40}$
4D 100K	59.83 ± 2.40	51.47 ± 4.58	—	71.30 ± 1.54	$\mathbf{91.08 \pm 0.49}$
Average	35.74	24.69	—	41.03	**68.53**

known exactly, new $\mathbf{x}$ and $\mathbf{y}$ are sampled every batch. This is a luxury that is usually absent from real-world cases with fixed datasets.

In addition to this 1 dimensional GP, we also consider 2 and 4 dimensional GPs. We keep $\mathcal{X} = [-2.0, 2.0]^{D_x}$, but we adjust the range of lengthscales to be $(0.1\sqrt{D_x}, 0.6, \sqrt{D_x})$. We use anisotropic kernels and sample lengthscales independently for each dimension. As an evaluation metric, we compute the joint log likelihood using the NPs in "autoregressive mode", see Section 2.1 and Bruinsma et al. (2023)).

Table 1 shows the results averaged over five random seeds. GTNP achieves better results than the other baselines, likely because it is translation invariant while not being bound to a grid. We also show how the test metric changes as a translational shift is applied to $\mathbf{x}$ in Figure 3. This figure helps demonstrate that our architectures and ConvCNP are robust to translational shifts while the other baselines are highly susceptible.

3.2 Synthetic Experiments in the Finite Data Regime

We now consider the "finite" data regime where there is a fixed dataset $\mathcal{D}$ to learn from. To better understand the performance of the NPs, we simplify the problem by making the data generating process a GP with fixed parameters. In particular, we set the lengthscale for each dimension to the midpoint of the ranges in Section 3.1, we set σ_k to 1.0, and we fix $N = 50$. At test time, we compute the joint likelihood for every possible context size and average the results. We also standardize this metric so that 0.0 corresponds to predicting a standard normal for each point, and 100.0 corresponds to matching the performance of the true underlying GP. More details can be found in Appendix C.4.

Table 2 shows the results for datasets generated from 1D, 2D, and 4D GPs. In addition, we consider dataset sizes of 1K, 10K, and 100K points (i.e. the 1K dataset has $1,000 \div 50 = 20$ GP function samples). In these experiments, it is clear that GTNP shines in low data regimes. Figure 4 shows contour plots for the standard deviation predictions in 2D for a dataset size of 1K. Under the limited data regime, it is clear that TNP cannot learn any useful uncertainty. GTNP produces far better uncertainties than any other NP in this regime. Overall, these results suggest that GTNP is the best choice in any data size but especially when there is a limited amount of data.

3.3 Nuclear Fusion Application

We now turn to real-world data, applying our models to a key fusion energy application: tokamaks, devices that magnetically confine plasma in a toroidal chamber to achieve nuclear fusion. A major challenge in this domain is predicting plasma evolution, a task well-suited to machine learning, which can leverage historical data to learn dynamics models (Abbate et al., 2021; Char et al., 2023; Jalalvand et al., 2021; Kolemen et al., 2023; Wang et al., 2023).

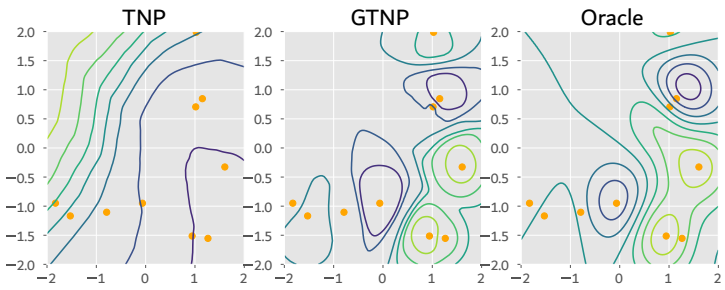

Figure 4: **Contour of the Predicted Standard Deviation for the 2D 1K Task**. The far right plot titled "Oracle" shows the standard deviation of the posterior GP that generated the data. The orange points show points in the context set.

Prediction is difficult due to the complex dynamics and variability of conditions across experiments (each typically comprising several "shots"). Assuming similar dynamics within an experiment, we aim to estimate uncertainties in these models by meta-learning from previous experiments.

We predict two plasma properties: rotation and β_N, the normalized plasma-to-magnetic pressure ratio. Let $\mathcal{X} \subset \mathbb{R}^{10}$ represent the plasma state and actuator settings. The transition function $T : \mathcal{X} \to \mathbb{R}^2$ outputs β_N and rotation 25 ms ahead, while $\hat{T}$ is the learned approximation. We define the residual $f(x) = T(x) - \hat{T}(x)$, and adjust NPs to output an uncorrelated multivariate normal.

Our dataset contains 3,226 experimental groupings and 736,900 transitions from the DIII-D tokamak, split 80/10/10 for training $\hat{T}$, training NPs, and testing. We set $N = 50$ and use the most recent points as context, drawn from the current or previous shots. $\hat{T}$ follows the "Probabilistic Neural Network" (PNN) architecture from Chua et al. (2018), producing a normal distribution; we use its mean for predictions and its variance as a baseline, alongside a fixed Gaussian baseline.

Table 3 reports the joint log-likelihood averaged over context sizes:

$$\frac{1}{(N-1)M} \sum_{m=1}^{M} \sum_{c=1}^{N-1} \frac{1}{N-c} \log p_\theta(\mathbf{y}_{c+1:N}^{(m)} \mid \mathbf{x}^{(m)}, \mathbf{y}_{1:c}^{(m)}) \tag{2}$$

Most NPs do not outperform the unconditioned PNN, but TNP and GTNP yield superior results, with GTNP achieving the highest score. Interestingly, although our choice of ϕ assumes translation invariance in f—likely false—the sample efficiency gained appears to outweigh the resulting bias.

Table 3: **Average Joint Log Likelihood for Nuclear Fusion Task**. We use five seeds to report the average and the standard error. The method "Fixed Normal" is the result of predicting a single normal distribution over the test set using statistics from the training dataset and as such does not have an associate standard error.

