TRANSFORMER NEURAL PROCESSES WITH GROWING RECEPTIVE FIELD FOR UNCERTAINTY PREDICTION OVER CLUSTERED OBSERVATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Processes (NPs) are neural-network-based and Gaussian Process (GP)-inspired models for uncertainty prediction, yet most NP backbones are designed and evaluated under approximately uniform observations. Real-world data are rarely so ideal: measurements collected across the Earth are often clustered, with dense sampling near accessible regions and gaps elsewhere. Recent transformer NPs restrict attention to small local windows for efficiency and report little loss in accuracy; we show, however, the size and structure of the receptive field can affect predictive performance. We propose a growing-window processor that enlarges the attention window layer by layer and appends a global layer to capture both short-range and long-range dependencies. On synthetic Gaussian random fields and real-world climate data (ECA&D), our model achieves better predictive log-likelihood on clustered tasks than the baselines with local windows, indicating that clumped observations benefit from combining local and global receptive fields.

1 INTRODUCTION

Inspired by Gaussian Processes (GPs), Neural Processes (NPs) (Garnelo et al., 2018a;b) were developed to model predictive uncertainty while exploiting the representational capacity of neural networks. Like GPs, NPs predict a posterior distribution over unobserved data conditioned on observed data. A growing body of work adopts (C)NP variants for regression tasks (Scholz et al., 2023; Vaughan et al., 2021; Maraval et al., 2023).

Although the range of data types amenable to Neural Processes is broad, most work on constructing them has focused on tasks in which observation locations are chosen independently, so that contexts are approximately uniformly distributed over the domain; only a few studies consider non-uniformly distributed contexts (Vaughan et al., 2021; Gordon et al., 2020; Ashman et al., 2024b). Among them, Convolutional Conditional Neural Processes (ConvCNPs) evaluated extrapolation on a split-image task as a probe of generalization (Gordon et al., 2020). This family has proven useful in practical applications (Vaughan et al., 2021; Scholz et al., 2023). However, its convolutional backbone is memory-inefficient, and the Fourier Neural Operator recently proposed as a replacement (Mohseni & Duffield, 2026; Li et al., 2020) incurs cost in memory for the model that grows exponentially with the input dimension. Using a transformer as the backbone of NPs has been therefore emerged as a memory-efficient alternative with applications to high dimensional datasets (Ashman et al., 2024a;b); we focus on transformer-based models for this reason.

Building on the success of transformer-based Neural Processes, Gridded Transformer Neural Processes (Gridded TNPs) introduced scalable encoding and decoding techniques for transformer backbones (Ashman et al., 2024b). That work evaluated the models on climate data which contain non-uniform observations, and reported that restricting attention to a small and fixed-size window at every layer performs comparably to full global attention at a grid spacing of more than 1°. That evaluation, however, was carried out only at this coarse spacing; at finer, downscaled resolutions, where spatial correlation persists across neighboring locations, the outcome may differ (Thomassen et al., 2022; Kunz & Laepple, 2024).

To the best of our knowledge, prior studies leave a more specific question unexplored: when does a local receptive field become problematic, and when is a global receptive field useful? An analo-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

gous question arises for GPs: Snelson & Ghahramani (2007) observes that, in GP approximation, attending only to local observations yields poor predictions when those observations are clumped within the domain. We ask whether the same holds for Neural Processes—whether attending only to nearby observations helps or hurts prediction when observations are clumped.

In this work, we study tasks with clustered sampling and the token-processing methods suited to them, and design a transformer backbone configuration tailored to such tasks. All transformer-based neural processes tested here adopt a grid encoder and decoder (Ashman et al., 2024b) and should therefore be called Gridded TNP, Gridded Swin-TNP, and Gridded Growing-Swin-TNP (ours); for brevity, we hereafter refer to them as TNP, Swin-TNP, and Growing-Swin-TNP (G-Swin-TNP).

Clustered observations are common in datasets collected on Earth: stations cannot be installed at every location, so data collection tends to concentrate around accessible regions while some of them lack measurements entirely (Gao, 2021; Dunn et al., 2012).

Our contributions in this work are threefold:

1. We systematically study prediction from clustered observations by transformer-based Neural Processes that process tokens locally, globally, or with a combination of both.
2. We show that enlarging the receptive field of transformer layers is effective for clustered-observation tasks when relevant contexts lie near the targets.
3. We propose Growing-Swin-TNP (G-Swin-TNP), which grows its attention window layer by layer to combine local and global receptive fields, and show that it performs similarly to a global-attention model on synthetic data, while better performing than both purely local and purely global models on real-world clustered climate data.

2 Background

2.1 Conditional Neural Processes and Neural Processes

Conditional Neural Processes (CNPs) and Neural Processes (NPs) (Garnelo et al., 2018a;b) were developed for posterior prediction using neural networks. Among them, we focus on CNP-based models in this work, following the line of ConvCNPs and Gridded TNPs. CNP-based models are those in the Neural Process family that construct the statistics of the predictive posterior distribution – its mean and standard deviation, or a covariance matrix in some cases (Bruinsma et al., 2021). CNPs, like NPs, are trained to perform *tasks*, and we assume a distribution over tasks, $\mathcal{P}$. Each task $T \sim \mathcal{P}$ contains a finite number of observations: *contexts* $\{(\mathbf{x}_c, \mathbf{y}_c)\}_{c=1}^{M}$ and *targets* $\{(\mathbf{x}'_t, \mathbf{y}'_t)\}_{t=1}^{N}$. Given the contexts and the target inputs $\{\mathbf{x}'_t\}_{t=1}^{N}$, (C)NPs aim to find the posterior distribution over the target outputs $\{\mathbf{y}'_t\}_{t=1}^{N}$.

CNPs do so by providing the means and variances $\{(\mu_t, \sigma_t^2)\}_{1 \le t \le N}$ of the posterior distributions, which are assumed to be normal. NPs, by contrast, provide samples from the target posterior distribution. In this work, we focus on the prediction scheme of CNPs. CNPs are trained with a negative log-likelihood (NLL) loss:

$$\mathcal{L}_{\text{NLL}} = -\sum_{t=1}^{N} \log p\left(\mathbf{y}'_t | \mathbf{x}'_t, \mathbf{x}_c, \mathbf{y}_c, c \in [M]\right) \quad [M] = \{c \,|1 \le c \le M\}$$

where p denotes the predicted density of the posterior distribution. Models trained with the same loss as CNPs generally consist of three components, following the convention established by Ashman et al. (2024b): an *encoder*, a *processor*, and a *decoder*. The encoder embeds contexts into a high-dimensional feature space, the processor further encodes this feature space, and the decoder aggregates the feature vectors into the parameters of the target posterior distribution.

2.2 Local-Global Gaussian Process Approximation

We now describe the analogous motivation for clustered contexts. Gaussian Processes are a mathematically tractable method for uncertainty prediction, but they incur a cubic cost in the number of observations – $\mathcal{O}(n^3)$ time for n data points. To retain the functionality of Gaussian Processes while

reducing this cost, approximation methods that consider only a subset of observations have been developed, such as local Gaussian Processes and sparse Gaussian Processes. Snelson & Ghahramani (2007) proposed considering both idea of local and sparse global approximation.

2.3 Attention, Translation-Equivariant Attention, and the Shifting-Window Transformer

The transformer is a widely used neural network architecture, applied to tasks such as language and vision (Vaswani et al., 2017; Dosovitskiy et al., 2020). At its core lies the attention mechanism, which updates each input by taking a weighted sum over other related inputs. The input is called the *query*, the related inputs the *values*, and the inputs used to weight the values the *keys*; in most cases the values and keys coincide. Denote the query as $\boldsymbol{Z} = [\boldsymbol{z}_1; \cdots; \boldsymbol{z}_N]$ and the values as $\boldsymbol{U} = [\boldsymbol{u}_1; \cdots; \boldsymbol{u}_M]$. The attention weights are computed via a scaled dot product between $\boldsymbol{Z}$ and $\boldsymbol{U}$ followed by `softmax` normalization. With parameters W_h^Q, W_h^K, W_h^V, and W_h^O, the query is updated as follows, where H is the number of *heads* (the number of attention operations combined in the update)

$$\boldsymbol{z}_n \leftarrow \sum_{h=1}^{H} \operatorname{softmax}\left(\boldsymbol{z}_n W_h^Q {W_h^K}^\top \boldsymbol{U}^\top\right) \boldsymbol{U} W_h^V W_h^O. \tag{1}$$

When $\boldsymbol{Z}$ and $\boldsymbol{U}$ contain different components, the multi-head attention (MHA) is called multi-head cross-attention (MHCA); when they contain the same components, it is called multi-head self-attention (MHSA).

Although versatile, the attention mechanism has no inherent inductive bias toward *stationary* data. Some datasets, such as images and climate data, are assumed to be *stationary processes*. For instance, consider translating an image by a few pixels within the frame: although the coordinate of each pixel changes, the relative positions of the pixel values are unchanged. This property is called *stationarity*. Rigorous definitions of stationary stochastic processes and translation equivariance are given in Appendix C. Ashman et al. (2024a) developed a translation-equivariant attention mechanism that adds a bias to the attention weights. When the spatial/temporal locations corresponding to $\boldsymbol{Z}$ and $\boldsymbol{U}$ are $\boldsymbol{X} = [\boldsymbol{x}_1; \cdots; \boldsymbol{x}_N]$ and $\boldsymbol{Y} = [\boldsymbol{y}_1; \cdots; \boldsymbol{y}_M]$, the translation-equivariant multi-head attention is computed as

$$\boldsymbol{z}_n \leftarrow \sum_{h=1}^{H} \operatorname{softmax}\left(\left\{\boldsymbol{z}_n W_h^Q {W_h^K}^\top \boldsymbol{U}^\top,\ k([\boldsymbol{x}_n - \boldsymbol{y}_1; \cdots, \boldsymbol{x}_n - \boldsymbol{y}_M])\right\}\right) \boldsymbol{U} W_h^V W_h^O \tag{2}$$

where the kernel k is implemented as an MLP (Ashman et al., 2024a).

