Taylorformer: Probabilistic Modelling for Random Processes including Time Series

Omer Nivron [*1] **Raghul Parthipan** [*1 2] **Damon J. Wischik** [1]

Abstract

We propose the Taylorformer for random processes such as time series. Its two key components are: 1) the LocalTaylor wrapper which adapts Taylor approximations (used in dynamical systems) for use in neural network-based probabilistic models, and 2) the MHA-X attention block which makes predictions in a way inspired by how Gaussian Processes' mean predictions are linear smoothings of contextual data. Taylorformer outperforms the state-of-the-art in terms of log-likelihood on 5/6 classic Neural Process tasks such as meta-learning 1D functions, and has at least a 14% MSE improvement on forecasting tasks, including electricity, oil temperatures and exchange rates. Taylorformer approximates a consistent stochastic process and provides uncertainty-aware predictions. Our code is provided in the supplementary material.

1. Introduction

Stochastic processes are appealing because they provide uncertainty along with point predictions, making them useful for modelling random systems and for decision-making. Limitations of conventional techniques like Gaussian Processes (GP) include high computational costs and difficulties in specifying kernels. Notable efforts to combine the benefits of stochastic processes and neural networks have included leveraging attention, such as the Attentive Neural Process (Kim et al., 2019), a batch-generation model, and the state-of-the-art Transformer Neural Process (TNP) (Nguyen & Grover, 2022), an autoregressive model. Both models are built on the standard multi-head attention introduced by Vaswani et al. (2017).

Models based on the standard multi-head attention, such as GPT (Liu et al., 2018), have shown impressive results on natural language tasks, where the goal is to predict sequences of discrete values. However, the scientific community is still working out what adaptations are needed to get similarly impressive results for continuous problems (such as time-series or spatial processes), where the goal is to predict real-valued targets on a continuum.

Problem setup. Our goal is to model the distribution of a set of unobserved points (target), $\mathbf{Y}_T$, at a specified set of indices, $\mathbf{X}_T$, given a set of observed points and their associated indices (context), $\{\mathbf{X}_C, \mathbf{Y}_C\}$. Here, $\mathbf{Y}_T \in \mathbb{R}^{n_T \times \alpha}$, $\mathbf{X}_T \in \mathbb{R}^{n_T \times \beta}$, $\mathbf{Y}_C \in \mathbb{R}^{n_C \times \alpha}$ and $\mathbf{X}_C \in \mathbb{R}^{n_C \times \beta}$, where n_C and n_T are the numbers of points in the context and target sets respectively. The context or target set may be permuted arbitrarily — they need not be ordered.

Our Contributions. (1) We propose the Taylorformer, a probabilistic model for continuous processes. It produces predictions and associated uncertainty for interpolation and extrapolation settings and irregularly sampled data. It approximates a consistent stochastic process (Garnelo et al., 2018b) and builds on insights derived from two well-known approaches for modelling continuous processes: Taylor series and Gaussian Processes (GPs). (2) We introduce the LocalTaylor wrapper. Taylor series can be used for local approximations of functions, such as in dynamical systems. But they are only useful under certain conditions. LocalTaylor uses a neural network to learn how and when to use the information provided by a Taylor approximation. (3) We create the MHA-X block to incorporate an inductive bias from GPs (specifically, how the mean prediction is a linear smoothing of contextual values). (4) We introduce a masking procedure for attention units in the Taylorformer to allow training the ability to predict points at arbitrary locations.

This work focuses on building better models for stochastic processes, so we do not focus on computational cost or inference efficiency.

We demonstrate Taylorformer's performance on several

[*]Equal contribution [1]Department of Computer Science and Technology, University of Cambridge, UK [2]British Antarctic Survey, Cambridge, UK. Correspondence to: Omer Nivron <on234@cam.ac.uk>, Raghul Parthipan <rp542@cam.ac.uk>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023.

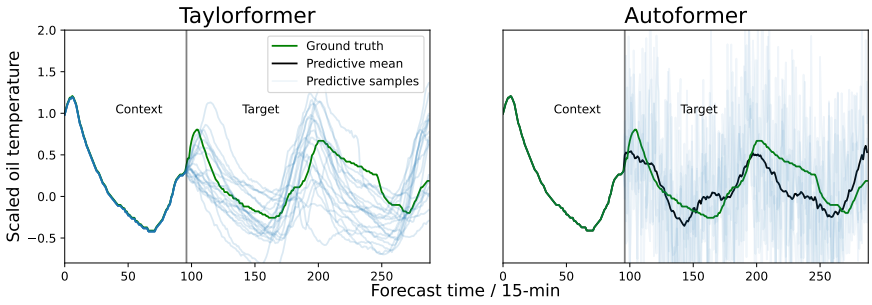

Figure 1. Taylorformer (left) can generate higher quality samples on the ETT (Wu et al., 2021) dataset than the state-of-the-art Autoformer (Wu et al., 2021) (right). This is representative of the general difference between these models at generation time. The task is to predict the next 48 hours (192 target points) given a 24-hour window (96 context points).

tasks in the Neural Process (NP) and time-series forecasting literature. Together, these evaluate the following i) both mean predictions and modelled uncertainty, ii) in both interpolation and extrapolation settings and iii) consistency, where the context/target sets can either be ordered or take arbitrary permutations. Taylorformer has a better log-likelihood/mean-squared-error than state-of-the-art models on 17/18 tasks, including image completion (2D regression) and electricity consumption. The Taylorformer shows 14-18%, 21-34% and 95-99% reductions in MSE compared to the next best model for forecasting transformers' oil temperatures, electricity consumption, and exchange rates, respectively. Figure 1 shows samples from the Taylorformer and the Autoformer for the ETT dataset (Wu et al., 2021).

2. Related Work

There has been a separation in the literature between those that use neural networks to model random processes (the Neural Process family) and those that model time-series. We believe there is a shared project and are motivated by both.

Neural Processes and Consistency. The initial members of the Neural Process family (the Conditional Neural Process, CNP, (Garnelo et al., 2018a) and the Neural Process, NP, (Garnelo et al., 2018b)) set out to create a neural network version of Gaussian Processes (GPs), to combine the benefits of both. GPs are probabilistic models which specify distributions over functions. They can be used to perform regression and provide uncertainty estimates. Prior knowledge can be incorporated through the specification of a parametric kernel function, and they can adapt to new observations. One of their desirable properties is that a GP is a consistent stochastic process (roughly, it does not matter what order you present the data in); however, they can be computationally intensive to use, and selecting appropriate kernels is a challenging task.

One consequence of consistency is target equivariance which means that for a given probability model with a likelihood function, p_θ, for a given context set $\{\mathbf{X}_C, \mathbf{Y}_C\}$ and target set $\{\mathbf{X}_T, \mathbf{Y}_T\}$

$$p_\theta(\mathbf{Y}_{\pi(T)} \mid \mathbf{Y}_C, \mathbf{X}_C, \mathbf{X}_{\pi(T)}) = p_\theta(\mathbf{Y}_T \mid \mathbf{Y}_C, \mathbf{X}_C, \mathbf{X}_T) \tag{1}$$

where $\pi(T)$ is a permutation of the target set. Another consequence is context invariance, which means,

$$p_\theta(\mathbf{Y}_T \mid \mathbf{Y}_C, \mathbf{X}_C, \mathbf{X}_T) = p_\theta(\mathbf{Y}_T \mid \mathbf{Y}_{\pi(C)}, \mathbf{X}_{\pi(C)}, \mathbf{X}_T) \tag{2}$$

where $\pi(C)$ is a permutation of the context set.

The TNP (Nguyen & Grover, 2022) is a state-of-the-art autoregressive Neural Process based on the transformer architecture, with adaptations, such as a new mask, for continuous processes. Context invariance is enforced via the architecture, but they rely on a shuffling approach to encourage target equivariance through the training algorithm. Nguyen & Grover (2022) define the desired target-equivariant model by means of its likelihood $\tilde{p}_\theta$, $\tilde{p}_\theta(\mathbf{Y}_T \mid \mathbf{Y}_C, \mathbf{X}_C, \mathbf{X}_T) = \mathbb{E}_\pi[p_\theta(\mathbf{Y}_{\pi(T)} \mid \mathbf{Y}_C, \mathbf{X}_C, \mathbf{X}_{\pi(T)})]$ where p_θ is the likelihood of the base TNP model. The Expectation is approximated using a Monte Carlo average over randomly sampled permutations during training. Inspired by the TNP, we use a similar shuffling procedure for training.