Fixed Normal	PNN	AttnLNP	AttnCNP	TNP	GTNP
-2.65	-2.26 ± 0.00	-2.37 ± 0.03	-3.56 ± 0.06	-2.21 ± 0.01	$\mathbf{-2.17 \pm 0.01}$

3.4 Weather Prediction Application

As another real-world application, we consider weather prediction, which is still a challenging problem for current machine learning models. (Bonavita, 2024) We use the ERA5-Land dataset by Muñoz-Sabater et al. (2021), which has daily meteorological fields such as precipitation and temperature. Each input $x \in \mathcal{X} \subset \mathbb{R}^4$ comprises standardized latitude, longitude, 2m-temperature, and geopotential (a proxy for elevation) from the same day; the target $y \in \mathcal{Y} \subset \mathbb{R}$ is the standardized daily total precipitation. After standardization, we treat the process as approximately stationary.

For training, we use a central European region (like (Foong et al., 2020), note that we do not compare to them since they have no official code released for the climate experiments) and sample 500 days uniformly between January 1, 1981 and December 31, 2020. For each day, we choose a random 28×28 spatial window restricted to valid cells. Each point is assigned to the context with probability

0.7, with the remainder as targets. For GTNP, edge features exclude temperature and geopotential; edges encode only pairwise geographic distance.

We hold out 10% of this data for validation. To test robustness to distribution shifts, we create 500 test samples from a different region and an earlier period (1 January 1970 - 31 December 1980). Note that climate change has been in effect significantly especially since 70s. (Sarkar & Maity, 2021) We also use 32×32 windows and, per example, sample the number of target points uniformly from $\{1, \ldots, 49\}$, placing them at random on the grid. This means any reasonable method would do better on the test set since the task depends on more context points and requires the prediction on a smaller target set. However, all other models fail under the shift, whereas GTNP performs even better due to injected equivariances, and outperforms other methods significantly in both tasks.

Table 4: **Total Target Log Likelihood for the Weather Prediction Tasks**. Validation log likelihood is the best seen during training. We use five seeds to report the mean and the standard error.

Task	AttnLNP	AttnCNP	TNP	GTNP
Best Validation Set	75.37 ± 30.73	114.56 ± 6.69	-332.37 ± 21.99	$\mathbf{796.62 \pm 26.99}$
Test Set	-386.28 ± 111.59	-488.80 ± 138.32	-1608.01 ± 36.93	$\mathbf{2752.63 \pm 134.63}$

3.5 Lipophilicity Application

Lipophilicity is a fundamental physicochemical property that influences drug pharmacokinetics and pharmacodynamics Miller et al. (2020); Constantinescu et al. (2019). Accurate prediction of it, together with well-calibrated uncertainty estimates, is therefore highly valuable in cheminformatics and drug discovery (Isert et al., 2023). Here, we use GTNP to meta-learn posteriors over unseen molecule lipophilicity, conditioned on a set of structurally similar molecules from a training dataset.

We use the Lipophilicity dataset from the GAUCHE library Griffiths et al. (2023), which contains 4,200 compounds represented as SMILES strings curated from the ChEMBL database Zdrazil et al. (2024). Each label corresponds to the octanol/water distribution coefficient (log D at pH 7.4). Additional dataset details are available in Griffiths et al. (2023).

Our NPs are trained by meta-learning across many local regression problems. This approach is motivated by the observation that, in many real-world tasks, local Gaussian Process (GP) models outperform global ones (Eriksson et al., 2019; Krityakierne & Ginsbourger, 2015). Global GPs often struggle because commonly used kernels are too rigid, and a single set of kernel hyperparameters is rarely optimal for the entire input space. For instance, if different regions of $\mathcal{X}$ require different kernel lengthscales, forcing a single global model to use one lengthscale typically degrades predictive performance. By contrast, local models tuned to the neighborhood of a specific test-time task can capture finer-grained structure. In Appendix C.7, we show that optimal kernel lengthscales for our lipophilicity task exhibit clear multi-modality, further motivating a local approach.

Following Griffiths et al. (2023), we represent each molecule using Mordred descriptors (Moriwaki et al., 2018), followed by dimensionality reduction via PCA to a 51-dimensional feature vector. We evaluate GTNP against two local GP baselines: (1) a GP conditioned on the context set with a fixed set of kernel hyperparameters chosen for strong overall performance, and (2) a GP that tunes its kernel hyperparameters locally by maximizing the marginal log-likelihood of the context set using L-BFGS (Liu & Nocedal, 1989). Both baselines use the Rational Quadratic (RQ) kernel, and we also replace the norm in ϕ with the RQ kernel for consistency.

We split the dataset into training, validation, and test sets of 1,400 molecules each. For every molecule, we identify the 24 most similar molecules (by RQ kernel similarity) within the same split to form the context set, with the molecule itself serving as the target ($N = 25$, $C = 24$). This design allows the model to focus on localized prediction tasks that mimic realistic discovery scenarios where only a small set of relevant analogs is available.

We do not compare against other Neural Process variants for this task because the combination of high-dimensional molecular representations and relatively small data size makes them impractical. As an additional reference, we include an uninformed baseline that predicts from a fixed standard normal distribution, which serves as a prior after whitening the labels.

Table 5 reports the mean and median log-likelihood on the test set for the single target point. GTNP performs reasonably compared to both GP baselines, demonstrating that meta-learning can serve as an effective alternative to carefully engineered priors, even with limited training data.

Table 5: **Target Log Likelihood for Lipophilicity Task** We use five seeds to report the mean and standard deviation of the statistics.