The transformer, however, is costly relative to other architectures: for n inputs, a transformer layer costs $\mathcal{O}(n^2)$ time, whereas a convolutional layer costs $\mathcal{O}(n)$. To scale to tasks with large or high-dimensional inputs, the shifting-window (Swin) transformer (Liu et al., 2021) mimics the window-sliding mechanism of convolutional neural networks (CNNs): attention is performed within a window, limiting its complexity to the window size. The hierarchical Swin-Transformer further reduces the number of inputs layer by layer via learnable down-sampling, enabling various vision tasks – such as image classification, segmentation, and object detection – using only a transformer architecture. Gridded TNPs also employ shifting-window attention, but exclude the hierarchical down-sampling, mentioning that it was found not to improve performance Ashman et al. (2024b).

2.4 Gridded Transformer Neural Processes

Attention and transformers have been used in the Neural Process family (Kim et al., 2019a; Nguyen & Grover, 2022) and are known to improve performance (Le et al., 2018), while increasing computational complexity. The use of pseudo-tokens (Feng et al., 2022; Ashman et al., 2024a) has therefore been proposed as a solution. Recently, Gridded Transformer Neural Processes (Gridded TNPs) (Ashman et al., 2024b) arranged gridded tokens by fixing each token to a set location, rather than randomizing the pseudo-tokens. Its encoder and decoder process each grid token/target by gathering the nearest contexts/grid tokens, thereby using multi-head cross-attention efficiently for encoding and decoding. Compared with plain pseudo-token TNPs and grid-based ConvCNPs, Gridded TNPs report improved performance on unstructured tasks (Ashman et al., 2024b).

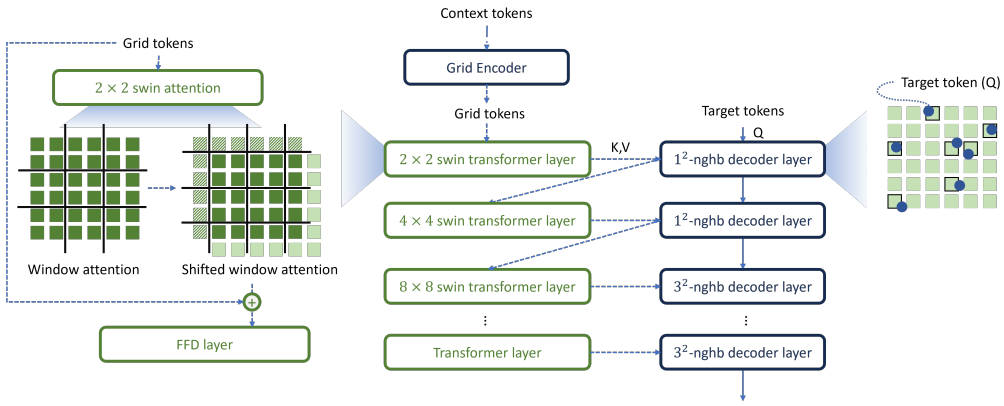


Figure 1: The model shown here is built for a dataset with a 2-dimensional input. **(Left)** Each Swin-Transformer layer follows the construction of the original Swin-Transformer (Liu et al., 2021), detailed in Appendix D.2. **(Center)** The overall structure of the model. **(Right)** Each decoder layer collects neighboring grid tokens for each target, according to the window size of the corresponding processor layer. The detail is in Appendix D.1

3 Methodology

3.1 Construction

The main intuition behind our construction is as follows: when the context is clustered, there are rooms to be filled with, which requires a broader view of the overall realizations. We propose a model with a hierarchical structure which is more cost-efficient than simply stacking multiple global layers. The design suits the structure of transformer-based neural processes adopting grid tokens.

The key design choice of our processor is a stack of *growing-window* local layers followed by a single global layer: rather than down-sampling tokens as in the Swin-Transformer, we enlarge the attention window layer by layer to aggregate multi-scale information, then attend globally once. Since our model adopts Swin-Transformer layers of which window sizes gradually grow, we call it the Growing-Swin-Transformer Neural Process (G-Swin-TNP). We detail each component below. Following the anatomy of the model of Ashman et al. (2024b), we explain the model by each part.

3.1.1 Encoder

We follow the encoder construction of Ashman et al. (2024b), described in Appendix D.1.

3.1.2 Processor and decoder

Prior work on transformer-based Neural Processes (NPs) stacks processor and decoder layers in two ways: alternating processor and decoder layers – the Perceiver style (Jaegle et al., 2021; Ashman et al., 2024b) – or stacking all processor layers before all decoder layers (Nguyen & Grover, 2022; Ashman et al., 2024a). We adopt the Perceiver style, following Ashman et al. (2024b) rather than Nguyen & Grover (2022); Ashman et al. (2024a). In Perceiver-style transformer-based NPs, each decoder layer decodes the targets after its corresponding processor layer, so we pair each decoder layer with its processor layer.

The *processor*, which processes grid tokens with multi-head self-attention, consists of a sequence of Swin-Transformer layers (Liu et al., 2021) with a translation-equivariant bias (TE-bias), followed by a Transformer layer that also uses a TE-bias. The detailed mechanism of the Swin-Transformer layer used in this structure is described in Appendix D.2, following the original paper (Liu et al., 2021). Briefly, a Swin-Transformer layer performs multi-head self-attention among tokens grouped into a local region called a *window*. Figure 1 illustrates an instance of the overall structure of G-Swin-TNP for a 2-dimensional input. In general, one stacks Swin-Transformer layers with windows

of size ω_i^ℓ along each axis at the ℓ-th layer, each paired with a decoder layer that uses $(\min(\upsilon^\ell, \mathfrak{n}))^d$ nearest neighbors where $\mathfrak{n}$ is the maximum number of nearest neighbors per axis.

The original Swin-Transformer (Liu et al., 2021), used for vision tasks, adopts a fixed window size across all layers and reduces the number of pixels at each layer. In G-Swin-TNP, however, the grid decoder gathers the nearest grid tokens to each target based on the grid tokens' positions, and thus requires positional information: each grid token carries a corresponding position, and downsampling the grid tokens would require a different set of positions. We therefore keep the number of grid tokens fixed across layers. Starting from a small-window Swin-Transformer layer, we increase the window size of successive layers, in contrast to the fixed window size of the original Swin-Transformer. This *growing-window* design lets deeper layers integrate information over progressively larger regions, and a final Transformer layer with a global view is appended to capture long-range dependencies. We refer to the Swin-Transformer layers, whose attention covers tokens within a local window, as *local layers*, and to the final Transformer layer as the *global layer*.

The *grid decoder* in G-Swin-TNP is implemented following the recent work of Ashman et al. (2024b), and is explained in Appendix D.1 to keep this paper self-contained. Briefly, the grid decoder collects several nearest grid tokens for each target and applies multi-head cross-attention, with the target tokens as queries and the grid tokens as keys and values. The collection of grid tokens for each target is depicted on the right-hand side of Figure 1.

To keep the decoder consistent with the processor layer's receptive field, the number of nearest-neighbor grid tokens for each target is adjusted according to the window size of the preceding processor layer; this number depends on the window size. Suppose the window of the ℓ-th processor layer has length ω_i^ℓ along axis i, for $1 \leq i \leq d$, where d is the input dimensionality. Following the implementation of the grid decoder layer (Ashman et al., 2024b), the decoder finds the nearest grid token to each target and takes as keys and values the tokens surrounding it within a cube, collecting r^d nearest neighbors – with masking where necessary – for an odd number r, as shown in Figure 8b. For consistency, this cube must fit within the window. We therefore set the number of nearest neighbors collected by the following grid decoder layer to $\big(\min\big(\upsilon^\ell,\ \mathfrak{n}\big)\big)^d$, where υ^ℓ is the largest odd number not exceeding $\min_i \lfloor \omega_i^\ell / 2 \rfloor$, and $\mathfrak{n}$ is the maximum allowed number of neighbors per axis.

3.2 Computational complexity

The computational complexity of each Transformer or Swin-Transformer layer—the building blocks of Growing-Swin-TNP—depends on the window size ω. For a window of size ω, the TE-bias we adopt adds a cost of order ω^2; since this is absorbed into the constant factor of the big-$\mathcal{O}$ notation, we omit it in the analysis below. By design, successive local layers in Growing-Swin-TNP use window sizes that grow by a factor of 2^d, where d is the input dimensionality. Let T denote the number of grid tokens, L_{L} the number of local layers, and L_{G} the number of global layers. The total complexity of the processor in Growing-Swin-TNP is

$$\sum_{k=0}^{L_{\mathrm{L}}-1} 2\left\{2^{-d(L_{\mathrm{L}}-1-k)}\, T\right\}^2 + \sum_{k=0}^{L_{\mathrm{G}}-1} T^2 = \frac{2\left(1-4^{-dL_{\mathrm{L}}}\right)}{1-4^{-d}} + L_{\mathrm{G}} T^2. \tag{3}$$

Since $\dfrac{2\left(1-4^{-dL_{\mathrm{L}}}\right)}{1-4^{-d}} < \dfrac{2}{1-4^{-d}}$, the combined cost of all local layers is bounded by a constant independent of L_{L}. Consequently, the total complexity of Growing-Swin-TNP does not exceed that of a $\big(\lceil 2/(1-4^{-d})\rceil + L_{\mathrm{G}}\big)$-layer Transformer. The factor 2 for the local layers come from the implementation of Swin-Transformer (Liu et al., 2021), as described in Appendix D.2.

Table 1 summarizes the computational complexity of each model and the empirical wall-clock time for training over 100 epochs, for the experiment in Section 4.1. All models are trained on four NVIDIA GeForce RTX 3090 GPUs. We report GPU-hours, computed as the number of GPUs multiplied by the wall-clock training time.

Table 1 reports the training time over 100 epochs for two settings: the 2D grid-token case (**Time 1**) and the 3D grid-token case (**Time 2**). In the 2D case, SConvCNP requires the longest time, followed by ConvCNP and G-Swin-TNP. The higher training cost of G-Swin-TNP, relative to TNP, stems from the effective doubling of layers inherent to Swin-Transformer blocks. Although Con-

Table 1: Computational complexity of the processor of each model.T is the number of grid points. L is the number of processor layers. ω is the window size of Swin-TNP; for Growing-Swin-TNP, L_{L} and L_{G} denote the number of local and global layers of the processor. For brevity, we write TNP, Swin-TNP, and G-Swin-TNP for the gridded, TE-biased TNP, Swin-TNP, and Growing-Swin-TNP, respectively. Time 1 and Time 2 report GPU-hours for the 2D and 3D grid-token experiments (experiments in Section 4.1.1 and Section 4.2, respectively). ConvCNP and SConvCNP are omitted in **Time 2** because they were not trained in 3-dimensional case 4.2.