The initial Neural Process work (CNP, NP) enforced consistency through constraints to the architectures, but such constraints resulted in the underfitting of data. Advances included using attention in a way that maintains consistency (Kim et al., 2019), using convolutional architectures (Gordon et al., 2019) and using hierarchical latent variables to capture global and local latent information separately (Wang & Van Hoof, 2020). The TNP outperformed all these by trading an increase in flexibility for the loss of architecture-enforced consistency.

Forecasting. There have been various improvements to standard Attention layers in the forecasting literature. Two

state-of-the-art examples are Informer (Zhou et al., 2020) and Autoformer (Wu et al., 2021). Autoformer proposes a replacement for the classic multi-head attention block by using an auto-correlation mechanism, and the Informer offers a more efficient way to choose the most relevant keys for specific queries using ProbSparse attention.

One key difference between Taylorformer and Autoformer/Informer is that we model the targets autoregressively, while the latter uses a batch-generation approach (one-shot generation). Batch-generation approaches model the targets as conditionally independent given the context, analogous to CNPs (Garnelo et al., 2018a). The conditional independence assumption is restrictive, and allowing conditional dependencies between targets can improve results, as seen by how the NP (Garnelo et al., 2018b) improves on the CNP.

Much other work focuses on making attention mechanisms more efficient (Zaheer et al., 2020; Wang et al., 2020; Katharopoulos et al., 2020; Kitaev et al., 2020; Choromanski et al., 2020). And there is another strand combining both accuracy and efficiency (Wu et al., 2021; Zhou et al., 2020; LI et al., 2019). Many are batch-generation models, so they are more efficient at forecast time than autoregressive ones since all the targets can be produced in one shot. As noted earlier, though, the gap that we aim to fill is how to make a better attention model for the data, putting efficiency aside.

3. Our approach: Taylorformer

Taylorformer is an autoregressive probabilistic model. The likelihood assigned to $\mathbf{Y}_T$ given $\mathbf{X}_T, \mathbf{Y}_C, \mathbf{X}_C$ is decomposed in the typical autoregressive manner

$$p(\mathbf{Y}_T|\mathbf{X}_T, \mathbf{Y}_C, \mathbf{X}_C) = p(Y_T^1|X_T^1, \mathbf{Y}_C, \mathbf{X}_C)\times \prod_{i=2}^{n_T} p(Y_T^i|X_T^i, \mathbf{Y}_T^{1:i}, \mathbf{X}_T^{1:i}, \mathbf{Y}_C, \mathbf{X}_C) \quad (3)$$

where $\mathbf{Y}_T \in \mathbb{R}^{n_T\times\alpha}, \mathbf{X}_T \in \mathbb{R}^{n_T\times\beta}, \mathbf{Y}_C \in \mathbb{R}^{n_C\times\alpha}, \mathbf{X}_C \in \mathbb{R}^{n_C\times\beta}$ and $Y_T^i \in \mathbb{R}^\alpha$. The superscript notation $1:i$ means all indices from 1 to $i-1$.

The following equation reflects our full contribution. We model Y_T^i given X_T^i and a set of $\{\mathbf{X}, \mathbf{Y}\}$ information (which will contain context points and already-predicted target points) as follows, illustrated here using $\{\mathbf{X}_C, \mathbf{Y}_C\}$

$$Y_T^i = \text{LocalTaylor}(\text{MHA-X-Net}, X_T^i, \mathbf{X}_C, \mathbf{Y}_C; p, q) + \text{Linear}(\text{MHA-X-Net}(\mathbf{W}, X_T^i, \mathbf{X}_C, \mathbf{Y}_C)) \cdot Z_i \quad (4)$$

where LocalTaylor is detailed in section 3.1, and p and q are its hyperparameters. It is a wrapper around the neural network MHA-X-Net and uses approximations based on the Taylor series to augment predictions. Part of the LocalTaylor is the creation of the features $\mathbf{W}$ (equation (9)). The MHA-X-Net is a neural network composed of standard multi-head attention units (referred to here as MHA-XY) and our new MHA-X block (section 3.2). $Z^i \sim N(0,1)$. The model architecture is shown in Figure 2a. In section 3.3, we explain how the model is trained.

3.1. LocalTaylor

Taylor polynomials (curtailed Taylor series) are useful for making local predictions for functions, in low-noise settings, based on information at neighbouring points. Such approaches are useful in machine learning contexts too. The usefulness of a Taylor approximation is especially clear in the modelling of dynamical systems (Brenowitz & Bretherton, 2018; Gagne et al., 2020; Liu et al., 2022; Parthipan et al., 2022; Chen et al., 2018; Ruthotto & Haber, 2018; Lu et al., 2018), where one goal is to emulate the fixed-time-step evolution of a system that evolves based on an ordinary differential equation (ODE) of the form $\frac{dy}{dt} = f(y)$ where an approach based on modelling residuals

$$y_{a+\delta t} = y_a + \delta t\, \text{NeuralNet}(y_a) \quad (5)$$

often works far better than modelling the target outright, as in

$$y_{a+\delta t} = \text{NeuralNet}(y_a) \quad (6)$$

where NeuralNet is a machine learning function. In this example, equation (5) has a clear parallel with the first order Taylor approximation of $y_{a+\delta t}$ about y_a which is $y_{a+\delta t} = y_a + \delta t \left.\frac{dy}{dt}\right|_{t=a}$. The neural network in equation (5) can be interpreted as a term to model all higher-order terms in the Taylor expansion.

Taylor approximations are not obviously suited for making long-range predictions (i.e., where δt in equation (5) is large) nor when the data is too noisy. The naive fix for long-range predictions would be to include higher-order terms in the Taylor approximation. However, estimations of higher-order derivative terms will become more inaccurate given the successive estimations needed for each higher-order derivative. Noisy data would compound the difficulty in estimating accurate derivatives. Moreover, some functions will have a Taylor series that only converges within a specific range. Hence, a Taylor approximation's appropriateness will depend on the function being approximated, the point, a, used for the expansion and the distance between a and the predicted point.

The **LocalTaylor wrapper** models the mean prediction of Y_T^i as the sum of i) explicit Taylor expansion terms and ii) a neural network adjustment function that uses Taylor expansion terms as features. This approach confronts the challenge of working out when Taylor approximations are useful (dependent on the noisiness of data, number of terms

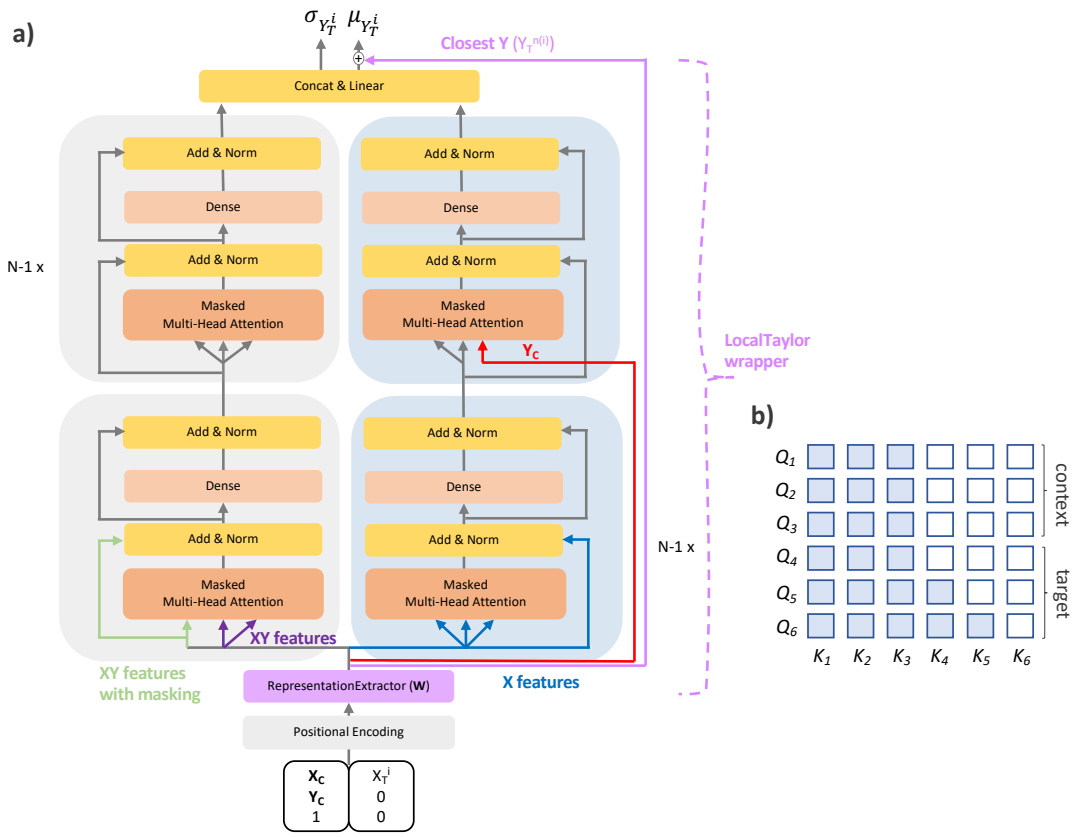

Figure 2. a) Taylorformer architecture corresponding to equation (4). LocalTaylor is a wrapper around the central neural network, MHA-X-Net. The channels on the right-hand side are MHA-X. The ones on the left are MHA-XY. The noted features are shown in the following equations: XY features with masking, eq. (14), XY features, eq. (15), and X features, eq. (16). b) Example mask for $n_C = 3$ and $n_T = 3$. Each token can attend to other shaded tokens in its row.