	Mean	Median
Standard Normal	-1.53	-1.21
GP (Fixed)	-1.38 ± 0.00	-1.03 ± 0.00
GP (With Tuning)	$\mathbf{-1.27 \pm 0.00}$	$\mathbf{-1.01 \pm 0.00}$
GTNP	-1.45 ± 0.01	-1.43 ± 0.01

4 Related Work

There are several other NPs that incorporate graphs into their architecture. Nassar et al. (2018) introduce the "Conditional Graph NP" (CGNP), which leverages bipartite graph convolutions (Nassar, 2018) to group samples in the context set that are in some neighborhood of each other. Importantly, they do not label the edges of their graph, a key aspect of our architectures. In addition, the "Function Neural Process" (FNP) (Louizos et al., 2019) first projects $\mathbf{x}$ into a latent space and then constructs two directed acyclic graphs in this space: one between the context points and one which is a bipartite graph between context and target sets. This avoids the need to create a global latent encoding that summarizes the context set. Our work differs from theirs since our graph encodes relationships in the original $\mathcal{X}$ space via edge labels. Lastly, in contrast to our work which frames regression data as a graph, there are also works on neural processes that operate on graph datasets (Carr & Wingate, 2019; Day et al., 2020; Liang & Gao, 2022).

Another relevant approach is the Graph Structured Surrogate Model (GSSM) (Wang & van Hoof, 2021), which was proposed in a meta-reinforcement learning setting and focuses on how well a downstream policy can adapt by inferring a latent task embedding from context data. While GSSM also incorporates graph structure and attention, its architecture and training objective are tightly coupled to policy optimization. In contrast, our work introduces a graph transformer–based neural process that directly meta-learns predictive uncertainties through global attention, without relying on latent task embeddings or reinforcement learning objectives.

In addition, there are two works that combine graph neural processes with attention, though in ways distinct from ours. The first is a chemical application by García-Ortegón et al. (2024), where molecules are represented as graphs of atoms and attention is used as the aggregator in the GNN. In contrast, our graph transformer employs global attention across the entire graph. The second is a spatio-temporal sensing application by Hu et al. (2023), where node features are time-dependent signals with external covariates, and distant nodes are truncated. Their method primarily relies on a Bayesian aggregator, using a local attentive aggregator only as a baseline, whereas our approach builds on global attention as a core mechanism.

5 Discussion

In this work, we advocate for representing the context and target sets for an NP as a graph. While our proposed models often had stronger performance over previous baselines, we found that they are especially stronger when the data is scarce. There are several limitations to our architectures, however. One of these limitations is that these architectures incur higher computational costs (Appendix C.8). This cost is unavoidable when viewing the data as a fully-connected graph since the architecture must account for the N^2 edges. Additionally, in this work we only use ϕ that is based on the norm between two points in $\mathcal{X}$. Depending on the application, this may be a drawback if the underlying process is non-stationary in $\mathcal{X}$ or if the data lies on a lower dimensional manifold.

REFERENCES

Joseph Abbate, R Conlin, and E Kolemen. Data-driven profile prediction for diii-d. *Nuclear Fusion*, 61(4):046027, 2021.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Massimo Bonavita. On some limitations of current machine learning weather prediction models. *Geophysical Research Letters*, 51(12):e2023GL107377, 2024.

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

Wessel P Bruinsma, Stratis Markou, James Requiema, Andrew YK Foong, Tom R Andersson, Anna Vaughan, Anthony Buonomo, J Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. *arXiv preprint arXiv:2303.14468*, 2023.

Andrew Carr and David Wingate. Graph neural processes: Towards bayesian graph neural networks. *arXiv preprint arXiv:1902.10042*, 2019.

Ian Char, Joseph Abbate, László Bardóczi, Mark Boyer, Youngseog Chung, Rory Conlin, Keith Erickson, Viraj Mehta, Nathan Richner, Egemen Kolemen, et al. Offline model-based reinforcement learning for tokamak control. In *Learning for Dynamics and Control Conference*, pp. 1357–1372. PMLR, 2023.

Hengwei Chen and Jürgen Bajorath. Meta-learning for transformer-based prediction of potent compounds. *Scientific Reports*, 13(1):16145, 2023.

Ruijie Chen, Ning Gao, Ngo Anh Vien, Hanna Ziesche, and Gerhard Neumann. Meta-learning regrasping strategies for physical-agnostic objects. *arXiv preprint arXiv:2205.11110*, 2022.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Teodora Constantinescu, Claudiu Nicolae Lungu, and Ildiko Lung. Lipophilicity as a central component of drug-like properties of chalchones and flavonoid derivatives. *Molecules*, 24(8):1505, 2019.

Ronald James Cotton, Fabian Sinz, and Andreas Tolias. Factorized neural processes for neural processes: K-shot prediction of neural responses. *Advances in Neural Information Processing Systems*, 33:11368–11379, 2020.

Iva Čvorović-Hajdinjak, Andjelka B Kovačević, Dragana Ilić, Luka Č Popović, Xinyu Dai, Isidora Jankov, Viktor Radović, Paula Sánchez-Sáez, and Robert Nikutta. Conditional neural process for nonparametric modeling of active galactic nuclei light curves. *Astronomische Nachrichten*, 343 (1-2):e210103, 2022.

Ben Day, Cătălina Cangea, Arian R Jamasb, and Pietro Liò. Message passing neural processes. *arXiv preprint arXiv:2009.13895*, 2020.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

Andrew Foong, Wessel Bruinsma, Jonathan Gordon, Yann Dubois, James Requeima, and Richard Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. *Advances in Neural Information Processing Systems*, 33:8284–8295, 2020.

Miguel García-Ortegón, Srijit Seal, Carl Rasmussen, Andreas Bender, and Sergio Bacallado. Graph neural processes for molecules: an evaluation on docking scores and strategies to improve generalization. *Journal of Cheminformatics*, 16(1):115, 2024.

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. *arXiv preprint arXiv:1910.13556*, 2019.