Model	ConvCNP	SConvCNP	TNP	Swin-TNP	G-Swin-TNP (Ours)
Complexity	$\mathcal{O}(LT)$	$\mathcal{O}(LT \log T)$	$\mathcal{O}(LT^2)$	$\mathcal{O}(2L\omega^2)$	Eq. (3)
Time 1 (hr)	24.61	59.11	20.92	22.34	24.23
Time 2 (hr)	-	-	20.01	4.28	15.90

vCNP and SConvCNP scale better with the number of grid tokens, they take longer to train than the transformer-based models, likely due to their larger parameter counts.

The relative cost of TNP and G-Swin-TNP depends on the setting: TNP is faster in the 2D case, but slower in the 3D case. We attribute this to the larger number of tokens: as T grows, the cost of T^2 in every layer becomes to dominate the training cost and outweighs the layer-doubling overhead of the Swin-Transformer blocks.

4 Experiment

We compare G-Swin-TNP with the Transformer Neural Process (TNP) and the Swin-Transformer Neural Process (Swin-TNP) whose encoders and decoders are grid encoders and decoders. We also test ConvCNP and SConvCNP which use grid tokens for function representations. We add a translation-equivariant bias to the transformer-based models so that, like CNN and FNO, they incorporate such a bias, ensuring a fair comparison. For our experiments, we referred to the implementations of Ashman et al. (2024b) and Liu et al. (2021), which the authors kindly made available.

4.1 2D Gaussian random field with Matérn covariance

4.1.1 Experiment 1: Clustered sampling for task generation

For the synthetic dataset, we adopt the `GSTools` package (Müller & Heße, 2022) to generate realizations of a Gaussian random field (GRF) with a $5/2$ Matérn covariance, each of size 32×32. The implementation of `GSTools` is detailed in Appendix A.2. We consider two GRFs with length scales $\ell = 0.1$ and $\ell = 0.2$. Intuitively, the length scale controls the degree of spatial correlation: a larger ℓ induces stronger spatial correlation, so each pixel value depends more heavily on its neighbors.

We generate 20,000 realizations for training and 5,000 for testing, with each realization yielding one task. As in other Neural Process (NP) families, a task is formed by partitioning the points into contexts and targets. We follow this standard protocol, except that our *context sampling* differs from prior work (Ashman et al., 2024b; Vaughan et al., 2021; Ashman et al., 2024a): rather than sampling contexts uniformly, we draw them in spatial clusters.

Given $N = 32^2$ points in a realization, we set the number of contexts N_c by sampling $N_c/N \sim \mathcal{U}[0.025, 0.3]$. We then draw a cluster count $P \sim \mathrm{Unif}\{3, \dots, 10\}$, cluster centers $\boldsymbol{c}_1, \dots, \boldsymbol{c}_P$ uniformly over the domain, and standard deviations $\sigma_1, \dots, \sigma_P \sim \mathcal{U}[0.03, 0.1]$. The N_c contexts are sampled from these P blobs, the p-th centered at $\boldsymbol{c}_p$ with standard deviation σ_p, using the Gumbel-max trick (Kool et al., 2019) (see Appendix A.2.2). The remaining points form the targets.

Table 2 reports the performance of all models with synthetic GRF datasets of length scale 0.1 and 0.2. Among the models, the best performance is achieved by TNP, a transformer-based model with a global receptive field, and G-Swin-TNP follows closely. Even though TNP performs a little better than G-Swin-TNP, recall that it requires more time than ours when more tokens are necessary (e.g., the 3D case) as shown in Table 1. Comparing ConvCNP and SConvCNP, the latter, which uses a

Table 2: Performance of each model on datasets of different length scales. Negative log-likelihood is denoted by **NLL** (↓) and root mean squared error by **RMSE** (↓). For the transformer-based models (TNP, Swin-TNP, and G-Swin-TNP (ours)), we use grid encoders and decoders and add a TE-bias. We repeated all the experiments 5 times.

Models	**Length scale**			
	0.1		0.2	
	NLL (↓)	**RMSE (↓)**	**NLL (↓)**	**RMSE (↓)**
ConvCNP	0.776±0.016	0.0.696±0.005	0.226±0.038	0.508±0.006
SConvCNP	0.854±0.003	0.715±0.002	0.405±0.003	0.536±0.002
TNP	0.434±0.012	0.627±0.002	-0.741±0.031	0.353±0.004
Swin-TNP	0.545±0.075	0.642±0.01	-0.455±0.193	0.386±0.014
G-Swin-TNP (Ours)	0.452±0.021	0.63±0.003	-0.658±0.083	0.386±0.014

globally-receptive Fourier neural operator (FNO), underperforms ConvCNP. We attribute this to the 32×32 grid tokens shared by all models: this resolution admits only a small number of frequency modes for the FNO (Duruisseaux et al., 2025; Li et al., 2020), which likely limits its performance.

4.1.2 Experiment 2: Negative log-likelihood depending on characteristics of clusters

Under clustered observations, the reliability of the prediction at a target can vary with its distance from the observed clusters. The distance between contexts and targets can be affected by two factors: number of clusters and standard deviation of clusters. We generate tasks with test dataset with the following two settings. We reuse the task-generation procedure of Section 4.1.1, varying only the following controlled factors.

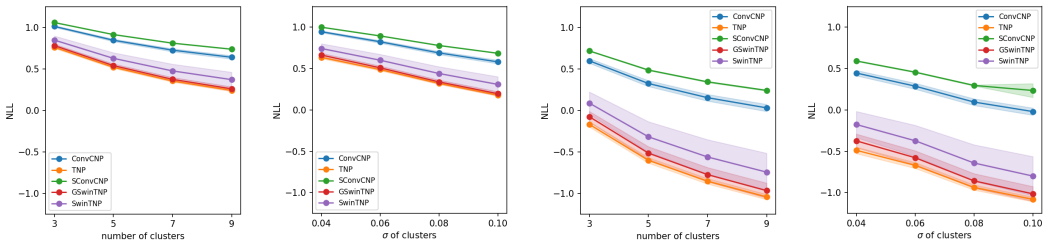
1. Tasks with varying number of clusters: Each test set uses contexts drawn from 3, 5, 7, or 9 clusters, with randomly selected standard deviations and number of contexts, as suggested in Section 4.1.1.
2. Tasks with varying standard deviations of clusters: Each test set uses contexts sampled from clusters with $\sigma_p = 0.04$, 0.06, 0.08, or 0.1 for all p. The number of contexts and the number of clusters are randomly selected, based on the context sampling method described in Section 4.1.1.

To quantify distance between targets and contexts, we consider two notions of distance.

1. Mahalanobis distance d_{M}: We represent the clusters by their centers $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_P$, and define the distance of a target $\boldsymbol{x}'$ from the clusters as the distance to the nearest center, i.e. $d_{\mathrm{M}}(\boldsymbol{x}') = \min_{1 \le p \le P} \|\boldsymbol{x}' - \boldsymbol{c}_p\| / \sigma_p$.
2. Euclidean distance d_{E}: Denoting the context coordinates by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N_c}$, we define the distance of a target $\boldsymbol{x}'$ from the clusters as the distance to the nearest context, i.e. $d_{\mathrm{E}}(\boldsymbol{x}') = \min_{1 \le c \le N_c} \|\boldsymbol{x}' - \boldsymbol{x}_c\|$.

In other words, Mahalanobis distance checks if the targets are in the dense area of the cluster since our blobs are approximately Gaussian distributions, while Euclidean distance tells the distance between a target and the nearest context.

Figure 3 in the appendix overlays, for each test set, the distribution of the corresponding distance (d_{M} or d_{E}) over all of its targets. Increasing the number of clusters or increasing the standard deviation of each cluster yields distributions of target distances with smaller Mahalanobis or Euclidean distances. This matches our intent: varying the number of clusters or the standard deviation of the blobs induces a range of context densities, letting us verify that the aggregate performance is not dominated by a specific density regime.



(a) (b) (c) (d)

Figure 2: Figure 2a and Figure 2c are plots of negative log-likelihoods (NLLs) with varying number of clusters. Figure 2b and Figure 2d are plots of NLLs with varying standard deviation of cluster blobs. The tasks for Figure 2a and Figure 2b are sampled from realizations of a GRF with length scale 0.1, and those for Figure 2c and Figure 2d are sampled from realizations with length scale 0.2.

The experimental results are summarized in the Figure 2. Overall, the models based on the grid encoder, decoder (Ashman et al., 2024b), and transformer-based processors, achieve lower negative log-likelihood (NLL), and the models that use a broad receptive field in the processor — TNP and G-Swin-TNP (ours) — outperform the model that uses a small window at every layer. As the number of clusters or the standard deviation of the blobs increases, all models attain lower NLL. As Figure 3 shows, a larger number of clusters or a larger standard deviation means more targets are likely to lie close to a context, so this trend is expected. The results on the length-scale 0.2 GRF test dataset are generally better than those on the length-scale 0.1 sets, which is again natural, since a larger length scale implies stronger spatial correlation.

Figures 2a and 2b, and Figures 2c and 2d, show similar NLL values and slopes within each length scale. Comparing length scales 0.1 and 0.2, however, the improvement from increasing the number of clusters or the standard deviation is larger at 0.2, and so is the performance gap between models with different degrees of broad receptive field, overall. At length scale 0.1, TNP and G-Swin-TNP perform almost identically, whereas at length scale 0.2 TNP is somewhat stronger. Together, these results suggest that under stronger spatial correlation, dense contexts are more helpful for prediction, and a broader processor receptive field—which can aggregate information from more grid tokens—is likewise more beneficial.