used in the Taylor approximation, and the gap over which predictions are made) by using a neural network to *learn* how to use this information when making predictions. Giving features based on the Taylor expansion to the neural network can remove the hard-coded Taylor expansion terms if deemed not useful. Concretely,

$$\mu_{Y_T^i} = \text{LocalTaylor}(\text{NeuralNet}, \mathbf{X}_C, X_T^i, \mathbf{Y}_C; p, q) \quad (7)$$

where p and q are hyperparameters referring to the truncation order of the Taylor terms used for the hard-coded and neural network features, respectively. This is illustrated below for $p = 1$ and $q = 1$,

$$\begin{aligned}\text{LocalTaylor}(\text{NeuralNet}, \mathbf{X}_C, X_T^i, \mathbf{Y}_C; 1, 1) = \\ Y_T^{n(i)} + \Delta X_T^{n(i)} D_T^{n(i)} + \\ \text{Linear}\left(\text{NeuralNet}(\mathbf{W}, \mathbf{X}_C, X_T^i, \mathbf{Y}_C)\right)\end{aligned} \quad (8)$$

the first two terms are a hard-coded first-order Taylor approximation, and the third is the neural network adjustment function taking in Taylor expansion terms ($\mathbf{W}$) as features. Below, we describe the terms $Y_T^{n(i)}, \Delta X_T^{n(i)}, D_T^{n(i)}, \mathbf{W}$, and how they are created.

We introduce the RepresentationExtractor to create the terms required to compute the terms for a Taylor series approximation. It takes the hyperparameter q, and we show it for $q = 1$:

$$\begin{aligned}\text{RepresentationExtractor}(X_T^i, \mathbf{X}_C, \mathbf{Y}_C; 1) = \\ X_T^{n(i)}, \mathbf{X}_C^{n(I)}, Y_T^{n(i)}, \mathbf{Y}_C^{n(I)}, \\ \Delta X_T^{n(i)}, \Delta \mathbf{X}_C^{n(I)}, \Delta \mathbf{Y}_C^{n(I)}, \mathbf{D}_C, \mathbf{D}_C^{n(I)}, D_T^{n(i)}\end{aligned} \quad (9)$$

where $X^{n(i)}$ (the i^{th} element of $\mathbf{X}^{n(I)}$) is the nearest already-seen neighbour of X^i, where the neighbour may come from either context set or previously seen values of the target set. Mathematically, $n(i) = \arg\min_{i' \neq i, i' \in C \text{ or } i' < i} ||X^i - X^{i'}||$. The remaining terms are defined as $\Delta \mathbf{X}^{n(I)} = \mathbf{X} - \mathbf{X}^{n(I)}$, $\Delta \mathbf{Y}^{n(I)} = \mathbf{Y} - \mathbf{Y}^{n(I)}$ and $\mathbf{D} = \frac{\Delta \mathbf{Y}^{n(I)}}{\Delta \mathbf{X}^{n(I)}}$, a data-based approximation of the derivative at $\mathbf{X}$, and $\mathbf{D}^{n(I)}$ is an

analogous data-based approximation of the derivative at $\mathbf{X}^{n(I)}$. These definitions hold for $\{\mathbf{X}, \mathbf{Y}\}$ whether they are from the context or target sets. We handle ties in the $\arg\min$ by randomly choosing one of the tied neighbours. We also positionally encode the $\mathbf{X}$ variables, similarly to Vaswani et al. (2017). We then define $\mathbf{W}$ as the concatenation of these features, $\mathbf{W} = \text{Concat}[\text{i for i in RepresentationExtractor}(X_T^i, \mathbf{X}_C, \mathbf{Y}_C; 1)]$.

Inspiration for dealing with noisy data also comes from local smoothing techniques, such as LOESS, which deal with noise by adjusting the size of the data subsets used to fit their local functions. Similarly, we can smooth out noise using averages based on different data subsets. For example, instead of estimating derivatives based only on the nearest-neighbor point, we can take an average of derivatives where each is calculated based on different points near the estimation point.

The estimation of derivatives can be generalised to higher-dimensions by using partial derivatives; we use it for our 2D regression experiments on the CelebA (Liu et al., 2015) and EMNIST (Cohen et al., 2017) datasets.

3.2. MHA-X block

It is worth drawing inspiration from how GPs model processes given they are well-liked probabilistic tools used to model various continuous problems. In fact, they are even included in scikit-learn (Pedregosa et al., 2011). Although they model a restricted class of functions, based on their wide usage, it is evident that the classes of functions which they do model are important to users.

For a GP, the mean of its predictions of Y_T^i is simply a linear combination of the contextual information Y_C. This is seen from their predictive distribution for Y_T^i at X_T^i conditioned on $\mathbf{Y}_C$ and $\mathbf{X}_C$, $Y_T^i \sim N(A, B)$, where

$$A = K(X_T^i, \mathbf{X}_C)K(\mathbf{X}_C, \mathbf{X}_C')\mathbf{Y}_C \tag{10}$$

where K is a kernel specifying the covariance and is a function only of $\mathbf{X}$. Therefore, equation (10) is a linear weighting of $\mathbf{Y}_C$, with the weighting done by the kernel, which is a non-linear function of the $\mathbf{X}$ values.

GPs weigh contextual information in a contrasting manner to that in a standard GPT-style attention model, where non-linear combinations of $\{\mathbf{X}_C, \mathbf{Y}_C\}$ (referred to as values, V) are weighted based on non-linear functions of *both* $\mathbf{X}_C$ and $\mathbf{Y}_C$ (referred to as the queries Q and keys K) as follows

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{11}$$

where $V \in \mathbb{R}^{n_C \times d_v}$, $K \in \mathbb{R}^{n_C \times d_k}$, $Q \in \mathbb{R}^{n_C \times d_k}$.

This standard attention's (MHA-XY) approach to weighting is important for language tasks as considerable weight should still be put on distant words if they have similar semantic meaning to closer-by words. But it is not an obvious requirement for regression tasks seen by how GPs are so useful despite this mechanism being absent.

We propose the **MHA-X block**, which allows the mean prediction of Y_T^i to be modelled as a linear weighting of $\mathbf{Y}_C$, with the weighting being a non-linear function, learnt using a neural network, of just $\mathbf{X}_C$ and X_T^i.

An MHA-X block with N layers is composed of $N-1$, multi-head attention units

$$\text{MultiHead}(f(\mathbf{X}_C), f(\mathbf{X}_C), f(\mathbf{X}_C)) = \text{Concat}(\text{head}_1, ..., \text{head}_h)R^O \tag{12}$$

with $\text{head}_i = \text{Attention}(QR_i^Q, KR_i^K, VR_i^V)$, where $f(\mathbf{X}_C)$ are non-linear functions only of features derived from $\mathbf{X}_C$, and $R_i^Q, R_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $R_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $R^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. The last layer of the MHA-X block is composed of the multi-head attention unit:

$$\text{MultiHead}(g(\mathbf{X}_C), g(\mathbf{X}_C), \mathbf{Y}_C) \tag{13}$$

where $g(\mathbf{X}_C)$ is the output of the previous layer in the MHA-X block.