Ryan-Rhys Griffiths, Leo Klarner, Henry Moss, Aditya Ravuri, Sang T. Truong, Yuanqi Du, Samuel Don Stanton, Gary Tom, Bojana Ranković, Arian Rokkum Jamasb, Aryan Deshwal, Julius Schwartz, Austin Tripp, Gregory Kell, Simon Frieder, Anthony Bourached, Alex James Chan, Jacob Moss, Chengzhi Guo, Johannes P. Dürholt, Saudamini Chaurasia, Ji Won Park, Felix Strieth-Kalthoff, Alpha Lee, Bingqing Cheng, Alan Aspuru-Guzik, Philippe Schwaller, and Jian Tang. GAUCHE: A library for gaussian processes in chemistry. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vzrA6uqOis.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9): 5149–5169, 2021.

Junfeng Hu, Yuxuan Liang, Zhencheng Fan, Hongyang Chen, Yu Zheng, and Roger Zimmermann. Graph neural processes for spatio-temporal extrapolation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 752–763, 2023.

Clemens Isert, Jimmy C Kromann, Nikolaus Stiefl, Gisbert Schneider, and Richard A Lewis. Machine learning for fast, quantum mechanics-based approximation of drug lipophilicity. *ACS omega*, 8(2):2046–2056, 2023.

Azarakhsh Jalalvand, Joseph Abbate, Rory Conlin, Geert Verdoolaege, and Egemen Kolemen. Real-time and adaptive reservoir computing with application to profile prediction in fusion plasma. *IEEE Transactions on Neural Networks and Learning Systems*, 33(6):2630–2641, 2021.

Saurav Jha, Dong Gong, Xuesong Wang, Richard E Turner, and Lina Yao. The neural process family: Survey, applications and perspectives. *arXiv preprint arXiv:2209.00517*, 2022.

Makoto Kawano, Wataru Kumagai, Akiyoshi Sannai, Yusuke Iwasawa, and Yutaka Matsuo. Group equivariant conditional neural processes. *arXiv preprint arXiv:2102.08759*, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Egemen Kolemen, Jaemin Seo, Rory Conlin, Andrew Rothstein, SangKyeun Kim, Joseph Abbate, Keith Erickson, Josiah Wai, Ricardo Shousha, and Azarakhsh Jalalvand. Avoiding tokamak tearing instability with artificial intelligence. 2023.

Tipaluck Krityakierne and David Ginsbourger. Global optimization with sparse and local gaussian process models. In *International Workshop on Machine Learning, Optimization and Big Data*, pp. 185–196. Springer, 2015.

Xiang Li, Viraj Mehta, Johannes Kirschner, Ian Char, Willie Neiswanger, Jeff Schneider, Andreas Krause, and Ilija Bogunovic. Near-optimal policy identification in active reinforcement learning. *arXiv preprint arXiv:2212.09510*, 2022a.

Yumeng Li, Ning Gao, Hanna Ziesche, and Gerhard Neumann. Category-agnostic 6d pose estimation with conditional neural processes. *arXiv preprint arXiv:2206.07162*, 2022b.

Huidong Liang and Junbin Gao. How neural processes improve graph link prediction. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3543–3547. IEEE, 2022.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Christos Louizos, Xiahan Shi, Klamer Schutte, and Max Welling. The functional neural process. *Advances in Neural Information Processing Systems*, 32, 2019.

Stratis Markou, James Requeima, Wessel P Bruinsma, Anna Vaughan, and Richard E Turner. Practical conditional neural processes via tractable dependent predictions. *arXiv preprint arXiv:2203.08775*, 2022.

Viraj Mehta, Ian Char, Joseph Abbate, Rory Conlin, Mark Boyer, Stefano Ermon, Jeff Schneider, and Willie Neiswanger. Exploration via planning for information about the optimal trajectory. *Advances in Neural Information Processing Systems*, 35:28761–28775, 2022.

Luckeciano C Melo. Transformers are meta-reinforcement learners. In *international conference on machine learning*, pp. 15340–15359. PMLR, 2022.

Randy R Miller, Maria Madeira, Harold B Wood, Wayne M Geissler, Conrad E Raab, and Iain J Martin. Integrating the impact of lipophilicity on potency and pharmacokinetic parameters enables the use of diverse chemical space during small molecule drug optimization. *Journal of Medicinal Chemistry*, 63(21):12156–12170, 2020.

Hirotomo Moriwaki, Yu-Shi Tian, Norihito Kawashita, and Tatsuya Takagi. Mordred: a molecular descriptor calculator. *Journal of cheminformatics*, 10(1):1–14, 2018.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

Joaquín Muñoz-Sabater, Emanuel Dutra, Anna Agustí-Panareda, Clément Albergel, Gabriele Arduini, Gianpaolo Balsamo, Souhail Boussetta, Margarita Choulga, Shaun Harrigan, Hans Hersbach, et al. Era5-land: A state-of-the-art global reanalysis dataset for land applications. *Earth system science data*, 13(9):4349–4383, 2021.

Marcel Nassar. Hierarchical bipartite graph convolution networks. *arXiv preprint arXiv:1812.03813*, 2018.

Marcel Nassar, Xin Wang, and Evren Tumer. Conditional graph neural processes: A functional autoencoder approach. *arXiv preprint arXiv:1812.05212*, 2018.

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *arXiv preprint arXiv:2207.04179*, 2022.

Ari Pakman, Yueqi Wang, Catalin Mitelut, JinHyung Lee, and Liam Paninski. Neural clustering processes. In *International Conference on Machine Learning*, pp. 7455–7465. PMLR, 2020.

Young-Jin Park and Han-Lim Choi. A neural process approach for probabilistic reconstruction of no-data gaps in lunar digital elevation maps. *Aerospace Science and Technology*, 113:106672, 2021.