4.2 E-OBS DAILY MAXIMUM TEMPERATURE

In this experiment, we train and evaluate models on the daily maximum-temperature E-OBS ensemble dataset (Cornes et al., 2018; Klein Tank & Coauthors, 2002).[1] [2]

We consider the daily maximum temperature over latitude $[35, 50]$ and longitude $[10, 25]$, a region in Southern Europe where stations are highly clustered. The models are trained on 1980–2015, validated on 2016–2020, and tested on 2021–2025. We normalize the dataset with the mean and standard deviation of the training dataset.

The E-OBS ensemble dataset provides daily maximum temperature on land at a $0.1°$ grid spacing. Using the metadata provided with the E-OBS dataset (Cornes et al., 2018), we obtain the station locations within the region and extract the $2{,}773$ grid points nearest to these stations (Figure 4 in the appendix). From the remaining grid points, we subsample 413 points on a uniform $0.5°$ grid.

For each task, we subsample eight days of maximum temperature at a two-day interval. We then sample a fraction $u \sim \mathcal{U}(0, 0.1)$ of the station points as contexts, and sample the targets from the remaining points, taking 10% of the total number of points (including both station and non-station

[1] The E-OBS/ECA&D data policy applies. These data are strictly for non-commercial research and education; see `https://www.ecad.eu` and the data policy at `https://knmi-ecad-assets-prd.s3.amazonaws.com/documents/ECAD_datapolicy.pdf`.

[2] We acknowledge the E-OBS dataset from the EU-FP6 project UERRA (`https://www.uerra.eu`) and the Copernicus Climate Change Service, and the data providers in the ECA&D project (`https://www.ecad.eu`).

Table 3: Performance results of each model for E-OBS ensemble dataset. Negative log-likelihood is denoted **NLL** (↓), root mean squared error is denoted in **RMSE** (↓). For the transformer-based models (TNP, Swin-TNP, and ours Growing-Swin-TNP), we use grid encoders and decoders and add a TE-bias. The performance is measured for normalized dataset.

	Model		
	TNP	Swin-TNP	G-Swin-TNP (Ours)
NLL (↓)	-0.351±0.012	-0.361±0.007	-0.395±0.006
RMSE (↓)	0.206±0.001	0.206±0.001	0.201±0.001

points). Although the context-sampling method does not explicitly enforce clustering, the clumped distribution of the underlying stations naturally yields clustered contexts.

We do not run ConvCNP due to its memory inefficiency in the three-dimensional setting, and we exclude SConvCNP because it does not define a grid length along the temporal axis.

Table 3 presents the performance of each model. On this dataset TNP which uses a global attention window, i.e., a plain transformer processor, and Swin-TNP which uses local window attention perform almost identically, while G-Swin-TNP outperforms both. Note that one main difference between this real-world dataset and the synthetic datasets before is the geography of the region. Because the selected area in the E-OBS ensemble dataset includes the sea, two nearby points separated by water may exhibit low spatial correlation; similarly, differences in terrain, such as intervening mountain ranges or changes in elevation, may also reduce spatial correlation. Indeed, the last panel of Figure 4 shows that the temperature distributions of two points on opposite sides of the Alps differ. In such cases, considering a limited receptive field, as proposed by Ashman et al. (2024a), may be beneficial. However, our earlier experiments suggest that a broad receptive field also helps when the contexts relevant to a target are clustered. Since G-Swin-TNP employs a processor that combines both limited and broad receptive fields, these two factors may jointly contribute to its better performance as shown here.

5 Conclusions

In this work, we systematically study neural processes (NPs) on tasks with clustered observations. Because targets must be predicted in the gaps between clusters, we hypothesize that the processor's receptive-field size would affect performance, and we focus on grid-based, transformer-based NPs for their memory efficiency.

Motivated by this hypothesis, we develop the Growing-Swin-TNP, which is built from Swin-Transformer blocks (Liu et al., 2021) and Gridded TNPs (Ashman et al., 2024b) and grows its window size layer by layer. Our experiments confirm that the per-layer receptive-field size affects the performance of transformer-based NPs, revealing a trade-off between computational cost and predictive performance under clustered observations. On the 2D synthetic GRF dataset, G-Swin-TNP attains the second-best performance after Gridded TNP. On a real-world daily-maximum-temperature dataset with highly clustered observations, it performs better than the other transformer-based models in NLL, which appears to reflect both the clustered observations and the geographic characteristics of the data. In training cost, G-Swin-TNP is more expensive than Gridded TNP in the 2D case but cheaper in the 3D case, indicating better efficiency when the number of grid tokens is large.

Because clustered observations are a common sampling pattern in spatial data (Gao, 2021), we believe our study broadens the applicability of neural processes toward such settings.

Acknowledgment

We acknowledge the E-OBS dataset from the EU-FP6 project UERRA (`https://www.uerra.eu`) and the Copernicus Climate Change Service, and the data providers in the ECA&D project (`https://www.ecad.eu`) (Cornes et al., 2018).

AI USE SECTION

In this work, we used generative AI tools to create or edit, and edit a research paper to improve readability. We have not used generative AI tools for other tasks, such as generation of synthetic data sets, developing theoretical models or conceptual frameworks, formulation of mathematical claims, provision of critical ingredients for proving mathematical claims, assistance in the writing of proofs, proposal or refinement of hypotheses, implementation of methods, cleaning and reformatting dataset, support qualitative and thematic data analysis, and interpretation of results. We have reviewed all AI-assisted work. We manually checked related literature to prevent potential plagiarism. For the possibility of malfunctioning of the code, we checked the code that AI revised or generated. We proof-read the sentences that generative AI added or corrected. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To support reproducibility, we report the GPUs used in our experiments in Section 3.2, and we provide the training and model hyperparameters in Appendix A.4. The construction of our model is described in detail in Section 3.1.2. For the synthetic dataset, we describe the packages and kernel choices in Appendix A.2 and the task-sampling method in Appendix A.2.2. The real-world dataset is publicly available for non-commercial use, and we specify the exact subset used in Section 4.2. Our code will be made available upon acceptance.

REFERENCES

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

Matthew Ashman, Cristiana Diaconu, Eric Langezaal, Adrian Weller, and Richard E. Turner. Gridded transformer neural processes for large unstructured spatio-temporal data, 2024b. URL https://arxiv.org/abs/2410.06731.

Pierre Brémaud. Fourier analysis of stochastic processes. In *Fourier analysis and stochastic processes*, pp. 119–179. Springer, 2014.

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

Richard C. Cornes, Gerard van der Schrier, Else J. M. van den Besselaar, and Philip D. Jones. An ensemble version of the E-OBS temperature and precipitation datasets. *Journal of Geophysical Research: Atmospheres*, 123(17):9391–9409, 2018. doi: 10.1029/2017JD028200.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *International conference on learning representation*, 2020.

Robert JH Dunn, Kate M Willett, Peter W Thorne, Emma V Woolley, Imke Durre, Aiguo Dai, David E Parker, and RS Vose. Hadisd: A quality-controlled global synoptic report database for selected variables at long-term stations from 1973–2011. *Climate of the Past*, 8(5):1649–1679, 2012.

Valentin Duruisseaux, Jean Kossaifi, and Anima Anandkumar. Fourier neural operators explained: A practical perspective. *arXiv preprint arXiv:2512.01421*, 2025.

Leo Feng, Hossein Hajimirsadeghi, Yoshua Bengio, and Mohamed Osama Ahmed. Latent bottlenecked attentive neural processes. *arXiv preprint arXiv:2211.08458*, 2022.

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.

Jay Gao. *Fundamentals of spatial analysis and modelling*. CRC Press., 2021.

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 Jimenez Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. *ArXiv*, abs/1807.01622, 2018b.

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional conditional neural processes. *International conference on learning representation*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Falk Heße, Vladyslav Prykhodko, Steffen Schlüter, and Sabine Attinger. Generating random fields with a truncated power-law variogram: A comparison of several numerical methods. *Environmental modelling & software*, 55:32–48, 2014.

Michael J Hutchinson, Charline Le Lan, Sheheryar Zaidi, Emilien Dupont, Yee Whye Teh, and Hyunjik Kim. Lietransformer: Equivariant self-attention for lie groups. In *International conference on machine learning*, pp. 4533–4543. PMLR, 2021.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *International conference on learning representation*, 2019a.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, S. M. Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *International conference on learning representation*, 2019b.

A.M.G. Klein Tank and Coauthors. Daily dataset of 20th-century surface air temperature and precipitation series for the european climate assessment, 2002. URL `https://www.ecad.eu`.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International conference on machine learning*, pp. 3499–3508. PMLR, 2019.

Torben Kunz and Thomas Laepple. Effective spatial degrees of freedom of natural temperature variability as a function of frequency. *Journal of Climate*, 37(8):2505–2518, 2024.

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

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

Haitao Liu, Yew-Soon Ong, Xiaobo Shen, and Jianfei Cai. When gaussian process meets big data: A review of scalable gps. *IEEE transactions on neural networks and learning systems*, 31(11): 4405–4423, 2020.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Ze Liu, Han Hu, Yutong Lin, Zhuliang Yao, Zhenda Xie, Yixuan Wei, Jia Ning, Yue Cao, Zheng Zhang, Li Dong, et al. Swin transformer v2: Scaling up capacity and resolution. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11999–12009. IEEE, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International conference on learning representation*, 2017.

Alexandre Maraval, Matthieu Zimmer, Antoine Grosnit, and Haitham Bou Ammar. End-to-end meta-bayesian optimisation with transformer neural processes. *Advances in Neural Information Processing Systems*, 36:11246–11260, 2023.

Met Office. *Cartopy: a cartographic Python library with a Matplotlib interface*. Exeter, Devon, 2010 - 2015. URL https://cartopy.readthedocs.io.

Peiman Mohseni and Nick Duffield. Spectral convolutional conditional neural processes. *Advances in Neural Information Processing Systems*, 38:40416–40446, 2026.

Schüler L. Zech A. Müller, S. and F. Heße. Gstools v1.3: a toolbox for geostatistical modelling in python. *Geosci. Model Dev.*, pp. 3161–3182, 2022. URL https://doi.org/10.5194/gmd-15-3161-2022.

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *International conference on machine learning*, 2022.