3.3. Training

Our model is trained by maximising the likelihood, $\Pr(\mathbf{Y}_T|\mathbf{X}_T, \mathbf{X}_C, \mathbf{Y}_C)$. Two key differences exist between our training and a standard transformer-decoder model like GPT. The first is using a shuffling method to encourage consistency through training. For tasks where we wish to prioritise target-equivariance, we use a shuffling and Expectation approach similar to Nguyen & Grover (2022): during training, we maximise the likelihood of $p(\mathbf{Y}_{\pi(T)}|\mathbf{X}_{\pi(T)}, \mathbf{Y}_C, \mathbf{X}_C)$ where π are permutations of the target set which are chosen randomly for each training batch. The Taylorformer enforces context invariance (equation (2)) through its architecture.

The second difference is an approach to prevent data leakage when training to predict in arbitrary orders. To predict points in arbitrary orders, we need a mechanism to query points at arbitrary $\mathbf{X}$ locations without revealing the corresponding $\mathbf{Y}$ value during training. The classic GPT-style mask is not capable of this. Nguyen & Grover (2022) introduced a mask and target-padding approach that deals with this. Our process is an alternative approach. The main difference is that the number of multiplications required for the scaled dot-product in equation (11) is $O((n_C+2n_T)^3)$ for Nguyen & Grover (2022), whereas ours is only $O((n_C + n_T)^3)$ operations, though we reiterate that efficiency is not our work's focus. Our mechanism combines separate keys and queries for the attention mechanism with specific masking.

The process is different for MHA-X and MHA-XY and is detailed below.

MHA-XY For the first layer in the MHA-XY block, there are $n_C + n_T$ queries, keys and values. The queries are

$$Q_i = \begin{cases} (X^{\text{fe},i}, Y^{\text{fe},i}, Y^{\text{seen},i}, 1) & \text{if } i \in C \\ (X^{\text{fe},i}, 0, Y^{\text{seen},i}, 0) & \text{if } i \in T \end{cases} \tag{14}$$

where $X^{\text{fe},i}$ are features which are only functions of X^i and X^j for $j \in C \cup \{j : j < i\}$, $Y^{\text{fe},i}$ are features which are functions of Y^i, X^i, Y^j and X^j for $j \in C \cup \{j : j < i\}$, and $Y^{\text{seen},i}$ are features which are functions of X^i, Y^j and X^j for $j \in C \cup \{j : j < i\}$. For this work, $X^{\text{fe},i} = [X^i, X^{n(i)}, \Delta X^{n(i)}]$, $Y^{\text{fe},i} = [Y^i, \Delta Y^{n(i)}, D^i]$ and $Y^{\text{seen},i} = [Y^{n(i)}, D^{n(i)}]$, where these terms are defined in section 3.1. We mask $Y^{\text{fe},i}$ so that when making predictions for the target set, we can query by Q_i (as we need to know the X information of the point we are looking to predict) without revealing the target value during training. The final item is a label indicating whether the variables $Y^{\text{fe},i}$ which contain Y^i are masked (set to zero) or not. The keys and values are

$$K_i = V_i = (X^{\text{fe},i}, Y^{\text{fe},i}, Y^{\text{seen},i}, 1) \tag{15}$$

The masking mechanism is designed so that: (1) the context points only attend to themselves and (2) target points only attend to the previous target points and the context points. Figure 2b shows an example mask for $n_C = 3$ and $n_T = 3$.

For subsequent layers in MHA-XY, the queries, keys and values are the outputs of the previous MHA-XY layers.

MHA-X For the first N-1 layers, the inputs are the outputs of the previous layer, where the inputs to the first layer are

$$Q_i = K_i = V_i = X^{\text{fe},i} \tag{16}$$

For the final layer, the inputs are $Q_i = K_i = O_i$, where O_i is the output of the previous MHA-X layer, and $V_i = Y_i$. The same mask as for MHA-XY is used for MHA-X.

4. Experiments

To assess the Taylorformer as a model for continuous processes, we evaluate on key tasks in the NP literature: 1D regression and 2D regression (tested on image completion). To assess it on time-series, we evaluate it on three key forecasting tasks: electricity consumption, the load of electricity transformers and exchange rate.

For these experiments, it was empirically better to truncate the explicit Taylor expansion terms to the zeroth order (setting $p = 0$) and keep the first order terms in $\mathbf{W}$ (setting $q = 1$) with using the approximate derivative of the nearest neighbour. Therefore, we model the mean of Y_T^i as $\mu_{Y_T^i} = \text{LocalTaylor}(\text{NeuralNet}, \mathbf{X}_C, X_T^i, \mathbf{Y}_C; 0, 1)$. We approximate the expectation using a single-sample Monte Carlo.

The Taylorformer was trained on a 32GB NVIDIA V100S GPU. Further implementation details are below.

4.1. Neural Process tasks

Datasets For 1D regression, we generate three datasets corresponding to a meta-learning setup — each dataset contains 100K sequences with $\mathbf{X} \sim U[-2, 2]$. We query Gaussian Process (GPs) at those $\mathbf{X}$ to give $\mathbf{Y}$ with a random selection of kernel hyper-parameters as specified in Appendix A. Each dataset uses either the RBF, Matérn or Periodic kernel for GP generation. The hold-out dataset includes sequences generated in the same manner as the training data.

For 2D regression, we use two datasets. The first comes from the balanced EMNIST (Cohen et al., 2017) dataset, which contains monochrome images. Each pixel $y_i \in \mathbf{R}^1$ is scaled to be in [-0.5, 0.5] and $x_i \in \mathbf{R}^2$ with each dimension scaled to be in [-1, 1]. We train on only a subset of available classes (0-9, corresponding to handwritten numbers). There are two hold-out sets, the first includes unseen images from the same classes seen during training. The second includes images from unseen classes (10-46, corresponding to handwritten letters). The second dataset is CelebA (Liu et al., 2015): coloured images of celebrity faces with each pixel $y_i \in \mathbf{R}^3$ with each dimension scaled to be in [-0.5, 0.5] and $x_i \in \mathbf{R}^2$ with each dimension scaled to be in [-1, 1].

Implementation details Models were trained for 250K, 71K and 125K iterations with 32, 64, and 64 batch sizes for 1D regression, EMNIST and CelebA, respectively. Optimization was done using Adam with a 10^{-4} learning rate. For 1D-regression, $n_C \sim U[3, 97]$ and $n_T = 100 - n_C$. For EMNIST and CelebA $n_C \sim U(6, 200)$ and $n_T \sim U(3, 197)$. Full implementation details are found in the Appendix.

Results. The meta-learning 1D-regression and image completion experiments were used extensively in the NP literature (Garnelo et al., 2018b; Gordon et al., 2019; Kim et al., 2019; Lee et al., 2020; Nguyen & Grover, 2022; Wang & Van Hoof, 2020), and we compare to NP (Garnelo et al., 2018b), ANP (Kim et al., 2019) and TNP (Nguyen & Grover, 2022). Table 1 shows that Taylorformer improves on the other methods for 5 out of 6 tasks. Figure 4 in Appendix B shows that for all 1D regression tasks, the validation loss is lowest for the Taylorformer. For the 2D regression, we perform best on both EMNIST tasks, but there is no

improvement over the TNP for CelebA. We hypothesise that noise in the CelebA dataset may benefit from taking averages of derivatives.

4.2. Forecasting

Datasets We use three datasets: 1) hourly electricity consumption data from 2013 to 2014.[1] We pick client number 322 out of the 370 available clients. 2) 15-minute Electricity transformers load (ETT) (Zhou et al., 2020) and 3) Daily exchange rate (Lai et al., 2017) from 1990-2016 (we pick the country denoted OT). For each dataset, we run four experiments in which each sequence has a fixed length context set, $n_C = 96$, (with randomly selected starting position) and a prediction length $n_T \in \{96, 192, 336, 720\}$. We standardise $y_i \in \mathbf{R}$ for all experiments. $x_i \in \mathbf{R}$ is scaled to be in [-1, 1] in each sequence — only relative time matters to us.

Implementation details Models were trained for 40K iterations with batch sizes of 32. Optimization was done using Adam with a 10^{-4} learning rate. All datasets are split chronologically into training, validation and test sets by the ratios 72:8:20, 69:11:20, and 69:11:20, respectively. Full implementation details are in Appendix A.