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

Alexander Pondaven, Märt Bakler, Donghu Guo, Hamzah Hashim, Martin Ignatov, and Harrison Zhu. Convolutional neural processes for inpainting satellite images. *arXiv preprint arXiv:2205.12407*, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Subharthi Sarkar and Rajib Maity. Global climate shift in 1970s causes a significant worldwide increase in precipitation extremes. *Scientific Reports*, 11(1):11574, 2021.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International conference on machine learning*, pp. 5779–5788. PMLR, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

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

Allen M Wang, Darren T Garnier, and Cristina Rea. Hybridizing physics and neural odes for predicting plasma inductance dynamics in tokamak fusion reactors. *arXiv preprint arXiv:2310.20079*, 2023.

Qi Wang and Herke van Hoof. Model-based meta reinforcement learning using graph structured surrogate models. *arXiv preprint arXiv:2102.08291*, 2021.

Xuesong Wang, Lina Yao, Xianzhi Wang, Hye-young Paik, and Sen Wang. Global convolutional neural processes. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 699–708. IEEE, 2021.

Cristopher Williams and Carl Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press, 2005.

Yigit Yildirim and Emre Ugur. Learning social navigation from demonstrations with conditional neural processes. *Interaction Studies*, 23(3):427–468, 2022.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. corr abs/1703.06114 (2017). *arXiv preprint arXiv:1703.06114*, 2017.

Barbara Zdrazil, Eloy Felix, Fiona Hunter, Emma J Manners, James Blackshaw, Sybilla Corbett, Marleen de Veij, Harris Ioannidis, David Mendez Lopez, Juan F Mosquera, et al. The chembl database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic Acids Research*, 52(D1):D1180–D1192, 2024.

A Graph Edge Evolution Neural Process

Graph Edge Evolution Neural Process In GTNP, the embeddings for the vertices are updated after each block; however, a more powerful model would iteratively update the edge embeddings. Towards this, we propose the Graph Edge Evolution NP (GEENP) which acts exclusively on the edge embeddings of the graph. As such, we must include both information from $\mathbf{x}$ and $\mathbf{y}$ in the edges. The edge encoder, f_θ^E now takes in five inputs:

$$E_{i,j}^0 = f_\theta^E\big(\phi(\mathbf{x}_i, \mathbf{x}_j), \mathbf{y}_i \mathbb{1}\{i \le C\}, \mathbf{y}_j \mathbb{1}\{j \le C\}, \mathbb{1}\{i \le C\}, \mathbb{1}\{j \le C\}\big)$$

This additional information is also used to indicate which vertices are context vertices and which are target vertices.

How should one update these edge embeddings? One naive approach would be to use a transformer on the N^2 edge embeddings. Not only would the computational costs of this scale with $\mathcal{O}(N^4)$, but this model would likely have too much flexibility and not enough structure. To strike the correct balance, we look to the Gaussian Process (GP) for inspiration.

The GP is a stochastic process where the distribution of any finite collection of points is distributed as a multivariate normal. The GP is fully characterized by a mean function and a kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let $K \in \mathbb{R}^{N \times N}$ be the matrix such that $K_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. After conditioning on a context set of observations, the posterior is again a multivariate normal distribution. The bulk of the computational effort for computing the posterior is dedicated to inverting $K_{1:C,1:C}$, and one straight-forward way of doing this is via Gaussian elimination. Thus, perhaps the only operation necessary for sophisticated behavior is updating each row of E with a linear combination of the other rows' features.

Leveraging this idea, we replace the regular attention scheme over the N^2 edge embeddings with "Gaussian Elimination Attention" (GEAttention). Thinking about the set of edges as a matrix of embeddings, the key idea is to constrain the attention mechanism to only make convex combinations of values in the same column. Moreover, the weighting scheme for these values must be the same for each column in E. This key idea is depicted in Figure 5. The `GEAttention` operator for the edge embedding at row i and column j is

$$\texttt{GEAttention}_{i,j}(E) = \sum_{r=1}^{N} \alpha_{i,r} v_{r,j}$$

$$\text{where} \quad \alpha_{i,r} = \frac{1}{N} \sum_{n=1}^{N} \frac{\exp\left(\frac{1}{\sqrt{d}} \langle q_{i,n}, k_{r,n} \rangle\right)}{\sum_{m=1}^{N} \exp\left(\frac{1}{\sqrt{d}} \langle q_{i,n}, k_{m,n} \rangle\right)}$$

Because it is somewhat unconventional, we will now further explain $\alpha_{i,j}$ for `GEAttention`. First note that $\alpha_{i,r}$ prescribes the weighting on values from row r when updating embeddings in row i. The term inside the summation is standard attention across the n^{th} column. The sum is then used for averaging this information across columns. This scheme reduces the computational complexity from $\mathcal{O}(N^4)$ to $\mathcal{O}(N^3)$. Again, we use a multi-headed version of this scheme in practice. After L blocks, the mean and standard deviation of a normal are predicted via $\mu_i, \sigma_i = g_\theta\left(E_{i,i}^L\right)$. The full architecture description can be found in Appendix B.2.

B Implementation

B.1 More GTNP Details

Algorithm 1 shows one graph transformer block. Here q_θ is a neural network with one hidden layer neural network with a ReLU activation function. Before multi-headed graph attention and before q_θ we use `LayerNorm` (Ba et al., 2016). Note that the architecture for this block is almost exactly the same as the blocks in GPT2 (Radford et al., 2019); however, we use $2d$ hidden units in q_θ following Nguyen & Grover (2022) instead of $4d$.