C. E. Rasmussen and C. K. I. Williams. Gaussian processes for machine learning, 1st edn. The MIT Press, 2005. ISBN 9780262256834. URL https://doi.org/10.7551/mitpress/3206.001.0001.

Jonas Scholz, Tom R Andersson, Anna Vaughan, James Requeima, and Richard E Turner. Sim2real for environmental neural processes. *arXiv preprint arXiv:2310.19932*, 2023.

Edward Snelson and Zoubin Ghahramani. Local and global sparse gaussian process approximations. In *Artificial Intelligence and Statistics*, pp. 524–531. PMLR, 2007.

Emma Dybro Thomassen, Søren Liedtke Thorndahl, Christoffer Bang Andersen, Ida Bülow Gregersen, Karsten Arnbjerg-Nielsen, and Hjalte Jomo Danielsen Sørup. Comparing spatial metrics of extreme precipitation between data from rain gauges, weather radar and high-resolution climate model re-analyses. *Journal of Hydrology*, 610:127915, 2022.

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.

Anna Vaughan, Will Tebbutt, J Scott Hosking, and Richard E Turner. Convolutional conditional neural processes for local climate downscaling. *Geoscientific Model Development Discussions*, 2021:1–25, 2021.

Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014. URL http://timvieira.github.io/blog/gumbel-max-trick-and-weighted-reservoir-sampling/.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *2021 IEEE/CVF international conference on computer vision (ICCV)*, pp. 548–558. IEEE, 2021.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

A EXPERIMENT DETAILS

A.1 TRAINING DETAILS

All the models are trained with AdamW optimizer (Loshchilov & Hutter, 2017) with 0.0005 learning rate and 0.5 gradient norm clipping. All the training batches are of size 16, and the test batches are of size 4. For experiment in Section 4.1, each training epoch, each model iterates the whole dataset. For experiment on climate dataset, to generalize over the large period, 200 batches are randomly selected for each training epoch.

A.2 2D GAUSSIAN RANDOM FIELD WITH MATÉRN COVARIANCE

A.2.1 GENERATING DATASET

Using the `GSTools` package (Müller & Heße, 2022), we generate Gaussian random field (GRF) realizations on a two-dimensional input domain with a Matérn covariance. The Matérn covariance is given by the correlation function (Müller & Heße, 2022; Rasmussen & Williams, 2005):

$$\frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{r}{\ell}\sqrt{\nu}\right)^{\nu} K_{\nu}\left(\frac{r}{\ell}\sqrt{\nu}\right),$$

with r being the distance between inputs, $\nu = 5/2$, $\ell = 0.1$ and $\ell = 0.2$ (Experiment 4.1).

The following describes the GRF approximation adopted by `GSTools`; we follow the formulation of Heße et al. (2014). The Gaussian random field $u(\boldsymbol{x})$ is represented by a stochastic Fourier integral:

$$u(\boldsymbol{x}) = \int_0^{\infty} \cos(2\pi \boldsymbol{k}\cdot\boldsymbol{x})\sqrt{E(\boldsymbol{k})}\mathrm{d}W^1(\boldsymbol{k}) + \int_0^{\infty} \sin(2\pi \boldsymbol{k}\cdot\boldsymbol{x})\sqrt{E(\boldsymbol{k})}\mathrm{d}W^2(\boldsymbol{k}),$$

where $E(\boldsymbol{k}) = S(\boldsymbol{k})/\sigma^2$, and $W^1(\boldsymbol{k})$ and $W^2(\boldsymbol{k})$ are independent Wiener processes. Here $S(\boldsymbol{k})$ is the Fourier transform of the covariance function of $u(\boldsymbol{x})$. This integral can in turn be approximated as

$$u(\boldsymbol{x}) \approx \sqrt{\frac{2\sigma^2}{N}} \sum_{i=1}^{N} \left(Z_i^1 \cos(2\pi \boldsymbol{k}_i\boldsymbol{x}) + Z_i^2 \sin(2\pi \boldsymbol{k}_i\boldsymbol{x})\right)$$

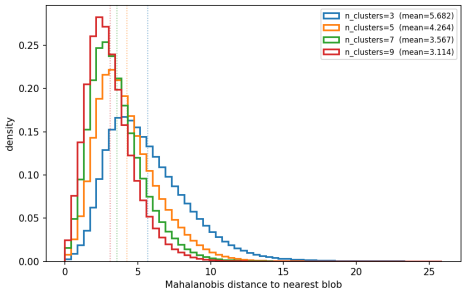
where $Z_i^1, Z_i^2 \sim \mathcal{N}(0,1)$ are independent standard normal random variables, and each $\boldsymbol{k}_i$ is a random variable (one per mode i) whose probability density function is the spectral density $S(\boldsymbol{k})/\sigma^2$.

A.2.2 CONTEXT SAMPLING METHOD

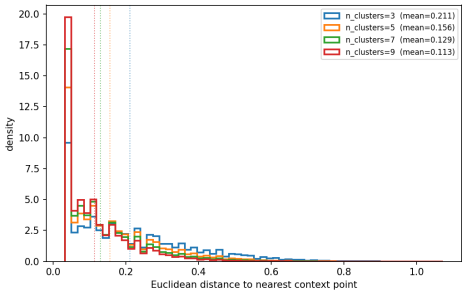
To sample contexts, we adopt the algorithm described in Kool et al. (2019), as follows:

1. Draw cluster count $P \sim \mathrm{Unif}\{3,\dots,10\}$.
2. Sample the cluster centers $\boldsymbol{c}_1,\dots,\boldsymbol{c}_P$ uniformly over the domain, and the standard deviations $\sigma_1,\dots,\sigma_P \sim \mathcal{U}(0.03, 0.1)$.
3. At each coordinate $\boldsymbol{x}_i$ in the domain, compute $\phi_i = \log \sum_p \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{c}_p\|^2}{2{\sigma_p}^2}\right)$.
4. For each coordinate $\boldsymbol{x}_i$, independently draw $u_i \sim \mathcal{U}(0,1)$.
5. Compute the score $\phi_i - \log(-\log(u_i))$, and collect the N_C highest-scoring coordinates in the domain.

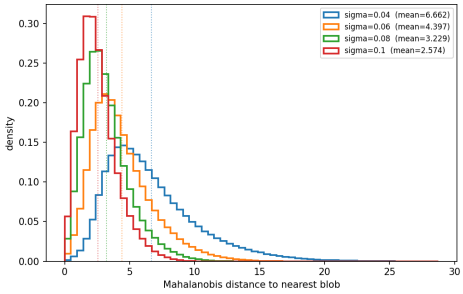
Kool et al. (2019); Vieira (2014) show that this procedure samples N_C coordinates without replacement from the categorical distribution Categorical $\left(\frac{\exp(\phi)}{\sum_i \exp(\phi_i)}\right)$ without replacement. We thereby sample contexts approximately from the probability density $f(\boldsymbol{x}) = \frac{1}{C}\sum_{p=1}^{P} \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{c}_p\|^2}{2{\sigma_p}^2}\right)$, a mixture of P Gaussian blobs, where C is the normalizing constant.



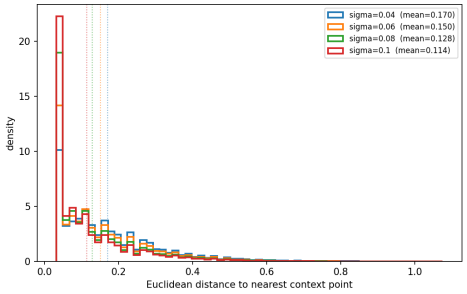
(a) Mahalanobis distance distributions of datasets with varying number of clusters



(b) Euclidean distance distributions of datasets with varying number of clusters



(c) Mahalanobis distance distributions of datasets with varying number of clusters



(d) Euclidean distance distributions of datasets with varying number of clusters

Figure 3: Each density line shows the distribution of targets with different Mahalanobis or Euclidean distances from the contexts.

A.2.3 Testing tasks for Mahalanobis and Euclidean distances

The Figure 3 shows the distribution of distances from targets to clusters. Figure 3a and Figure 3b shows the distribution of targets for testing with varying the number of clusters, and Figure 3c and Figure 3d shows that with varying standard deviation of clusters. Large number of clusters or standard deviation means statistically smaller Mahalanobis distance from the contexts to targets, therefore the Figure 3a and Figure 3c show decreasing average distance with the number of clusters. Figure 3b and Figure 3d also shows decreasing average distances with the size of standard deviation.

A.3 E-OBS maximum temperature experiment

We subsampled E-OBS daily maximum temperature dataset dataset (Cornes et al., 2018; Klein Tank & Coauthors, 2002) as described in Section 4.2. We normalize the each training, validation, and test dataset with mean and standard deviation of the training dataset.

The coordinates of each task is normalized to $[0, 1]$ along the temporal, latitudinal, and longitudinal axes. This is consistent with the usage of translation-equivariant bias of the transformer-based neural processes, where the relative position among the contexts and targets matters.

A.4 Model hyperparameters

All models compared in our experiments are grid-based; that is, their processors operate on grid tokens. For the 2D Gaussian random field synthetic dataset, the grid tokens of every model have size 32×32. For the 3D climate dataset, the grid tokens have size $4 \times 32 \times 32$, corresponding to the temporal, latitudinal, and longitudinal axes, with a two-day spacing along the temporal axis. For the transformer models that adopt an attention window, the temporal extent of the window is fixed to 4 and the window is not shifted along the temporal axis, since the clustering structure lies in the latitude–longitude plane rather than along time.