Results The electricity, ETT and exchange rate are used extensively in the literature, and we compare to Autoformer (Wu et al., 2021), Informer (Zhou et al., 2020) and TNP (Nguyen & Grover, 2022). For Autoformer and Informer, we keep the exact setup used in the papers concerning network and training parameters so the model is trained on mean-squared-error. We train TNP to maximise log-likelihood, and we set the number of network parameters to match ours since their paper does not implement these tasks. Further, we run a hyper-parameter search for TNP. We do not compare to the classic ARIMA model as it is outperformed by Autoformer and Informer on these datasets.

Table 2 shows that Taylorformer is notably better in terms of MSE for all forecasting tasks. We outperform the autoregressive TNP and the batch-generating Autoformer and Informer with 21-34%, 95-99% and 14-18% reductions in MSE compared to the closest model for the ETT, exchange and electricity tasks, respectively. Examples of our generated sequences can be seen in Figures 1 and 6 (Appendix). The likelihood results show a similar trend and are provided in Appendix B.

[1] https://drive.google.com/file/d/1jinfTAApPyuyvW1P1hUDpI3rl0Jq8in1/view?usp=share_link

4.3. Ablation study

We study what contributions are key to the final model performance by performing a 1D regression task on four equally sized models: a base transformer-decoder (just MHA-XY units), a model with only MHA-X, a model with the Local-Taylor wrapped around the base model, and a model with both contributions. Results in Figure 5 indicate that the combination of both contributions yields the best outcome.

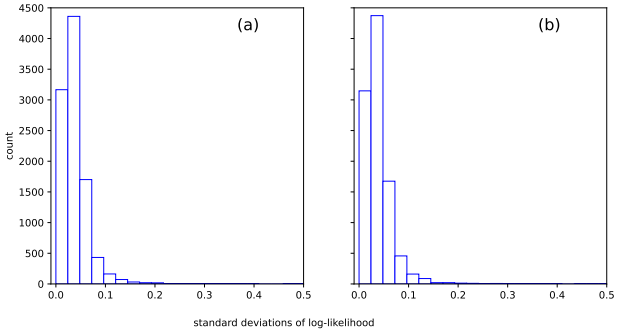

Figure 3. By shuffling the target variables given the context during training, we drive the model to be approximately target equivariant. If all log-likelihood scores are equal, their standard deviation will be zero. The histograms show these standard deviations for training with (a) one permuted sample and (b) five permuted samples on the RBF task. We can see that the models are 'close' to consistency. Furthermore, for the specific RBF task, one permuted sample during training seems suitable to drive consistency.

4.4. Evaluating consistency

Our model attempts to approximate target equivariance via shuffled training. We evaluate the target equivariance (equation (1)) of Taylorformer by computing the log-likelihood for sequences with permuted target points and taking the standard deviation. If consistent, all the likelihoods would be the same and the standard deviation would be zero.

Concretely, for each of the H sequences in the hold-out set, the target set is permuted 40 times. The log-likelihoods of these permuted sequences are calculated, and the standard deviations of these are computed. This results in an array of H standard deviations. This can then be visualised with a histogram. A consistent model would show a histogram only composed of points at zero (as the calculated log-likelihoods of a given sequence would not change for the differing target set permutations).

We show the results of this procedure for the RBF experiment in Figure 3a. Our model does not achieve consistency but comes close. The mean of the standard deviations is 0.038, with a 95% confidence interval of [0.002, 0.105]. We compare this to an identical model trained with 5 permutation samples for each training sequence (instead of 1) to

Table 1. Log-likelihood on a test set. Higher is better. Taylorformer outperforms the SOTA TNP in 5/6 1D and 2D regression tasks. Each model is run five times to report log-likelihood with standard deviation results.

		Log-Likelihood			
		Taylorformer	TNP	ANP	NP
1D	RBF	$\mathbf{4.13 \pm 0.03}$	3.50 ± 0.05	0.96 ± 0.02	0.24 ± 0.01
	Matern	$\mathbf{3.87 \pm 0.05}$	3.22 ± 0.05	1.07 ± 0.01	0.40 ± 0.02
	Periodic	$\mathbf{0.31 \pm 0.02}$	0.13 ± 0.04	-0.85 ± 0.05	-1.56 ± 0.00
2D	EMNIST-seen	$\mathbf{2.18 \pm 0.01}$	1.59 ± 0.00	0.82 ± 0.04	0.62 ± 0.01
	EMNIST-unseen	$\mathbf{1.90 \pm 0.03}$	1.47 ± 0.01	0.68 ± 0.08	0.44 ± 0.00
	CelebA	4.00 ± 0.02	$\mathbf{4.13 \pm 0.02}$	2.78 ± 0.03	2.30 ± 0.01

Table 2. MSE on a test set. Lower is better. Taylorformer outperforms three state-of-the-art models on three forecasting tasks (details in text). Each task is trained and evaluated with different prediction lengths $n_T \in \{96, 192, 336, 720\}$ given a fixed 96-context length. Each model is run five times to report log-likelihood with standard deviation results.

		MSE			
		Taylorformer	TNP	Autoformer	Informer
ETT	96	$\mathbf{0.00030 \pm 0.00001}$	0.00041 ± 0.00004	0.43 ± 0.21	0.40 ± 0.20
	192	$\mathbf{0.00029 \pm 0.00000}$	0.00044 ± 0.00008	0.57 ± 0.26	0.65 ± 0.30
	336	$\mathbf{0.00030 \pm 0.00001}$	0.00038 ± 0.00001	0.67 ± 0.32	0.72 ± 0.34
	720	$\mathbf{0.00029 \pm 0.00000}$	0.00039 ± 0.00004	0.87 ± 0.42	0.85 ± 0.40
Exchange	96	$\mathbf{0.002 \pm 0.000}$	0.040 ± 0.030	0.41 ± 0.20	0.57 ± 0.26
	192	$\mathbf{0.002 \pm 0.000}$	0.079 ± 0.035	0.59 ± 0.28	1.53 ± 0.80
	336	$\mathbf{0.002 \pm 0.000}$	0.067 ± 0.025	0.96 ± 0.46	3.28 ± 1.4
	720	$\mathbf{0.001 \pm 0.000}$	0.152 ± 0.091	1.08 ± 0.51	1.47 ± 0.76
Electricity	96	$\mathbf{0.036 \pm 0.000}$	0.042 ± 0.002	0.130 ± 0.039	0.133 ± 0.005
	192	$\mathbf{0.037 \pm 0.000}$	0.045 ± 0.001	0.117 ± 0.001	0.148 ± 0.011
	336	$\mathbf{0.040 \pm 0.001}$	0.048 ± 0.002	0.143 ± 0.011	0.150 ± 0.013
	720	$\mathbf{0.039 \pm 0.001}$	0.048 ± 0.003	0.216 ± 0.040	0.129 ± 0.012

better approximate equation. Figure 3b shows that there is no obvious improvement from this, which is supported by the mean of the standard deviations also being 0.038, with a 95% confidence interval of [0.002, 0.110].

5. Discussion

Limitations Two limitations of the LocalTaylor approach are as follows: first, when dealing with noisy data, there needs to be a clear recommendation on how many samples of derivative estimations to use when computing the average derivative, for example. This is just a hyperparameter for now. Second, it would be better if we could also *learn* how many of the hard-coded Taylor approximation terms to use instead of it just being a hyperparameter. An extension to augment the LocalTaylor approach would be to use a weighting function to more strongly weight data points closer to the point of estimation.

Taylorformer is an autoregressive model so is slower to generate predictions than batch-generation models. Moreover, the extra steps in the RepresentationExtractor (such as the nearest neighbour search) have an associated computational cost affecting training and inference — one which is absent from the other NP models. Taylorformer would need to be better-suited for tasks where fast sampling and generation of sequences are required. A one-shot model such as the Autoformer (Wu et al., 2021) may be more suitable.

Another limitation is that autoregressive models like ours are prone to error accumulation, unlike batch-generation models. It is well-known in the dynamical modeling literature (Lorenz & Hilborn, 1995) that predictions from an autoregressive model slowly deviate from target values. A fully consistent process would address this, but as noted in section 4.4, Taylorformer does not have strict target equivariance. Nevertheless, the shuffling approach to training encourages consistency and is one part of what we expect to be the solution to the error accumulation issue.

Conclusions The Taylorformer is built on two key components: the MHA-X block and the LocalTaylor wrapper. Although the MHA-X block relies on using attention-based models, the LocalTaylor approach could be easily deployed for use in other classes of models too, such as RNNs, if preferred by the modeller.