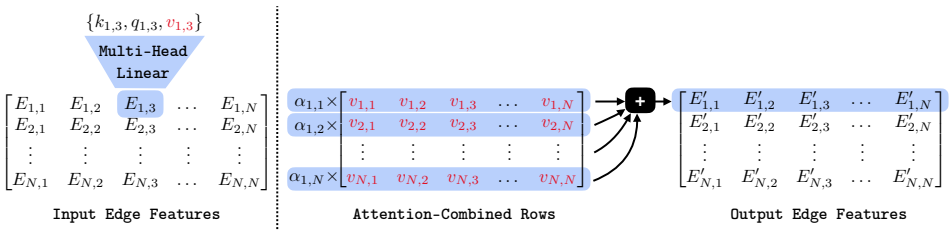

Figure 5: **An Illustration of the Key Idea behind `GEAttention`.** Each edge embedding outputs a key, query, and value (left of the divider). However, we combine entire rows of values together inspired by the row operations of Gaussian Elimination (right of the divider).

Autoregressive Training Scheme. We use the same scheme as TNP-A in Nguyen & Grover (2022) in order to efficiently consider multiple context sizes at once per mini-batch update. To do this, instead of using $\mathbf{x}$ we use $\tilde{\mathbf{x}} = \mathbf{x} \oplus \mathbf{x}_{C+1:N}$. The corresponding node embeddings are then $\nu_i = f_\theta^V(\mathbf{y}_i)$ for $i \leq N$ and $\nu_i = f_\theta^V(\mathbf{0})$ if $i > N$. Let $M \in \mathbb{R}^{2N-C \times 2N-C}$ be a matrix of the masks for the attention where $M_{i,j} = 1$ if the i^{th} vertex can attend to the j^{th} vertex and 0 otherwise. The entries of this matrix are as follows:

$$M_{i,j} = \begin{cases} 1 & j \leq C \\ 1 & i \leq N \text{ and } j \leq i \\ 1 & j - C < i - N \\ 0 & \text{else} \end{cases}$$

This is visualized on page four of Nguyen & Grover (2022). Effectively, this masking scheme allows target points to attend to observations of other target points occuring earlier in the sequence. Alternatively, we could simply use $\mathbf{x}$, the vertex encoding scheme described in Section 2.2, and a masking matrix of $M \in \mathbb{R}^{N \times N}$ where

$$M_{i,j} = \begin{cases} 1 & j \leq C \\ 0 & \text{else} \end{cases}$$

This would only consider one context during training (although at test time, the target set could still be predicted autoregressively). This is akin to TNP-D in Nguyen & Grover (2022), which was found to achieve worse log likelihood. We also find that this more straightforward masking scheme yields slightly worse results for GTNP. Check Appendix D for experimental results of GEENP.

B.2 More GEENP Details

Algorithm 2 shows one GEENP block. Note that most of the details are indistinguishable from Algorithm 1 besides the type of attention used and the operation over edge embeddings instead of vertex embeddings.

Algorithm 1 One Graph Transformer Block

Input: V, E
for $\nu_i^\ell \in V$ **do**
  $\nu_i^\ell = \nu_i^\ell + \texttt{MultiHeadedGraphAttention}_i($
    $\texttt{LayerNorm}(V), E)$
  $\nu_i^{\ell+1} = \nu_i^\ell + q_\theta(\texttt{LayerNorm}(\nu_i^\ell))$
end for
Return: V

C Experiment Details

C.1 Reproducibility

We include an anonymized pytorch (Paszke et al., 2019) implementation of GTNP and GEENP in the supplementary material, instructions for installing dependencies, and directions for launching synthetic experiments (i.e. the experiments in Sections 3.1 and 3.2). See README.md for more details.

C.2 Hyperparameters

Table 6: **Hyperparameters for GTNP and GEENP.**

Batch Size	16
Learning Rate	5e-4
Learning Rate Scheduler	Cosine Annealing
d	64
H	4
Blocks	6
Activation Function	ReLU
f_θ^V and f_θ^E Depth	4
f_θ^V and f_θ^E Width	64
g_θ Depth	1
g_θ Width	128
q_θ width (see Algorithms 1 and 2	128

Table 6 show the hyperparameters used for GTNP and GEENP. We selected these hyperparameters to be as close as possible to the ones used in TNP for a fair comparison. For each of the other baselines, we use the same hyperparameters they report. In particular, we use the same configurations that appear in `https://github.com/tung-nd/TNP-pytorch/tree/master` and `https://github.com/cambridge-mlg/convcnp`.

C.3 Additional Infinite Data Regime Details

Following the procedure in Nguyen & Grover (2022), all methods were trained on 1M batch updates, where each batch contains 16 sampled functions. The exception to this is in the 4D case, where it seemed that models were still learning after 1M batch updates. Thus we increase the number of batch updates to 10M for this setting. For test time, we generate 3,000 of these batches to evaluate on. Like during training, each batch has a random N and C chosen in the ranges of $[6, 50]$ and $[3, 47]$, respectively. We use the code in the official repository for Nguyen & Grover (2022) to generate the test set for the 1D experiment and our own code for the higher dimensions.

C.4 Additional Finite Data Regime Details

When dealing with finite datasets, we shuffle the data every epoch in order to get different context-target splits in the data. We reserve 10% of the total data as a validation set. We train each model

Algorithm 2 One GEENP Block

Input: E
for $E_{i,j}^\ell \in E$ **do**
  $E_{i,j}^\ell = E_{i,j}^\ell +$ `MultiHeadedGEAttention`$_{i,j}$(
    `LayerNorm`(E))
  $E_{i,j}^{\ell+1} = E_{i,j}^\ell + q_\theta($`LayerNorm`$(E_{i,j}^\ell))$
end for
Return: E

until 100 epochs after the best validation loss was observed (with a maximum of 5,000 epochs). We checkpoint the model that achieves the best validation loss and use this during evaluation.