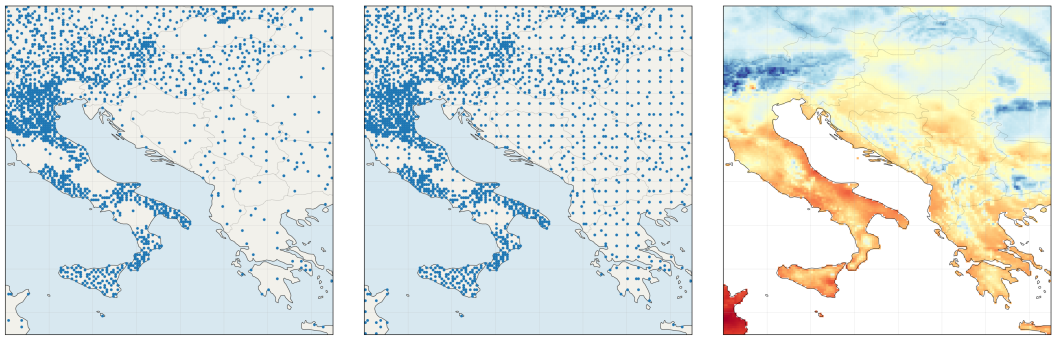


Figure 4: Distribution of stations in the range of latitude [35, 50] and longitude [10, 25]. We adopted `Cartopy` library (Met Office, 2010 - 2015) to present the map on the background. The figure on the left hand side is the locations of the stations. The number of stations, from which the contexts are sampled, is 2,773. The figure in the middle are the extracted locations for targets, including both stations and extra grid points. The number of the grid points is 3,186. The figure on the right hand side is the plot of the whole maximum temperature on one day.

ConvCNP and SConvCNP. For ConvCNP, the kernel size is 7 in every layer. For SConvCNP, in the experiment of Section 4.1, the backbone Fourier Neural Operator (FNO) uses 16 frequency modes, the maximum admissible for this grid size. Both models use the SetConv encoder and decoder of Gordon et al. (2020) with a learnable length scale, and follow a ResNet-style architecture (He et al., 2016) as described by Mohseni & Duffield (2026).

Transformer-based models. For the transformer models with a gridded encoder and decoder, TNP and Swin-TNP use 3^d nearest neighbors in the grid decoder, where d is the input dimensionality. Growing-Swin-TNP determines the number of nearest neighbors as described in Section 3.1.2, capped at 3^d.

The Processor of Swin-TNP adopts window sizes 4×4 window, with the shifting size the half – 2×2 for 2-dimensional tasks. For 3-dimensional tasks, the model adopts $4 \times 4 \times 4$ windows, shifting only on longitudinal–latitudinal axes.

The processor of G-Swin-TNP is specified by the per-layer window sizes. In the experiment of Section 4.1, the local layers use window sizes 2×2, 4×4, 8×8, 16×16, and 32×32, in this order, followed by an appended global (full-attention) transformer layer. In the experiment of Section 4.2, the local layers use window sizes $4 \times 2 \times 2$, $4 \times 4 \times 4$, $4 \times 8 \times 8$, $4 \times 16 \times 16$, and $4 \times 32 \times 32$, in this order, again followed by a global transformer layer. Thus the total number of layers for G-Swin-TNP is, counting the Swin-Transformer layer as a single layer, 6 like the other transformer-based models. In both experiments, the shifting of the local layers are the half of the window size, e.g. 1×1 for 2×2 window, except that the leading 4 is the temporal extent of climate dataset, as noted above, and windows are not shifted along the temporal axis.

Common settings. For all models, the embedding dimension is 64 and the number of layers is 6, except for ConvCNP and SConvCNP, which use 12 and 10 layers respectively, owing to their use of skip connections. To keep the predictive log-likelihood bounded, we impose a constant $\sigma_{\min} = 0.1$ on the predicted standard deviation as below in Equation (4), following (Gordon et al., 2020; Le et al., 2018). The raw output σ_{Model} of the network is mapped to

$$0.1\,\sigma_{\min} + 0.9\,\zeta(\sigma_{\text{Model}}), \tag{4}$$

where ζ denotes softplus function.

B EXPERIMENT RESULT

B.1 2D GAUSSIAN RANDOM FIELD WITH MATÉRN COVARIANCE

Figure 5 and Figure 6 show the root mean square error (RMSE) of each model's prediction and the predicted standard deviation for a realization of Gaussian random field with Matérn covariance of length scale 0.1 and 0.2, respectively.

B.2 E-OBS MAXIMUM TEMPERATURE EXPERIMENT

Figure 7 shows the observed contexts, the mean prediction, and the predicted standard deviation of each model on a single day.

C TRANSLATION EQUIVARIANT NEURAL NETWORKS AND TRANSLATION EQUIVARIANT BIAS

The rigorous definition of stationary stochastic process, from which a stationary dataset is sampled, is borrowed from (Brémaud, 2014).

Definition 1 (Stationary Process) *A stochastic process* $\{X(t)\}_{t\in T}$ *is* stationary *if, for all* $\{t_i\}_{t\in D}$*, the probability distribution of*

$$(X(t_1+\tau),\ \cdots,\ X(t_k+\tau))$$

is invariant about all shifts $\tau \in T$ *with* $t_i+\tau \in T, 1 \le i \le k$.

It is reported that *translation equivariance* (Ashman et al., 2024a)-bias helps improve performance about datasets sampled from stationary processes, that is, the neural network outputs the same result with the translated data as with the original.

Definition 2 (Translation Equivariance) *A function* $f : \mathbb{R}^d \longrightarrow \mathbb{R}^d$ *is* translation equivariant *if for a translation* T_τ *for a shift* $\tau \in \mathbb{R}^d$.

$$f(T_\tau \mathcal{D}) = T_\tau f(\mathcal{D})$$

Convolution neural networks (CNNs) are naturally translation equivariant due to the sliding window kernel.

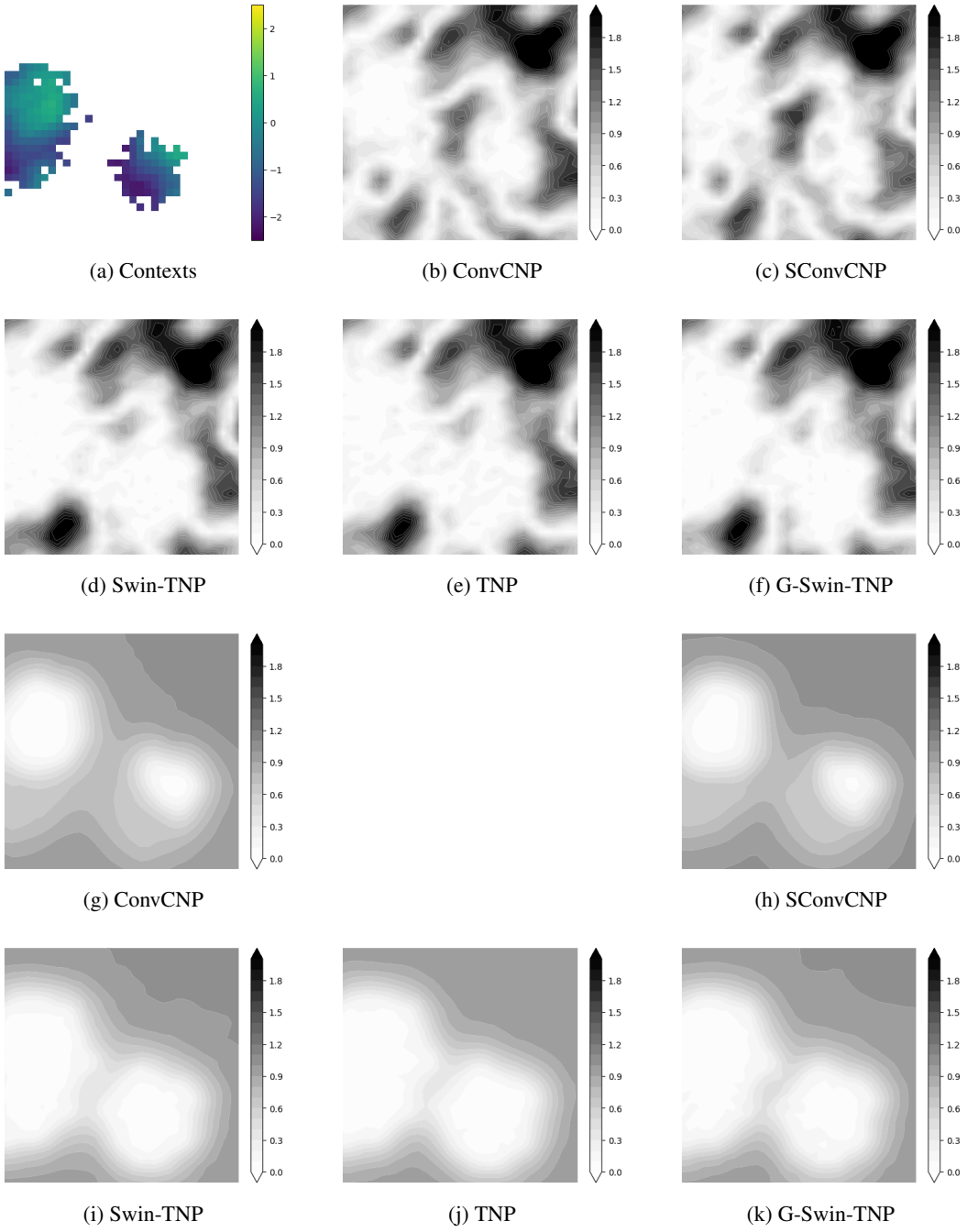
Fourier Neural Operators (FNOs) (Li et al., 2020) encode a function by representing it with a finite series of trigonometric functions, thus encoding data through various frequencies. Because the basis of Fourier series is translation-equivariant, FNO inherits the property. FNOs mix channels and scale frequency components via trainable parameters. However, the number of parameters grows as $\mathcal{O}(k^d)$ with input dimensionality d, making the model memory-intensive in higher dimensions. One important note is that the number of representable frequencies is bounded by the resolution of the training dataset to avoid aliasing as implied by Nyquist sampling theorem (Duruisseaux et al., 2025).

Shifted-window Transformer with Translation Equivariant bias Due to the structure of Swin-Transformer, Swin-TNP cannot be fully translation-equivariant even with translation-equivariant bias Hutchinson et al. (2021); Ashman et al. (2024a). If each input translates a certain distance which is smaller than the window size, the involving input in a window changes so the attention weight also changes. The Swin-Transformer layer is equivariant to translation whose length is a multiple of the window length. The TE-bias adds translation-equivariant property to the tokens within the window.