References

Brenowitz, N. D. and Bretherton, C. S. Prognostic validation of a neural network unified physics parameterization. *Geophysical Research Letters*, 45(12):6289–6298, 2018.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. J., and Weller, A. Rethinking attention with performers. *ArXiv*, abs/2009.14794, 2020.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Gagne, D. J., Christensen, H. M., Subramanian, A. C., and Monahan, A. H. Machine learning for stochastic parameterization: Generative adversarial networks in the lorenz'96 model. *Journal of Advances in Modeling Earth Systems*, 12(3):e2019MS001896, 2020.

Garnelo, M., Rosenbaum, D., Maddison, C. J., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D. J., and Eslami, S. M. A. Conditional neural processes. *ArXiv*, abs/1807.01613, 2018a.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S. M. A., and Teh, Y. W. Neural processes. *ArXiv*, abs/1807.01622, 2018b.

Gordon, J., Bruinsma, W. P., Foong, A. Y., Requeima, J., Dubois, Y., and Turner, R. E. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 13–18 Jul 2020.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, S. M. A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. Attentive neural processes. *ArXiv*, abs/1901.05761, 2019.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451, 2020.

Lai, G., Chang, W.-C., Yang, Y., and Liu, H. Modeling long- and short-term temporal patterns with deep neural networks. *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2017.

Lee, J., Lee, Y., Kim, J., Yang, E., Hwang, S. J., and Teh, Y. W. Bootstrapping neural processes. *Advances in neural information processing systems*, 33:6606–6615, 2020.

LI, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., and Yan, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *ArXiv*, abs/1907.00235, 2019.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, Ł., and Shazeer, N. Generating wikipedia by summarizing long sequences. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Liu, Y., Kutz, J. N., and Brunton, S. L. Hierarchical deep learning of multiscale differential equation time-steppers. *Philosophical Transactions of the Royal Society A*, 380 (2229):20210200, 2022.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Lorenz, E. N. and Hilborn, R. C. The essence of chaos. 1995.

Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3276–3285. PMLR, 10–15 Jul 2018.

Nguyen, T. and Grover, A. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. In Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., and Sabato, S. (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16569–16594. PMLR, 17–23 Jul 2022.

Parthipan, R., Christensen, H. M., Hosking, J. S., and Wischik, D. J. Using probabilistic machine learning to better model temporal patterns in parameterizations: a case study with the lorenz 96 model. *EGUsphere*, pp. 1–27, 2022.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62:352–364, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Wang, Q. and Van Hoof, H. Doubly stochastic variational inference for neural processes with hierarchical latent variables. In *International Conference on Machine Learning*, pp. 10018–10028. PMLR, 2020.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768, 2020.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Neural Information Processing Systems*, 2021.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297. Curran Associates, Inc., 2020.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. *ArXiv*, abs/2012.07436, 2020.

Table 3. Network and training details for forecasting tasks ETT/ exchange/ electricity. If a cell contains one value then all tasks share the same parameter and otherwise the cell is split by a slash respectively. * indicates that early stopping was used. For the electricity dataset the total number of parameters for Autoformer and Informer is reduced (relative to original implementation) to improve their results and match the number of Taylorformer parameters.

Models	**Taylorformer**	TNP	Autoformer	Informer
Network parameters				
Dropout rate	0.05/0.05/0.25	0.05/0.1/0.25	0.05	0.05
# Layers	attention heads 6/8/6	attention heads 7/6/4	2 encoder, 1 decoder	2 encoder, 1 decoder
# Parameters	$90K$/$90K$/$200K$	$90K$/$90K$/$200K$	$42m$/$42m$/$200K$	$45m$/$45m$/$200K$
Training parameters				
# Iterations	$40K^*$	$40K^*$	$15K^*$	$15K^*$
Batch size	32	32	32	32
Learning rate	$3e^{-4}$	$3e^{-4}$	$1e^{-4}$	$1e^{-4}$
Optimizer	Adam	Adam	Adam	Adam
Hardware				
Processor	32GB NVIDIA V100S GPU		120GB NVIDIA T4x4 GPU	

A. Experimental Details

The NP, ANP and TNP architectures follow closely from those used by (Nguyen & Grover, 2022). The TNP hyperparameters are the same as in (Nguyen & Grover, 2022) for the 1D regression and image completion tasks. For the electricity forecasting task, we set the TNP dropout to 0.25 to reduce overfitting. Specifics for the ANP and NP are provided below. The TNP, NP and ANP all use Relu activation functions for their Dense layers.

The Autoformer and Informer architectures follow exactly those from the official code base of (Wu et al., 2021) — code can be found in `https://github.com/thuml/Autoformer`. The number of parameters in both models are reduced to make a fair comparison with our model. This practically means that we set d_model $= 20$ and dff $= 512$ (notation as used in the original work).

In all cases, during training we monitored performance on a validation set and saved the model weights for the iterations they performed best (in terms of validation log-likelihood).

A.1. 1D Regression

Data generation. For a GP with a given kernel, first we sampled random kernel hyparameters (distributions are given below). Next, we generated $\mathbf{X} \sim U[-2, 2]$, where $\mathbf{X} \in \mathbb{R}^{200\text{x}1}$ and then queried the GP at those locations to give $\mathbf{Y}$. From this, n_C context pairs and n_T target ones were randomly selected, with $n_C \sim U(3, 97)$ and $n_T = 100 - n_C$. This process was repeated to generate all the sequences. The hyperparameters were drawn from the following distributions: for the RBF kernel, $k(x, x') = s^2 \exp(-|x - x'|^2/2l^2)$, $s \sim U(0.1, 1.0)$ and $l \sim U(0.1, 0.6)$. For the Matérn 5/2, $k(x, x') = (1 + \sqrt{5}d/l + 5d^2/(3l^2)) \exp(-\sqrt{5}d/l), d = |x - x'|$, $l \sim U(0.3, 1.0)$. For the periodic kernel, $k(x, x') = \exp(-2\sin^2(\pi|x - x'|^2/p)/l^2)$, $l \sim U(0.1, 0.6)$ and $p \sim U(0.5, 1.0)$.

Training and testing. Three training sets (for each kernel) were generated, comprising 100,000 sequences each. The test set (for each kernel) comprised 10,000 unseen sequences. All models were trained for 250,000 iterations, with each batch comprising 32 sequences. We used Adam with a learning rate of 10^{-4}.

Architectures. For the NP, the latent encoder has 3 dense layers before mean aggregation, and 2 after it. The deterministic encoder has 4 dense layers before mean aggregation. The decoder has 4 dense layers. All dense layers are of size 128. For the ANP, the latent encoder and decoder are the same as for the NP. The deterministic encoder has an additional 2 dense layers to transform the key and queries. The multihead attention has 8 heads. All dense layers are of size 128.

Table 4. Negative log-likelihood on a test set. Lower is better. Taylorformer outperforms three state-of-the-art models on three forecasting tasks (details in text). Each task is trained and evaluated with different prediction lengths $n_T \in \{96, 192, 336, 720\}$ given a fixed 96-context length. Each model is run five times to report means with standard deviation results.

		Negative Log-Likelihood			
		Taylorformer	TNP	Autoformer	Informer
ETT	96	$\mathbf{-2.69 \pm 0.02}$	-2.47 ± 0.05	0.65 ± 0.16	0.61 ± 0.16
	192	$\mathbf{-2.68 \pm 0.01}$	-2.42 ± 0.14	0.87 ± 0.13	0.92 ± 0.14
	336	$\mathbf{-2.64 \pm 0.01}$	-2.48 ± 0.02	0.88 ± 0.15	0.87 ± 0.14
	720	$\mathbf{-2.56 \pm 0.01}$	-2.38 ± 0.03	1.38 ± 0.07	1.01 ± 0.13
Exchange	96	$\mathbf{-1.31 \pm 0.04}$	0.27 ± 1.11	0.65 ± 0.15	0.86 ± 0.11
	192	$\mathbf{-1.21 \pm 0.03}$	0.38 ± 0.40	0.87 ± 0.13	1.33 ± 0.15
	336	$\mathbf{-1.05 \pm 0.04}$	1.21 ± 0.47	1.19 ± 0.09	1.80 ± 0.07
	720	$\mathbf{-0.68 \pm 0.02}$	0.94 ± 0.55	1.05 ± 0.13	1.52 ± 0.06
Electricity	96	$\mathbf{-0.53 \pm 0.01}$	-0.42 ± 0.03	0.38 ± 0.13	0.41 ± 0.02
	192	$\mathbf{-0.50 \pm 0.00}$	-0.40 ± 0.02	0.35 ± 0.01	0.46 ± 0.04
	336	$\mathbf{-0.41 \pm 0.03}$	-0.28 ± 0.08	0.44 ± 0.04	0.47 ± 0.05
	720	$\mathbf{-0.44 \pm 0.03}$	-0.25 ± 0.13	0.64 ± 0.09	0.39 ± 0.05

A.2. Image completion

Data generation. For a given image, we sampled the 2D coordinates of a pixel, and rescaled to [-1,1] to compose $\mathbf{X}$ and rescaled the pixel value to [-0.5,0.5] to compose $\mathbf{Y}$. The number of context, n_C, and target, n_T, points are chosen randomly for each image, with $n_C \sim U(6, 200)$ and $n_T \sim U(3, 197)$.