We use a test set with 100,000 data points for each of the dimensions. The same test set is used regardless of how much training data is used. We now describe how to compute the normalized metrics found in Table 2. Let s_c be the average log joint likelihood that the trained model achieves on the test set given a context size of c. Furthermore, let f be the true GP that generated the data. We then define s_c^{min} and s_c^{max} as

$$s_c^{\text{min}} = \frac{1}{M}\sum_{m=1}^{M} f(\mathbf{y}_{c+1:N}^{(m)} | \mathbf{x}_{c+1:N}^{(m)})$$

$$s_c^{\text{max}} = \frac{1}{M}\sum_{m=1}^{M} f(\mathbf{y}_{c+1:N}^{(m)} | \mathbf{x}^{(m)}, \mathbf{y}_{1:c}^{(m)})$$

where $\mathbf{x}^{(m)}$ and $\mathbf{y}^{(m)}$ are the m^{th} set of test points. In other words, s_c^{min} is the joint log likelihood using the true prior and s_c^{max} is the joint log likelihood using the true posterior. The final reported score is then

$$100 \times \frac{1}{49}\sum_{c=1}^{49} \frac{s_c - s_c^{\text{min}}}{s_c^{\text{max}} - s_c^{\text{min}}}$$

Note that when M is large this score cannot be over 100; however, it is possible to get a score lower than 0.

C.5 Additional Nuclear Fusion Experiment Details

The dimensions of $\mathcal{X}$ for the nuclear fusion data are the current measurements of β_N, rotation, power injected from the neutral beams, torque injected from the neutral beams, and the backward difference of these four measurements. Additionally, because the power and torque injected from the neutral beams are actuators, we include the forward differences for the power and torque. Each shot on average has roughly 107 time steps.

The architecture for the PNN is the same as it appears in Chua et al. (2018). In particular, we use a fully connected network with 4 hidden layers each with 200 units. We use the swish activation function (Ramachandran et al., 2017) and have two dedicated heads for the mean and log variance predictions. We use mini-batches of size 256 and a learning rate of 1e-3.

C.6 Additional Weather Prediction Experiment Details

The daily precipitation values are accumulated from 23:00 of the previous day to 23:00 of the day of interest. Our training and validation region if $48°\text{N}$–$56°\text{N}$, $8°\text{E}$–$26°\text{E}$, whereas the test region is $42°\text{N}$–$46°\text{N}$, $19°\text{E}$–$26°\text{E}$.

We keep architectural and training settings as elsewhere but set mini-batch size to 1 due to high context and test sizes.

C.7 Additional Lipophilicity Experiment Details

The RQ kernel is defined as

$$k_{RQ}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right).$$

Recall that the train split used to train our GTNP models contains sequences of molecule representations. For each of the 1400 sequences, we compute the optimal hyperparameters using L-BFGS that maximise the marginal log-likelihood of the context points in the sequence. The two histograms in Figure 6 show the distribution of optimal hyperparameters over the train split. For α in particular, two clear peaks are present, which illustrate the inappropriateness of using a global GP model with a single choice of kernel hyperparameters to fit all the training data.

For the GP (fixed) baseline, α and ℓ were chosen to be right modes in their respective histograms, $\ell = 2.47$, $\alpha = \exp(7.01)$. The scale parameter σ^2 was set to 1 as we are using whitened data.

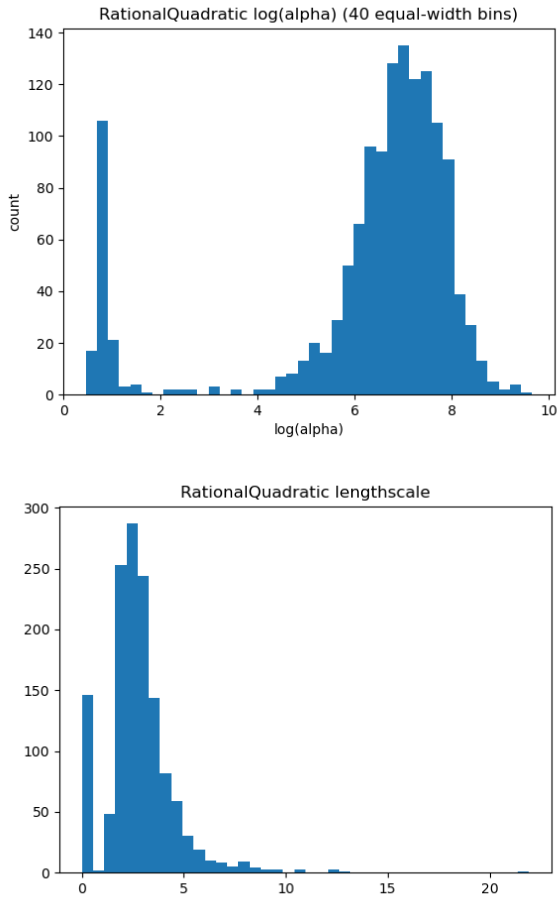

Figure 6: **Histograms of Optimal** $\log \alpha$ **and** ℓ **RQ Kernel Hyperparameters for the Lipophilicity Task.** Note the two peaks illustrating the multi-modal nature of optimal kernel hyperparamters for our task.

C.8 Compute Details

We train our models using Nvidia Titan Xp GPUs. Other baselines were trained using a mix of these GPUs and Titan X Pascals. Since the cluster we use to train on has many users, it is difficult to get a precise timing comparison; however, we give estimates in Table 7. Note that TNP and GTNP are considerably faster at evaluation because only one forward pass is necessary to compute the joint log probability due to the masking scheme described in Appendix B.1.

Table 7: **Approximate Run Times**. Each estimate is for one seed on the infinite 1D experiment.

Method	Train	Evaluation
AttnLNP	45m	2m 23s
AttnCNP	33m	1m 57s
ConvCNP	27m	2m 11s
TNP	52m	16s
GTNP	92m	39s
GEENP	89m	15m 32s

D EXPERIMENTAL RESULTS WITH GEENP

Here, we re-report some of the tables and figures from the main paper with GEENP results included.

The results in Table 8 and Figure 7 shows that in infinite data regime, because of its flexibility GEENP even outperforms GTNP.