D RELATED WORKS

D.1 GRID ENCODER AND DECODER

The grid encoder and grid decoder we adopt in our experiments were developed and described by Ashman et al. (2024b). To keep this description self-contained, we summarize them briefly. In the



(a) Contexts (b) ConvCNP (c) SConvCNP

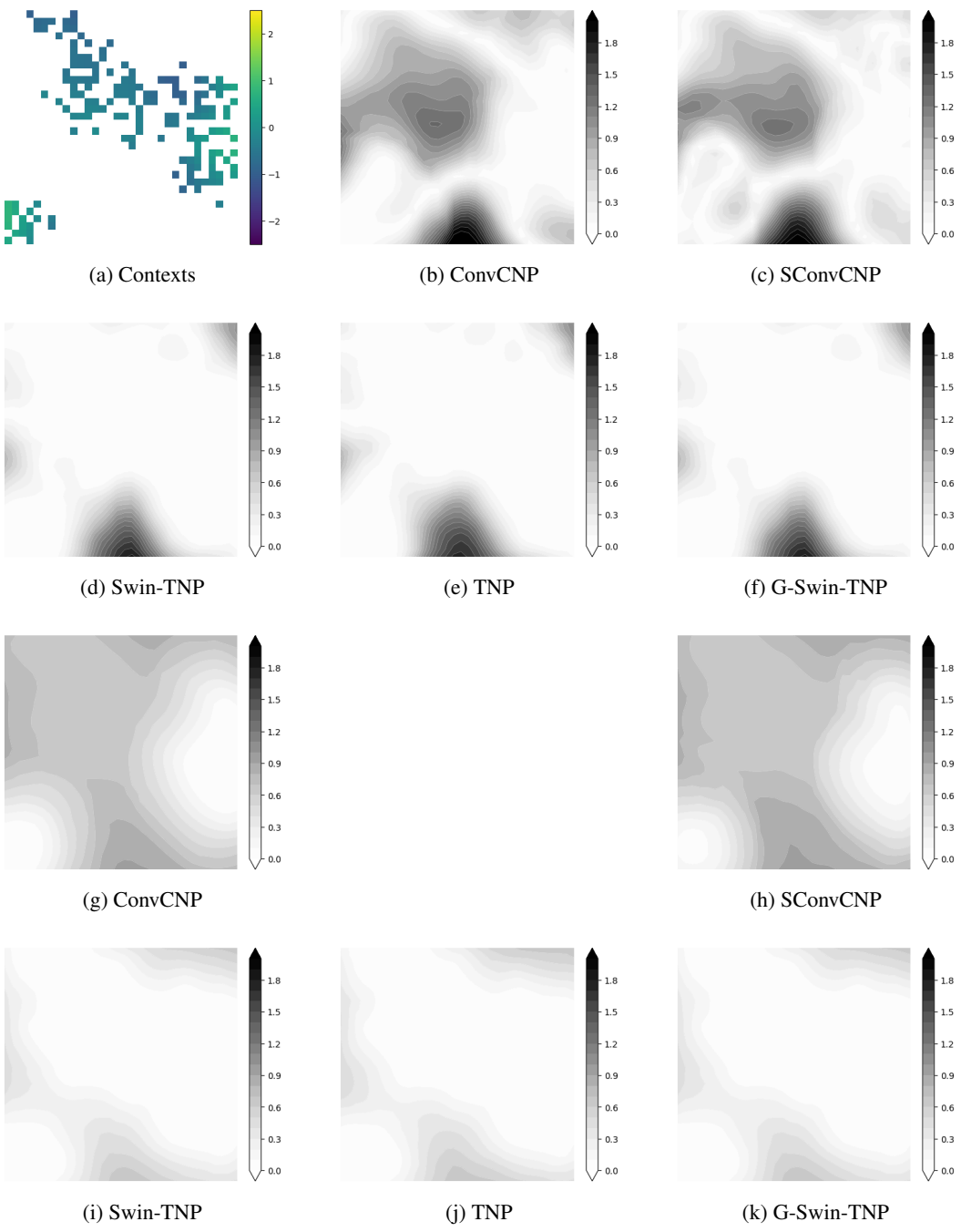
(d) Swin-TNP (e) TNP (f) G-Swin-TNP

(g) ConvCNP (h) SConvCNP

(i) Swin-TNP (j) TNP (k) G-Swin-TNP

Figure 5: Panels (b)–(f) show the root mean square error (RMSE) of each model's prediction given the contexts in panel (a); panels (g)–(k) show the corresponding predicted standard deviation. The realization is sampled from a Gaussian random field with a Matérn-$5/2$ covariance and length scale 0.1.

encoder, each context is assigned to its nearest grid token and serves as a key and value for that grid token. Denote the set of grid tokens by $\{\bar{\boldsymbol{z}}_k\}_{k\in\mathbb{K}}$, the set of context tokens by $\{\boldsymbol{z}_c\}_{c\in\mathbb{C}}$, and the set of target tokens by $\{\boldsymbol{z}'_t\}_{t\in\mathbb{T}}$, where $\mathbb{K}$, $\mathbb{C}$, and $\mathbb{T}$ are the corresponding index sets. Let $\mathfrak{N}_r(\boldsymbol{z})$ denote the indices of the r grid tokens nearest to a token $\boldsymbol{z}$. For brevity, $[\boldsymbol{z}_s;\cdots;\boldsymbol{z}]_{s\in\mathbb{S}}$ denotes the horizontal concatenation of the vectors $\{\boldsymbol{z}_s\}_{s\in\mathbb{S}}$ and $\boldsymbol{z}$ into a matrix of size $d\times(|\mathbb{S}|+1)$, where each $\boldsymbol{z}_s$ and $\boldsymbol{z}$ are d-dimensional column vectors. The grid encoder and grid decoder update the grid and target



(a) Contexts (b) ConvCNP (c) SConvCNP

(d) Swin-TNP (e) TNP (f) G-Swin-TNP

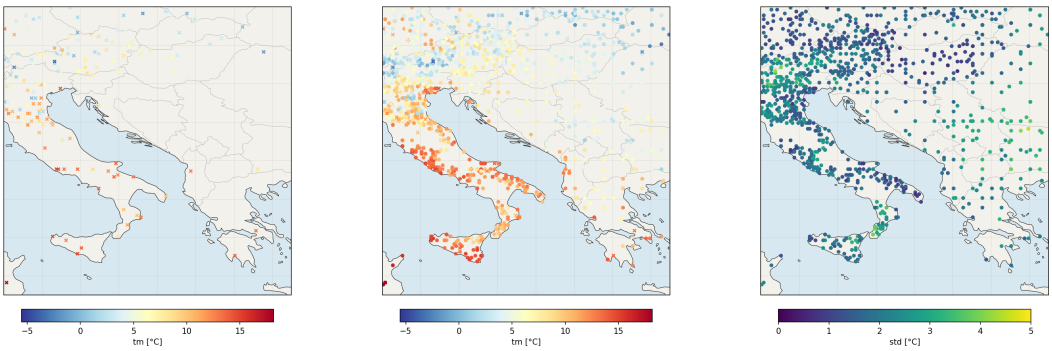
(g) ConvCNP (h) SConvCNP

(i) Swin-TNP (j) TNP (k) G-Swin-TNP

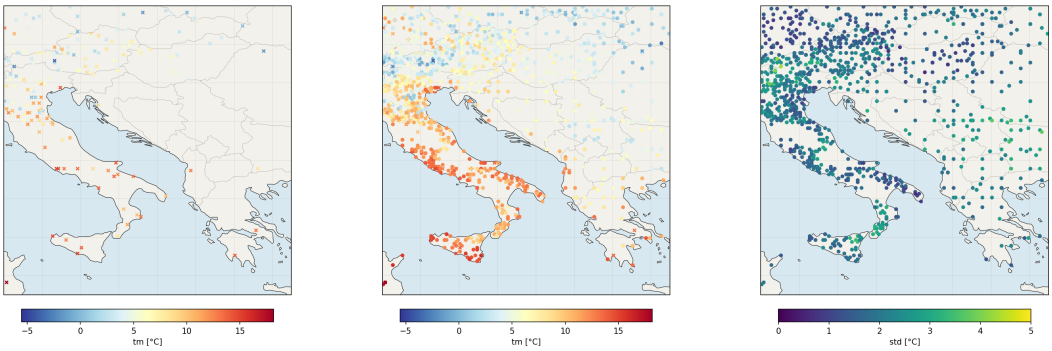
Figure 6: Panels (b)–(f) show the root mean square error (RMSE) of each model's prediction given the contexts in panel (a); panels (g)–(k) show the corresponding predicted standard deviation. The realization is sampled from a Gaussian random field with a Matérn-$5/2$ covariance and length scale 0.2.

tokens, respectively, as follows:

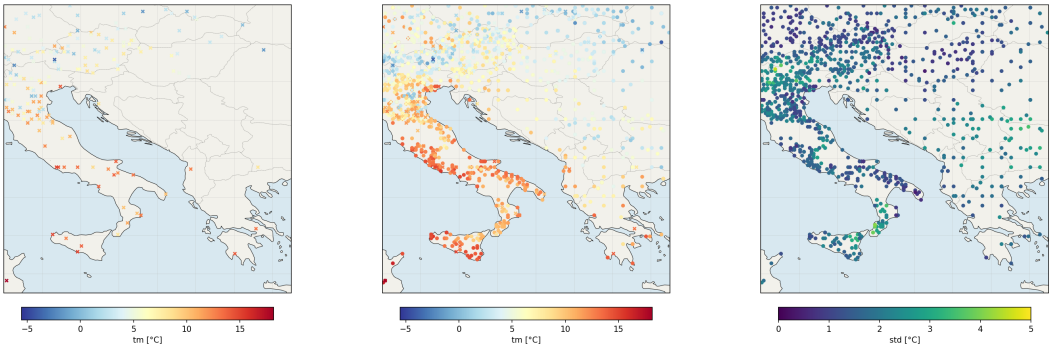
$$\bar{\boldsymbol{z}}_k \leftarrow \sum_{h=1}^{H} \operatorname{softmax}\left(\bar{\boldsymbol{z}}_k W_h^Q W_h^{K\top} [\boldsymbol{z}_c; \cdots; \bar{\boldsymbol{z}}_k]_{c \in \mathbb{C}_k}^\top\right) [\boldsymbol{z}_c; \cdots; \bar{\boldsymbol{z}}_k]_{c \in \mathbb{C}_k} W_h^V W_h^O \tag{5}$$