Training and testing. For EMNIST, all models were trained for 200 epochs (71,000 iterations with a batch size of 64). For CelebA, all were trained for 45 epochs (125,000 iterations with a batch size of 64). In both cases, we used Adam with a learning rate of 10^{-4}.

Architectures. For EMNIST, in the NP, the latent encoder has 5 dense layers before mean aggregation, and 2 after it. The deterministic encoder has 5 dense layers before mean aggregation. The decoder has 4 dense layers. In the ANP, the latent encoder has 3 dense layers before mean aggregation and 3 after it. The deterministic encoder has 3 dense layers. The decoder has 4 dense layers. The deterministic encoder has 3 dense layers followed by a self-attention layer with 8 heads, then 3 dense layers to transform the keys and queries. The final cross-attention has 8 heads. All dense layers are of size 128.

For CelebA, in the NP, the latent encoder has 6 dense layers before mean aggregation, and 3 after it. The deterministic encoder has 6 dense layers before mean aggregation. The decoder has 5 dense layers. In the ANP, the latent encoder has 4 dense layers before mean aggregation and 3 after it. The deterministic encoder has 4 dense layers. The decoder has 4 dense layers. The deterministic encoder has 4 dense layers followed by a self-attention layer with 8 heads, then 3 dense layers to transform the keys and queries. The final cross-attention has 8 heads. All dense layers are of size 128.

A.3. Electricity forecasting

Training and testing. All models were trained for 40,000 iterations with a batch size of 32. We used Adam with a learning rate of 10^{-4}.

B. Further results

Validation loss for 1D regression. For all the 1D regression experiments with GPs, we see in Figure 4 that the validation set negative log-likelihood is lowest for the Taylorformer compared to other Neural Process models. The TNP has a particularly noisy loss curve.

Log-likelihood results for forecasting experiments. Please refer to table 4 for results.

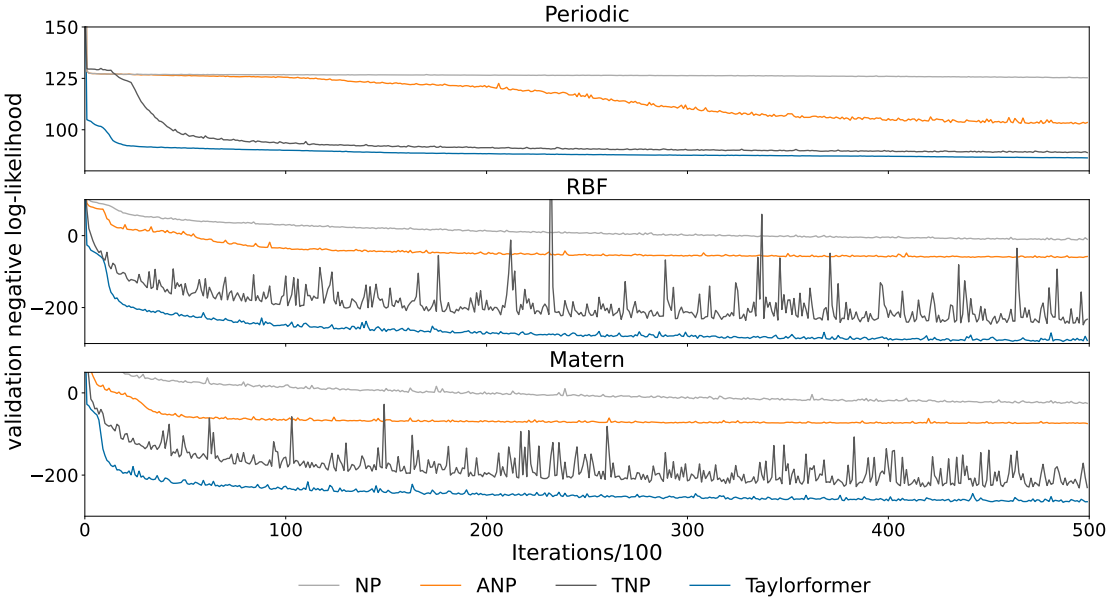

Figure 4. Validation set negative log-likelihood (NLL). Lower is better. Our Taylorformer outperforms NP(Garnelo et al., 2018b), ANP (Kim et al., 2019) and TNP (Nguyen & Grover, 2022) on the meta-learning 1D regression task (see task details in the main text).

Ablation study. The results of our ablation study are shown in Figure 5. We see that it is the combination of both the LocalTaylor approach and the MHA-X attention block which yields the best outcome.

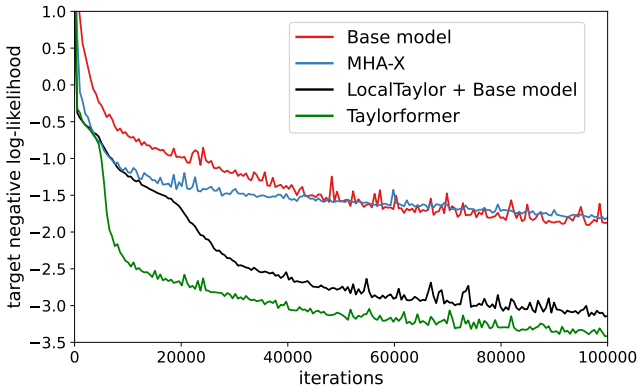

Figure 5. Ablations for our model showing that using both the LocalTaylor wrapper and the MHA-X together (green line) contributes to the improved results. This is shown for a 1D regression task (GP RBF kernel).

Electricity forecasting samples. We show samples from the Taylorformer and the TNP for the electricity forecasting task for target-336-context-96. Both models do well but the Taylorformer tracks the peaks and troughs better.

C. Code

Our code can be found at `https://github.com/oremnirv/Taylorformer`.

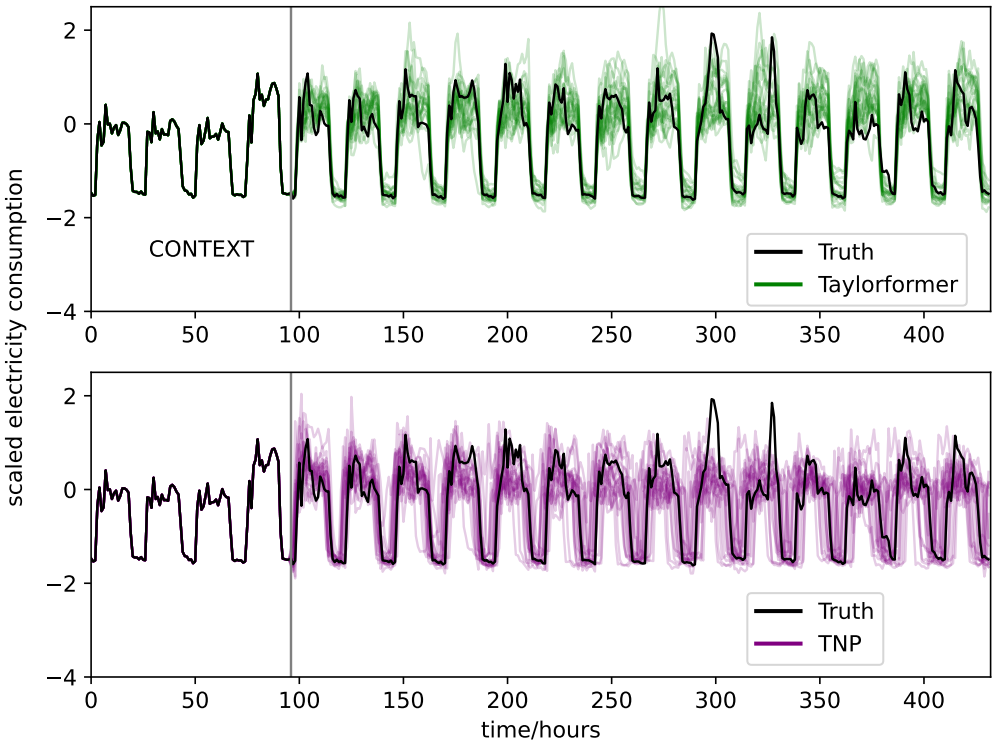

Figure 6. 20 samples from our model and the TNP on the electricity task. Ours tracks the periodic trend remarkably well. Although the TNP tracks the periodic trend, it struggles to maintain the appropriate periodicity, especially at longer lead times. The Taylorformer appears to have better-calibrated uncertainty, too: it is more uncertain about the peaks in consumption, and so can produce time-series samples that can capture the large peak around hour 300, unlike the TNP.