Table 8: **Infinite Dataset Results**. The reported metric average joint log likelihood over the test set. We use five seeds to report the average and the standard error.

Dimension	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP	GEENP
1D	12.92 ± 0.03	12.62 ± 0.03	20.64 ± 0.08	21.34 ± 0.02	21.80 ± 0.01	$\mathbf{22.48 \pm 0.05}$
2D	-4.43 ± 0.02	-4.69 ± 0.02	—	-4.16 ± 0.55	-2.99 ± 0.01	$\mathbf{-2.54 \pm 0.03}$
4D	-9.37 ± 0.01	-9.40 ± 0.00	—	-9.86 ± 0.00	$\mathbf{-9.25 \pm 0.00}$	$\mathbf{-9.25 \pm 0.00}$

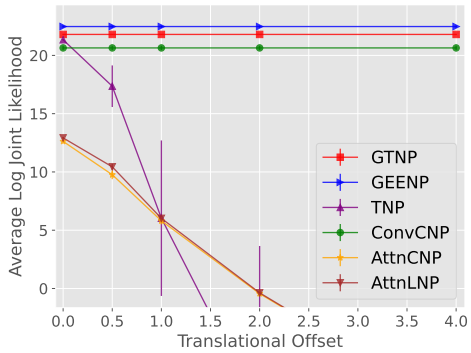

Figure 7: **Log Likelihood vs. Translational Offset in the 1D Infinite Data Case.** The y-axis shows the average log joint likelihood over the test set and the x-axis shows the amount of translational shift made on the x inputs.

Figure 9 and Table 9 show that although GTNP is reasonable, its uncertainty contours look unstructured away from observation points, and we need 100K data to match GTNP in performance, again solidifying the hyopthesis that GEENP is better only with more data.

Table 10 shows that GEENP is not particularly good in the fusion task.

We did not do the lipophilicity experiment with GEENP. Similarly, the weather prediction test set is simply infeasible with GEENP due to high context set size. The runs would take days. But validation performance is reported in Table 11.

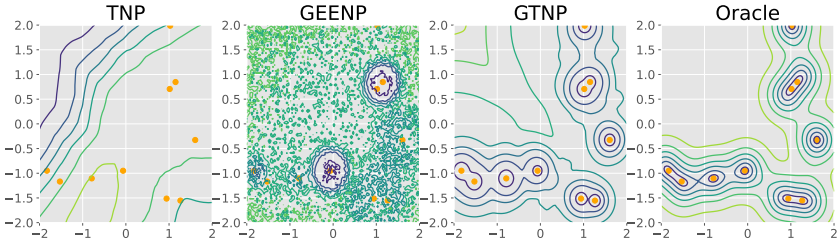

Figure 8: **Contour of the Predicted Standard Deviation for the 2D 1K Task**. The far right plot titled "Oracle" shows the standard deviation of the posterior GP that generated the data. The orange points show points in the context set.

Table 9: **Finite Dataset Results**. The metric is a standardized score where a score of 100 means that the NP has exactly reproduced the GP, and a score of 0 means that the NP produces a worse log likelihood than simply predicting the prior (see Appendix C.4 for the exact definition of this metric). We use five seeds to report the average and the standard error.

Dataset	AttnLNP	AttnCNP	ConvCNP	TNP	GTNP	GEENP
1D 1K	4.88 ± 1.74	-2.72 ± 4.20	34.26 ± 5.38	-2.62 ± 1.86	$\mathbf{61.90 \pm 1.92}$	44.81 ± 0.95
1D 10K	66.83 ± 0.75	63.94 ± 1.06	65.48 ± 4.52	82.01 ± 0.58	$\mathbf{83.73 \pm 0.59}$	74.94 ± 2.00
1D 100K	71.76 ± 0.74	71.92 ± 0.68	93.04 ± 0.35	92.95 ± 0.38	$\mathbf{93.86 \pm 0.23}$	93.45 ± 0.66
2D 1K	-2.56 ± 1.48	-2.80 ± 1.79	—	-8.78 ± 7.38	$\mathbf{56.94 \pm 0.64}$	11.20 ± 7.68
2D 10K	61.18 ± 0.96	57.81 ± 2.24	—	63.92 ± 0.42	$\mathbf{73.49 \pm 1.06}$	62.58 ± 2.74
2D 100K	66.04 ± 2.10	68.33 ± 1.07	—	82.33 ± 0.35	82.92 ± 0.34	$\mathbf{88.67 \pm 1.08}$
4D 1K	-31.04 ± 15.97	-41.94 ± 19.01	—	-9.24 ± 1.22	$\mathbf{-6.10 \pm 2.91}$	-9.52 ± 3.07
4D 10K	24.74 ± 7.62	-43.79 ± 29.08	—	-2.59 ± 2.31	$\mathbf{78.90 \pm 1.40}$	70.04 ± 3.34
4D 100K	59.83 ± 2.40	51.47 ± 4.58	—	71.30 ± 1.54	$\mathbf{91.08 \pm 0.49}$	90.31 ± 0.60
Average	35.74	24.69	—	41.03	**68.53**	58.50

Table 10: **Average Joint Log Likelihood for Nuclear Fusion Task**. We use five seeds to report the average and the standard error. The method "Fixed Normal" is the result of predicting a single normal distribution over the test set using statistics from the training dataset and as such does not have an associate standard error.

Fixed Normal	PNN	AttnLNP	AttnCNP	TNP	GTNP	GEENP
-2.65	-2.26 ± 0.00	-2.37 ± 0.03	-3.56 ± 0.06	-2.21 ± 0.01	$\mathbf{-2.17 \pm 0.01}$	-2.27 ± 0.01

Table 11: **GEENP Best Performance on Weather Prediction Validation Set**. Average log likelihood for the targets.

Training & Validation Time	Log Likelihood
48h	150.99
31h 21m	79.24
47h 59m	120.54
22h 08m	203.09
20h 59m	-90.99