(a) Swin-TNP



(b) TNP



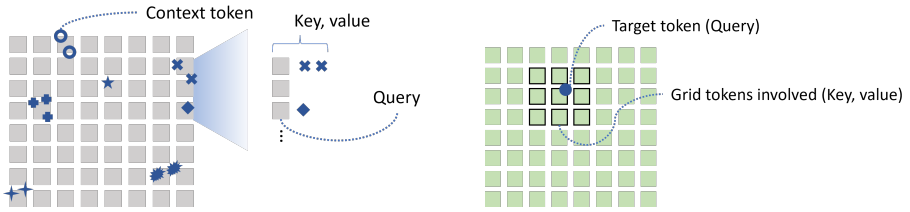
(c) G-Swin-TNP

Figure 7: Predictions on a single day for each model. The left column shows the observed contexts; the middle column shows the mean prediction, with target locations marked by ∘ and context locations by ×; and the right column shows the predicted standard deviation.

where W_h^Q, W_h^K, W_h^V, and W_h^O are trainable parameters and H is the number of heads. $\mathbb{C}_k := \{c \in \mathbb{C} \mid k \in \mathfrak{N}_1(\boldsymbol{z}_c)\}$ is the set of indices of contexts whose nearest grid token is $\bar{\boldsymbol{z}}_k$ The key-value matrix thus concatenates $\bar{\boldsymbol{z}}_k$ with its assigned contexts $\{\boldsymbol{z}_c\}_{c\in\mathbb{C}_k}$.

For the decoder, each target $\boldsymbol{z}'_t$ collects the grid tokens $\bar{\boldsymbol{z}}_k$ with $k \in \mathfrak{N}_r(\boldsymbol{z}'_t)$, which serve as keys and values for updating the target token:

$$\boldsymbol{z}'_t \leftarrow \sum_{h=1}^{H} \mathrm{softmax}\left(\boldsymbol{z}'_t W_h^Q {W_h^K}^\top [\bar{\boldsymbol{z}}_k; \cdots; \boldsymbol{z}'_t]_{k\in\mathfrak{N}_r(\boldsymbol{z}'_t)}^\top\right) [\bar{\boldsymbol{z}}_k; \cdots; \boldsymbol{z}'_t]_{k\in\mathfrak{N}_r(\boldsymbol{z}'_t)} W_h^V W_h^O \tag{6}$$



(a) Grid encoder (b) Grid decoder

Figure 8: Each subfigure shows how the grid encoder or grid decoder computes a grid token or target token, illustrated for a 2-dimensional input. **(a)** The grid encoder collects neighbors for each grid token. The aligned square tokens are grid tokens, and each context token is assigned to its nearest grid token; context tokens are drawn in various shapes, where tokens of the same shape share the same nearest grid token. The tokens collected for a grid token serve as keys and values in the attention mechanism, along with the grid token itself. **(b)** The grid decoder computes a target token. For simplicity, a single target is shown, as a dark round dot. The aligned squares in the background are grid tokens; those near the target, outlined in black, are the 3^2 nearest neighbors that serve as keys and values for the target token (the query) in the grid decoder.

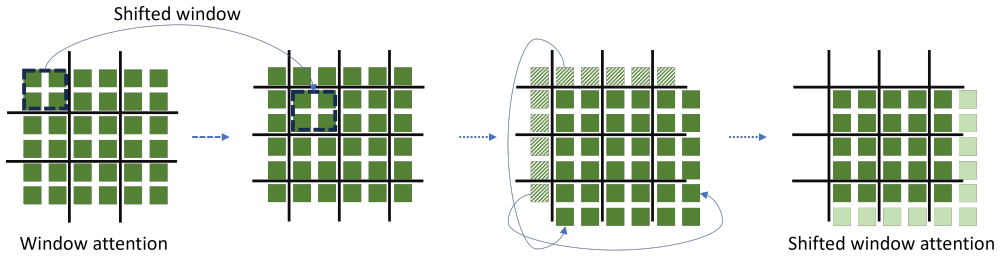


Figure 9: Implementation of the attention in a Swin-Transformer layer. The dark squares are grid tokens, and the black lines partition them into groups (called *windows*); grid tokens in the same partition (window) take part in the same multi-head self-attention. **(Leftmost)** Grid tokens under window attention. **(Second)** The windows are shifted by $(\omega_1/2, \omega_2/2)$, where (ω_1, ω_2) is the window size. **(Third)** Tokens at the edge are rolled to the opposite side to fill the windows. **(Rightmost)** The light-shaded grid tokens are masked tokens.

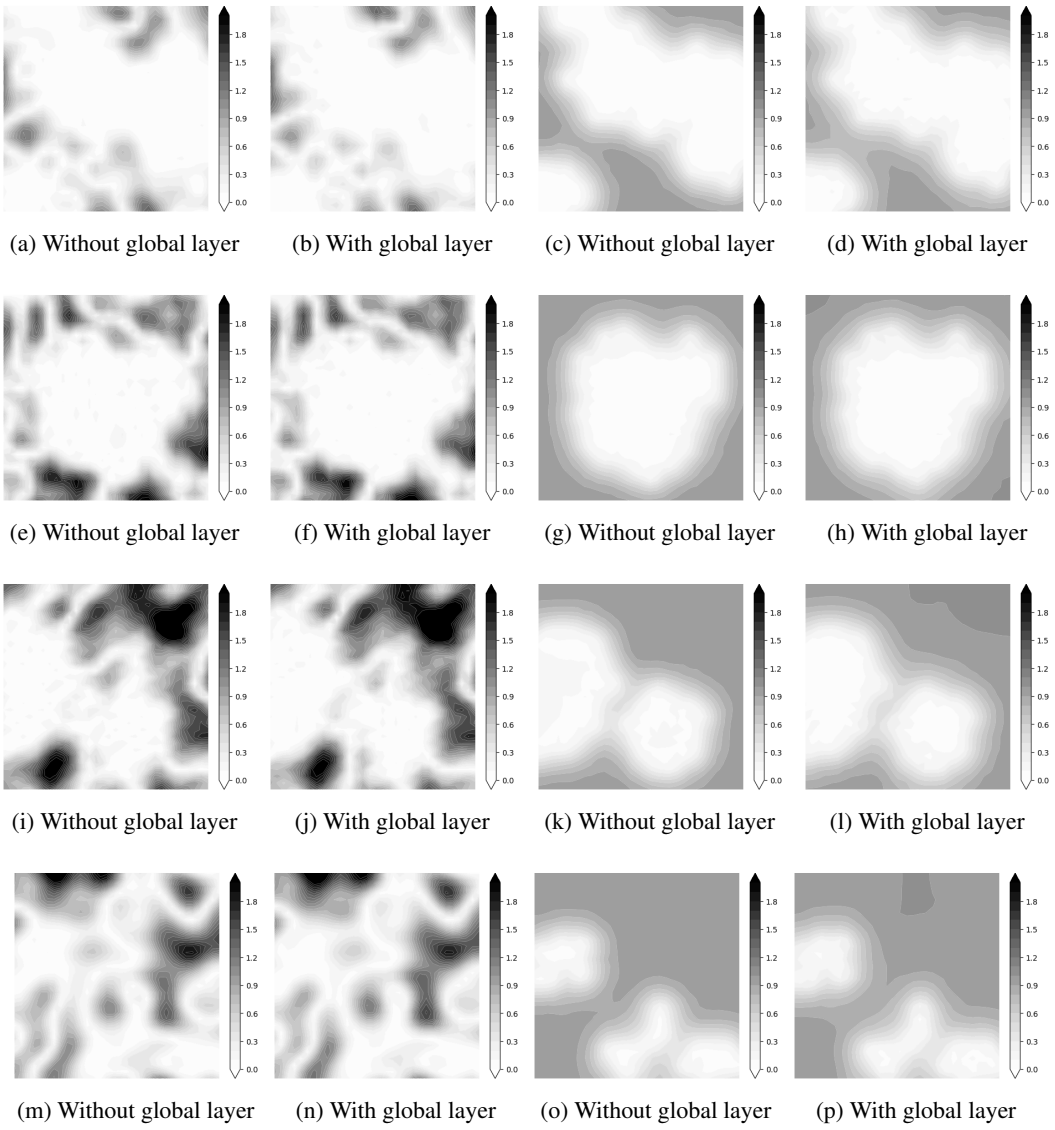
D.2 Swin-Transformer

Swin-Transformers were developed as a transformer-only vision model (Liu et al., 2021). A Swin-Transformer layer is identical to a plain Transformer layer except for the attention. The tokens fed to a Swin-Transformer are arranged in a grid. A Swin-Transformer layer applies attention twice: first, multi-head self-attention within each window, and then multi-head self-attention within the shifted windows, as shown in Figure 9. Each attention is followed by a residual connection and a feedforward layer. In the shifted-window attention, tokens that fall outside the shifted windows are moved to the opposite side and masked. Swin-Transformers assume grid-aligned tokens; since a Gridded Transformer Neural Process (Gridded TNP) arranges its tokens on a grid according to their input locations, Swin-Transformer layers are applicable to Neural Processes (NPs)

D.3 Ablation study of model architecture

As an ablation, we compare the predictions of Growing-Swin-TNP with and without the global layer (Figure 10).

The figures on the first column and those on the second column show contour plots of root mean square error (RMSE) over a task, and the figures on the third column and those on the fourth row show the corresponding standard deviation.



(a) Without global layer (b) With global layer (c) Without global layer (d) With global layer

(e) Without global layer (f) With global layer (g) Without global layer (h) With global layer

(i) Without global layer (j) With global layer (k) Without global layer (l) With global layer

(m) Without global layer (n) With global layer (o) Without global layer (p) With global layer

Figure 10: Ablation of the global layer in Growing-Swin-TNP. Each row shows one task; within a row, the four panels are the RMSE (left two) and the predicted standard deviation (right two), each shown without and with global layer.

Looking at the standard deviation, the one with global layer produces smoother uncertainty near contexts, where the standard deviation and RMSE become close to zero, while the one without global layer shows a more rapid increase in uncertainty away from the contexts. This suggests that the global-layer couples targets with nearby contexts.