D. Model implementation

D.1. Embedding

We follow a similar formulation to the embedding from Vaswani et al. (2017), but adapt it to continuous values.

$$PE(t, 2i) = \sin(\frac{t/\delta_t}{(tmax/\delta_t)^{2i/d}}) \tag{17}$$

$$PE(t, 2i+1) = \cos(\frac{t/\delta_t}{(tmax/\delta_t)^{2i/d}}) \tag{18}$$

where d is the total embedding dimension and i runs from 1 to d. We will denote the vector of all embedding dimensions for a specific t as $\mathrm{PE}(t)$.

D.2. Model implementation

We present a generic implementation of our training for batch data in algorithm 1. The algorithm notation follows the notation presented in the main text. We denote the mask as M. It follows the same masking described in the main text and in Figure 2. The $\mathrm{PositionalEncoding}$ is described in D.1. By $\mathrm{Split}(\mathbf{W})$ we refer to the operation of extracting subsets of variables from $\mathbf{W}$.

Algorithm 1 Model implementation for batch data – training phase

Input: data $\mathbf{X} = [\mathbf{X}_C, \mathbf{X}_T]$, $\mathbf{X} \in \mathbf{R}^{B\times\ell\times 1}$, $\mathbf{Y} = [\mathbf{Y}_C, \mathbf{Y}_T]$, $\mathbf{Y} \in \mathbf{R}^{B\times\ell\times\alpha}$, mask M; batch size B; context length n_C; target length n_T; sequence length ℓ; embedding dimension d; output dimension d_O; MHA layers N.

$\mathbf{X}' = \text{PositionalEncoding}(\mathbf{X})$ $\{\mathbf{X}' \in \mathbf{R}^{B\times\ell\times d}\}$

$\mathbf{W} = \text{Concat}[\text{i for i in RepresentationExtractor}(X_T^i, \mathbf{X}_C, \mathbf{Y}_C; q)]$ $\{\mathbf{W} \in \mathbf{R}^{B\times\ell\times(d+\alpha+4q)}\}$

$\mathbf{X}^{\text{fe}}, \mathbf{Y}^{\text{fe}}, \mathbf{Y}^{\text{seen}} = \text{Split}(\mathbf{W})$ $\{\mathbf{X}^{\text{fe}} \in \mathbf{R}^{B\times\ell\times(d+2q)}, \mathbf{Y}^{\text{fe}} \in \mathbf{R}^{B\times\ell\times(\alpha+2\alpha q)}, \mathbf{Y}^{\text{seen}} \in \mathbf{R}^{B\times\ell\times(2\alpha q)}\}$

$\mathbf{Q}_C{}^{\text{MHA-XY}} = [\mathbf{X}_C^{\text{fe}}, \mathbf{Y}_C^{\text{fe}}, \mathbf{Y}_C^{\text{seen}}, 1]$ $\{\mathbf{Q}_C{}^{\text{MHA-XY}} \in \mathbf{R}^{B\times n_C\times(4\alpha q+\alpha+2q+d+1)}\}$

$\mathbf{Q}_T{}^{\text{MHA-XY}} = [\mathbf{X}_T^{\text{fe}}, 0, \mathbf{Y}_T^{\text{seen}}, 0]$ $\{\mathbf{Q}_T{}^{\text{MHA-XY}} \in \mathbf{R}^{B\times n_T\times(4\alpha q+\alpha+2q+d+1)}\}$

$\mathbf{Q}^{\text{MHA-XY}} = \text{Concat}[\mathbf{Q}_C, \mathbf{Q}_T]$ $\{\mathbf{Q}^{\text{MHA-XY}} \in \mathbf{R}^{B\times\ell\times(4\alpha q+\alpha+2q+d+1)}\}$

$\mathbf{K}^{\text{MHA-XY}} = [\mathbf{X}^{\text{fe}}, \mathbf{Y}^{\text{fe}}, \mathbf{Y}^{\text{seen}}, 1]$ $\{\mathbf{K}^{\text{MHA-XY}} \in \mathbf{R}^{B\times\ell\times(4\alpha q+\alpha+2q+d+1)}\}$

$\mathbf{V}^{\text{MHA-XY}} = \mathbf{K}^{\text{MHA-XY}}$

$\mathbf{Q}^{\text{MHA-X}} = \mathbf{K}^{\text{MHA-X}} = \mathbf{V}^{\text{MHA-X}} = \mathbf{X}^{\text{fe}}$

$O_X = \text{MultiHead}(\mathbf{Q}^{\text{MHA-X}}, \mathbf{K}^{\text{MHA-X}}, \mathbf{V}^{\text{MHA-X}}, \text{mask} = M)$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \text{Dense}(\mathbf{Q}^{\text{MHA-X}})))$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \text{Linear}(\text{Dense}(O_X))))$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{MultiHead}(\mathbf{Q}^{\text{MHA-XY}}, \mathbf{K}^{\text{MHA-XY}}, \mathbf{V}^{\text{MHA-XY}}, \text{mask} = M)$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \text{Dense}(\mathbf{Q}^{\text{MHA-XY}})))$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \text{Linear}(\text{Dense}(O_{XY}))))$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

for $j = 1$ **to** $N - 2$ **do**

$\mathbf{Q}^{\text{MHA-X}} = O_X$, $\mathbf{Q}^{\text{MHA-XY}} = O_{XY}$

$O_X = \text{MultiHead}(O_X, O_X, O_X, \text{mask} = M)$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \mathbf{Q}^{\text{MHA-X}}))$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \text{Linear}(\text{Dense}(O_X)))$ $\{O_X \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{MultiHead}(O_{XY}, O_{XY}, O_{XY}, \text{mask} = M)$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \mathbf{Q}^{\text{MHA-XY}}))$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \text{Linear}(\text{Dense}(O_{XY})))$ $\{O_{XY} \in \mathbf{R}^{B\times\ell\times d_O}\}$

end for

$\mathbf{Q}^{\text{MHA-X}} = O_X$, $\mathbf{Q}^{\text{MHA-XY}} = O_{XY}$

$O_X = \text{MultiHead}(O_X, O_X, \mathbf{Y}, \text{mask} = M)$ $\{O_X \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \mathbf{Q}^{\text{MHA-X}}))$ $\{O_X \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O_X = \text{LayerNorm}(\text{Add}(O_X, \text{Linear}(\text{Dense}(O_X)))$ $\{O_X \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O_{XY} = \text{MultiHead}(O_{XY}, O_{XY}, O_{XY}, \text{mask} = M)$ $\{O_{XY} \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \mathbf{Q}^{\text{MHA-XY}}))$ $\{O_{XY} \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O_{XY} = \text{LayerNorm}(\text{Add}(O_{XY}, \text{Linear}(\text{Dense}(O_{XY})))$ $\{O_{XY} \in \mathbf{R}^{B \times \ell \times d_O}\}$

$O = \text{Linear}(\text{Concat}(O_X, O_{XY}))$ $\{O \in \mathbf{R}^{B \times \ell \times 2\alpha}\}$

$A, B = O[:, :, : \alpha], O[:, :, \alpha :]$ $\{A, B \in \mathbf{R}^{B \times \ell \times \alpha}\}$

$\mu, \sigma = A + \mathbf{Y}^{n(I)}, B$ $\{\mathbf{Y}^{n(I)} \subset \mathbf{W}, \mu, \sigma \in \mathbf{R}^{B \times \ell \times \alpha}\}$

Return $\mu[:, n_C :, :], \sigma[:, n_C :, :]$